IMPROVING THE ROBUSTNESS OF ARTIFICIAL NEURAL NETWORKS VIA BAYESIAN APPROACHES

by

Jun Zhuang

A Dissertation

Submitted to the Faculty of Purdue University

In Partial Fulfillment of the Requirements for the degree of

Doctor of Philosophy

Department of Computer and Information Science

Indianapolis, Indiana

August 2023

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF COMMITTEE APPROVAL

Dr. Mohammad Al Hasan

Department of Computer and Information Science

Dr. Snehasis Mukhopadhyay

Department of Computer and Information Science

Dr. George Mohler

Department of Computer and Information Science

Dr. Mihran Tuceryan

Department of Computer and Information Science

Approved by:

Dr. Shiaofen Fang

To my family and all my friends.

ACKNOWLEDGMENTS

First and foremost, I would like to express my heartfelt gratitude to my Ph.D. advisor, Dr. Mohammad Al Hasan, for his insightful guidance during my Ph.D. study. I'm truly grateful for his unwavering support in my job search. Besides, I would like to extend my sincere thanks to all professors who served as my thesis committee members, namely Dr. Snehasis Mukhopadhyay, Dr. George Mohler, Dr. Mihran Tuceryan, and Dr. Murat Dundar for their wealth of knowledge and constructive suggestions to improve my thesis paper. Also, I would like to thank my M.S. advisor, Dr. Mingchen Gao, for her guidance to bring me into the field of medical imaging.

Furthermore, I would like to thank all of my mentors, Dr. Dali Wang, Dr. Siva Chittajallu, and Dr. Qi Wei for their great guidance during my internship at the University of Tennessee, Knoxville, Roche Diabetes Care, Inc., and Uber Technologies, Inc., respectively. Their mentorships have enriched my academic and industrial pursuits. I couldn't successfully complete my challenging internship projects without their instructions.

Last but not least, I want to express my deepest gratitude to my family for their unconditional love, unwavering support, and unchanging encouragement throughout my academic endeavor. Their belief and sacrifices have been the foundation of my success. Their presence and understanding have given me the strength to overcome challenges in my academic journey.

TABLE OF CONTENTS

LIST OF TABLES 9

LIST OF FIGURES 11

ABSTRACT 13

1 INTRODUCTION 14

1.1 Overview 14

1.2 Notations and Preliminaries for Bayesian Label Transition 17

2 RELATED WORKS 19

2.1 Artificial Neural Networks 19

2.1.1 Graph Neural Networks 20

2.1.2 Generative Adversarial Networks 20

2.2 Learning with Noisy Labels 21

2.3 Open-set Learning 22

3 DEPERTURBATION OF ONLINE SOCIAL NETWORKS VIA BAYESIAN LABEL TRANSITION 23

3.1 Introduction 23

3.2 Methodology 25

3.2.1 Bayesian Label Transition 26

3.2.2 GraphLT Algorithm and Pseudo-code 28

3.3 Experiments 30

3.3.1 Datasets 30

3.3.2 Competing Defending Models 32

3.3.3 Experimental Settings 33

3.3.4 Hyper-parameters of Our Model 34

3.3.5 Node Classification Benefits from GraphLT 35

3.3.6 Comparison of The Defense 37

3.3.7 Analysis of Parameters 38
3.3.8 Ablation Study 40
3.3.9 Limitation and Future Improvement 40
3.4 Chapter Summary 40

4 BAYESIAN SELF-SUPERVISION AGAINST DYNAMIC GRAPH PERTURBATIONS 43
4.1 Introduction 43
4.2 Methodology 45
4.3 Experiments 48
4.3.1 Experimental Settings 48
4.3.2 Implementation of Dynamic Graph Perturbations 49
4.3.3 Competing Defending Methods 49
4.3.4 GraphSS Defends Node Classifiers 51
4.3.5 GraphSS Can Alert Dynamic Graph Perturbations 53
4.3.6 Analysis of Runtime 54
4.3.7 Analysis of Parameters 54
4.3.8 Ablation Study 55
4.3.9 Limitation and Future Improvement 56
4.4 Chapter Summary 57

5 ROBUST NODE CLASSIFICATION ON GRAPHS: JOINTLY FROM BAYESIAN LABEL TRANSITION AND TOPOLOGY-BASED LABEL PROPAGATION . . 60
5.1 Introduction 60
5.2 Methodology 63
5.2.1 Problem Statement 63
5.2.2 Asymmetric Dirichlet Distributions 64
5.2.3 Label Inference Jointly from Bayesian Label Transition and Topology-based Label Propagation 65
5.2.4 Pseudo-code of Our Proposed Model 69
5.3 Experiments 71

5.3.1 Datasets 71
5.3.2 Implementation of Topological Perturbations 72
5.3.3 Competing Methods 72
5.3.4 Evaluation Metrics 74
5.3.5 Node Classification Benefits from Our Model against Topological Perturbations 75
5.3.6 Empirical Analysis of Convergence 77
5.3.7 Comparison with Competing Methods 78
5.3.8 Analysis of Parameters 78
5.3.9 Limitation and Future Directions 79
5.4 Chapter Summary 80

6 NON-EXHAUSTIVE LEARNING USING GAUSSIAN MIXTURE GENERATIVE ADVERSARIAL NETWORKS 83
6.1 Introduction 83
6.2 Background 86
6.2.1 Generative Adversarial Networks (GAN) 86
6.2.2 Bidirectional Generative Adversarial Networks (BiGAN) 86
6.3 Methodology 87
6.3.1 Offline Training: Computing Multi-modal Prior Distribution 88
6.3.2 Extracting Potential Unknown Class 89
6.3.3 Estimating The Number of Emerging Class 90
6.4 Experiments 93
6.4.1 Dataset 93
6.4.2 Competing Methods 94
6.4.3 Evaluation Metrics 95
6.4.4 The Capability of Unknown Class Extraction 96
6.4.5 The Estimation of The Number of New Classes 97
6.4.6 Study of User-defined Parameters 98
6.4.7 Reproducibility of The Work 99

6.5 Chapter Summary 100

7 SUMMARY 101

REFERENCES 103

VITA 119

LIST OF TABLES

3.1 Statistics of Seven Benchmark Datasets ($|\mathcal{V}|$, $|\mathcal{E}|$, $|F|$, $|C|$, $\#Iter.$ denotes the number of nodes, edges, features, classes, and training epochs, respectively. $Avg.D$ denotes the average degree of test nodes.) 31

3.2 Hyper-parameters of GRAND in This Chapter for Small (Cora, Citeseer), Medium (AMZcobuy, Coauthor), and Large graphs (KDD20(S1), KDD20(S2), and Reddit) 33

3.3 Hyper-parameters of GNNs (#Hidden denotes the number of neurons in each hidden layer of GNNs.) 35

3.4 Model Architecture of GNNs 35

3.5 The Comparison of Defending Performance between GraphLT (nr=0.1) and Competing Defenders 36

3.6 Analysis of Average ($\mu \pm \sigma$) and Unit (Per 100 nodes) Runtime for GraphLT Inference with Different WS (nr=0.1) 39

3.7 The Investigation of How GraphLT Reverses The Performances Decline Caused by Training Node Classifiers f_θ with Noisy Labels under Different Noise Ratio nr (**Orig.** denotes the original accuracy. **LT** denotes the accuracy after label transition.) 41

3.8 The Examination of The Generalization of GraphLT over Three Classic Node Classifiers (nr=0.1) 42

4.1 Statistics of Datasets ($|\mathcal{V}|$, $|\mathcal{E}|$, $|F|$, and $|C|$ denote the number of nodes, edges, features, and classes, respectively. Avg.D denotes the average degree of test nodes.) 48

4.2 Hyper-parameters of GRAND in This Chapter 51

4.3 Runtime (s) Comparison among Five Methods on Cora 54

4.4 The Comparison of Defending Results (Test accuracy) between GraphSS and The Competing Methods (Original/Attack denote the test accuracy before/after the non-target attacks ($L\&F$).) 58

4.5 Analysis of The Average Runtime and The Unit Runtime (Per 100 nodes) in Seconds of GraphSS between Defense and Alert 59

5.1 Statistics of Datasets ($|\mathcal{V}|$, $|\mathcal{E}|$, $|F|$, and $|C|$ denote the number of nodes, edges, features, and classes, respectively. Avg.D denotes the average degree of test nodes. EHR denotes the edge homophily ratio.) 71

5.2 Hyper-parameters of DropEdge in This Chapter 74

5.3 Examination of Our Model on Top of GCN under Three Scenarios of Topological Perturbations across Five Datasets ("Original" denotes the original performance of GCN on perturbed graphs (Before inference). GraphSS is our baseline model with a fixed α value, 1.0. "Vanilla" denotes that we dynamically update the α vector using Gibbs samplers. "Random", "Major", and "Degree" denote that we employ the corresponding topology-based label sampler based on our vanilla architecture instead of using Gibbs samplers. All methods are evaluated by classification accuracy (Acc.) and average normalized entropy (Ent.) on victim nodes.) 81

5.4 Comparison between Competing Methods and Our Model under The Random Perturbations Scenario (Acc. (%) denotes classification accuracy. Ent. (%) denotes the average normalized entropy. Time (s) denotes total runtime.) 82

6.1 The Background of Related Tasks (Conv. for conventional method) 84

6.2 Statistics of Datasets (#Inst. denotes the number of instances; #F. denotes the number of features after one-hot embedding or dropping for network intrusion dataset; #C. denotes the number of classes.) 95

6.3 The F1-score of Four Models for UCs Extraction 96

6.4 F1 Score from Our Proposed Model by Using Different Prior 97

6.5 The S-R^2 between NE-GM-GAN and Baselines on Four Datasets (We denote "UCs" as the number of unknown classes in this table.) 97

6.6 Test of Three-sigma Rule (%) 98

6.7 Model Architectures 99

LIST OF FIGURES

3.1 The Diagram of Bayesian Label Transition ($\mathcal{V}$, $\mathcal{Z}$, and $\mathcal{Y}$ denote the nodes, latent labels, and noisy labels, respectively. ϕ denotes the conditional label transition matrix. α denotes the Dirichlet parameter. N and K denote the number of nodes and classes, respectively.) 25

3.2 The Workflow of Bayesian Label Transition (GraphLT) (The left-hand side represents online social networks (OSN). Each human figure denotes a user of OSN. The dashed line between two users denotes the virtual connection in OSN. Note that the location of each user does not indicate the real location on Earth. The latent labels represent the ground-truth labels of the nodes, which cannot be observed (White color). The noisy labels are manually annotated, which can be observed (Gray color).) 28

3.3 An Example of Non-malicious Perturbations (The left-hand side presents the OSN before perturbation (unperturbed environment). The node classifier can classify normal users into the correct group. The right-hand side shows the OSN after perturbation. Perturbators randomly connect with many other users and disturb the accuracy of the node classifier, causing incorrect classification. The red dashed line denotes the connection between the normal user and the perturbator.) . . . 32

3.4 The Confusion Matrices (Heatmap) of The Defending Results on KDD20(S1) and Reddit for GraphLT (nr=0.1) (We apply log-scale to the confusion matrix for fine-grained visualization.) 37

3.5 Analysis of The Number of Warm-up Steps WS for GraphLT Inference (nr=0.1) 38

3.6 Ablation Study of GraphLT over Three Node Classifiers on KDD20(S1) ({Fixed ϕ, Dynamic ϕ} denote whether we dynamically update the transition matrix ϕ in each iteration.) 39

4.1 The Workflow of Bayesian Self-Supervision Model, GraphSS (The latent labels (White color) represent the unobserved ground-truth labels whereas the auto-generated labels (Gray color) are observed noisy labels.) 46

4.2 The Confusion Matrices (Heatmap) of The Defending Result on Cora by GraphSS (We apply log-scale to the confusion matrix for fine-grained visualization.) . . . 52

4.3 The Assessment of Alerting Dynamic Graph Perturbations by GraphSS via ROC Curve 53

4.4 Analysis of The Number of Warm-up Steps WS (Blue) and The Number of Retraining Epochs *Retrain* (Red) for GraphSS 55

4.5 Ablation Study of GraphSS ({Retrain, No Retrain} denote whether we retrain the node classifier in each iteration. {Fixed ϕ, Dynamic ϕ} denote whether we dynamically update the transition matrix ϕ in each iteration.) 56

5.1 Toy Examples of Dirichlet Distributions 65

5.2 Toy Example of Our Proposed Sampling Method 67

5.3 Toy Examples of Our Three Proposed Samplers (We use red, yellow, and blue colors to represent three classes.) 68

5.4 Node Label Distributions of Cora (Test nodes) 75

5.5 Visualization of Node Label Distributions on Citeseer via Log-scale Fine-grained Confusion Matrices 76

5.6 Empirical Analysis of Convergence between Gibbs Sampler and Topology-based Label Sampler (Major) 77

5.7 Analysis of The Initial α Value for The Dynamic α Vector 79

6.1 The Model Architecture of NE-GM-GAN (Left-hand side) and The Workflow of I-means in Algorithm (6) (Right-hand side) 87

6.2 Comparison on The Estimation of New Emerging Class among Three Methods . 98

6.3 Investigation on The Number of Epochs in The Warm-up Stage (WS) for I-means on Four Datasets 99

ABSTRACT

Artificial neural networks (ANNs) have achieved extraordinary performance in various domains in recent years. However, some studies reveal that ANNs may be vulnerable in three aspects: label scarcity, perturbations, and open-set emerging classes. Noisy labeling and self-supervised learning approaches address the label scarcity issues, but most of the work couldn't handle the perturbations. Adversarial training methods, topological denoising methods, and mechanism designing methods aim to mitigate the negative effects caused by perturbations. However, adversarial training methods can barely train a robust model under the circumstance of extensive label scarcity; topological denoising methods are not efficient on dynamic data structures; and mechanism designing methods often depend on heuristic explorations. Detection-based methods devote to identifying novel or anomaly instances for further downstream tasks. Nonetheless, such instances may belong to open-set new emerging classes. To embrace the aforementioned challenges, we address the robustness issues of ANNs from two aspects. First, we propose a series of Bayesian label transition models to improve the robustness of Graph Neural Networks (GNNs) in the presence of label scarcity and perturbations in the graph domain. Second, we propose a new non-exhaustive learning model, named NE-GM-GAN, to handle both open-set problems and class-imbalance issues in network intrusion datasets. Extensive experiments with several datasets demonstrate that our proposed models can effectively improve the robustness of ANNs.

1. INTRODUCTION

1.1 Overview

Artificial neural networks (ANNs), also known as deep learning (DL), are a subset of machine learning algorithms that model biological neural activities in the brain of human beings [1]. By simulating such activities, ANNs can accomplish various learning tasks in multiple domains, such as images [2], language [3], and graphs [4], where ANNs have already achieved extraordinary performance in recent years. For example, conventional convolutional neural networks are widely used in large-scale image processing [5]. Generative Adversarial Networks (GANs) are kinds of popular generative models that yield high-quality in instance generation, such as image synthesis [6]. Another example is Graph Neural Networks (GNNs), which are widely used for graph representation learning [4].

Regardless of the great success of ANNs, robustness is still a crucial factor to ensure their performance in real-world applications [7]. Some studies reveal that ANNs may be vulnerable under certain circumstances [8]–[10]. For example, carefully crafted adversarial samples may seriously deteriorate the performance of ANNs [11]. In this thesis, we focus on three kinds of scenarios that weaken the robustness of ANNs. The first scenario is label scarcity, which refers to a situation that trains a supervised or semi-supervised ANN without sufficient labeled instances, which are usually manually annotated with tags for relevant tasks [8]. The label scarcity issue is common in the real world for the following reasons. First, human annotation unavoidably introduces noise [12]. Second, acquiring labeled data is difficult, expensive, and time-consuming in some domains, such as medical diagnosis [13], [14]. Due to these reasons, it is essential to address the label scarcity issue. The second scenario is perturbations, which refer to the deliberate modification of input data that causes the ANN to produce inferior outputs [9]. The perturbation in this dissertation includes non-malicious perturbation, such as random perturbations [15], and malicious perturbation, such as adversarial attacks [11]. The third scenario is novel classes in the test data, which is usually shown in the open-set learning paradigm [10]. Compared to conventional close-set learning, which assumes all classes are known in both train and test datasets, open-set learning refers to a problem that trains a model to recognize instances that may belong to

unseen classes. This problem is particularly relevant in real-world applications, where novel classes may emerge at any time. For example, network intrusion behaviors may be novel to the existing behaviors recorded in the database [16]. In this thesis, we mainly address the first two scenarios in the graph domain and investigate the novel-class issue in the open-set learning environment.

Extensive studies have been pursued to improve the robustness of ANNs under the aforementioned scenarios. We categorize such efforts into three groups. The first group of research aims to address label scarcity issues. Within this group in the graph domain, some works mitigate the issues with the help of noisy labels [17], whereas other works utilize the self-information of the instances instead [18], [19]. Most of these works focus on semi-supervised learning tasks and can't handle perturbations on graphs. The second group of research in the graph domain can be further divided into three sub-groups. First, the adversarial training methods [20]–[22] improve the robustness of GNNs by training with adversarial samples. However, these methods can barely train a robust model under the circumstances of extensive label scarcity. Second, topological denoising methods [23], [24] prune suspicious edges on the graph in the preprocessing stage. Nonetheless, such methods are not efficient on dynamic graphs, as the graph structure is always changing over time. Third, mechanism designing methods [25]–[29] mainly propose a message-passing mechanism, a.k.a. graph convolutional operator or node aggregator, to better classify the nodes. Such a mechanism sometimes extensively depends on heuristic explorations. The third group of research [30]–[32] devotes to detecting the novel or anomaly instances for further downstream tasks, such as re-classifying the novel instances or removing the anomaly instances. However, such instances may belong to non-exhaustive emerging classes.

To embrace the above-mentioned challenges, in this thesis, we utilize Bayesian approaches to address the robustness issues of ANNs. Specifically, we propose Bayesian label transition models to improve the robustness of Graph Neural Networks (GNNs) in the presence of label scarcity and perturbations in the graph domain [33]–[35]. Also, we propose a new online non-exhaustive learning model, namely Non-Exhaustive Gaussian Mixture Generative Adversarial Networks (NE-GM-GAN), to address the novel-class issue in network intrusion detection [16].

Overall, our contribution in this thesis can be summarized as follows. In this thesis, we address the robustness issues of ANNs from two distinct perspectives. On one hand, we propose a series of Bayesian label transition models to improve the robustness of Graph Neural Networks (GNNs) in the presence of label scarcity and perturbations in the graph domain [33]–[35]. Specifically, we first propose a new Bayesian label transition model, namely GraphLT, to enhance the performance of node classifications on online social networks under both unperturbed and perturbed environments [33]. This enhancement is generalizable over three classic GNNs and across various sizes of attributed graphs. We further generalize noisy supervision as a subset of self-supervised learning methods and propose a new Bayesian self-supervision model, namely GraphSS, to improve the robustness of the node classifier against adversarial attacks on the label-scarce dynamic graph [34]. GraphSS is proven to 1) effectively recover the prediction of a node classifier against dynamic graph perturbations, and 2) affirmatively alert such perturbations. Moreover, we address the slow convergence issues with our new label inference mechanism, LInDT, which is proven to help the Bayesian label transition converge faster than that using Gibbs sampling [35]. In this work, we also release the constraint of symmetric Dirichlet distributions on the Bayesian label transition and verify that dynamically updating the α vector (Dirichlet prior) can contribute to better label inference.

On the other hand, we propose a new non-exhaustive learning model, named NE-GM-GAN, that can detect novel classes in online test data accurately and defy the class imbalance problem effectively [16]. This model integrates Bayesian inference with distance-based and threshold-based methods to estimate the number of emerging classes in the test data. It also devises a novel scoring method to distinguish the UCs (unknown classes) from the KCs (known classes). Extensive experiments demonstrate that this model is superior to existing methods for accurate and robust online detection of emerging classes in streaming data over four datasets (three real and one synthetic).

The organization in this thesis is as follows. In Chapter 1, we introduce the background, the contributions, and the organization of this thesis. We also present the notations and preliminaries for our proposed Bayesian label transition models. In Chapter 2, we first introduce the development history of Artificial Neural Networks (ANNs), along with two

subsections about Graph Neural Networks (GNNs) and Generative Adversarial Networks (GANs). Furthermore, we discuss related works about Noisy-label Learning and Open-set Learning. In subsequent chapters, we present a series of Bayesian label transition models from Chapter 3 to Chapter 5 and verify that our proposed model can effectively improve the robustness of Graph Neural Networks (GNNs) in the presence of label scarcity and perturbations. In Chapter 6, we propose a new model, NE-GM-GAN, for the online detection of novel classes under the open-set environment. In the end, we summarize our works in Chapter 7.

1.2 Notations and Preliminaries for Bayesian Label Transition

In this thesis, an undirected attributed graph, e.g., an Online Social Network (OSN), is denoted as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{v_1, v_2, ..., v_N\}$ denotes the set of vertices (users), N is the number of vertices in $\mathcal{G}$, and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ denotes the set of edges between vertices (connections among users). We denote $\mathbf{A} \in \mathbb{R}^{N \times N}$ as the symmetric adjacency matrix and $\mathbf{X} \in \mathbb{R}^{N \times d}$ as the feature matrix (contains users' profile information), where d is the number of features for each vertex. We assume that all ground-truth labels, hereby referred as **latent labels** of the vertices $\mathcal{Z} \in \mathbb{R}^{N \times 1}$, are unobserved. We argue that manual annotation could be a potential solution to this problem but human annotation unavoidably introduces noises [12]. Another potential solution is to use a trained node classifier to label the vertices with pseudo-labels. This solution also yields noisy labels. We use $\mathcal{Y} \in \mathbb{R}^{N \times 1}$ to denote the manual-annotated **noisy labels**, which are observed for all nodes (train and test). In the Bayesian Self-supervision model, the noisy labels include both manual-annotated labels $\mathcal{Y}_m \in \mathbb{R}^{N_{train} \times 1}$ and auto-generated labels $\mathcal{Y}_a \in \mathbb{R}^{N_{test} \times 1}$, where N_{train} and N_{test} denote the number of nodes on the train and test graphs, respectively. Our task is to defend GNN-based node classification when its noisy labels (observed) deviate from its latent labels (unobserved). However, we assume that the entries of both $\mathcal{Y}$ and $\mathcal{Z}$ take values from the same closed category set. Below, we first discuss a variant of graph neural networks (GNNs) that we consider for our task. The most representative GNN proposed by Kipf and

Welling [4], Graph Convolutional Networks (GCNs), is our preferred variant. The layer-wise propagation of the GCN is presented as follows:

$$\mathbf{H}^{(l+1)} = \sigma\left(\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{H}^{(l)}\mathbf{W}^{(l)}\right). \tag{1.1}$$

In Equation (1.1), $\tilde{\mathbf{A}} = \mathbf{A}+I_N$, $\tilde{\mathbf{D}} = \mathbf{D}+I_N$, where I_N is the identity matrix and $\mathbf{D}_{\mathrm{i,i}} = \sum_{\mathrm{j}} \mathbf{A}_{\mathrm{i,j}}$ is the diagonal degree matrix. $\mathbf{H}^{(l)} \in \mathbb{R}^{N\times d}$ is the nodes hidden representation in the l-th layer, where $\mathbf{H}^{(0)} = \mathbf{X}$. $\mathbf{W}^{(l)}$ is the weight matrix in the l-th layer. $\sigma(\cdot)$ denotes a non-linear activation function, such as ReLU.

In the K-class node classification task, we denote $\mathbf{z}_n$ as the latent label of node v_n and $\mathbf{y}_n$ as the corresponding noisy label ($\mathbf{y}_{nk}$ is a one-hot vector representation of $\mathbf{y}_n$). In our setting, the GCN can be trained by the following loss function $\mathcal{L}$:

$$\mathcal{L} = -\frac{1}{N}\sum_{n=1}^{N}\mathcal{L}(\mathbf{y}_n, f_\theta(v_n)) = -\frac{1}{NK}\sum_{n=1}^{N}\sum_{k=1}^{K}\mathbf{y}_{nk}\ln f_\theta(v_{nk}), \tag{1.2}$$

where $f_\theta(\cdot) = softmax(\mathbf{H}^{(l)})$ is the prediction of node classifier parameterized by θ.

Our idea is to improve the performance of the node classifier by Bayesian label transition, in which a transition matrix of size $K \times K$ is learned, which reflects a mapping from $\mathcal{Y}$ to $\mathcal{Z}$. We represent such a matrix by ϕ (the same term is also used to denote a label-to-label mapping function). Under the presence of ϕ, the loss function $\mathcal{L}$ in Equation (1.2) can be rewritten as follows:

$$\mathcal{L} = -\frac{1}{N}\sum_{n=1}^{N}\mathcal{L}(\mathbf{y}_n, \phi^{-1}\circ f_\theta(v_n)). \tag{1.3}$$

However, learning accurate ϕ is a difficult task as the latent labels of the nodes are not available. So, in our method, the ϕ is iteratively updated to approximate the perfect one by using the Bayesian framework.

Note that we use distinct notations in the chapter about the novel-class detection in the open-set environment (Chapter 6).

2. RELATED WORKS

In this chapter, we first introduce the development of artificial neural networks, including graph neural networks and generative adversarial networks. Besides, we present the related literature about noisy-label learning and open-set learning.

2.1 Artificial Neural Networks

The concept of artificial neural networks (ANNs) originated from the study of how neurons behave in the brain. To simulate such intelligent behaviors, researchers construct the basis of neural networks using interconnected circuits. This kind of study is also known as "connectionism". In 1958, psychologist Frank Rosenblatt proposed a perceptron model for linear classification [36]. A single-layer perceptron model is a simple feedforward neural network that consists of one single output layer. This perceptron model could take multiple inputs. For each input, the model assigns a corresponding weight. The model then summarizes multiple weighted inputs with an activation function on top of this summation. The activation functions, such as Sigmoid [37], ReLU [38], and Leaky ReLU [39], are widely used in neural networks for deciding whether a neuron should be activated or not. Furthermore, the multi-layer perceptron model (MLP) is developed from the single-layer perceptron model by adding multiple hidden layers within the neural networks. Each hidden layer includes multiple neurons that connect to the neurons in the next layer. MLP is very practical since this model can learn non-linear representations via supervised-learning approaches.

Compared to MLP, the Boltzmann machine is a type of stochastic neural network that borrows concepts from statistical physics and is widely used in the unsupervised learning paradigm [40]. In 1986, Hinton et al. improved this model to an advanced version, named the restricted Boltzmann machine, by skipping the interconnections within each layer [41]. In the same year, David Rumelhart, Geoffrey Hinton, and Ronald Williams proposed a famous back-propagation algorithm to learn the errors in neural networks for effective training [42]. This algorithm is still widely used in the training of neural networks nowadays.

Another variation of MLP is known as Convolutional Neural Networks (CNNs), which achieve extraordinary performance on image classification. For example, Yann LeCun et

al. proposed the LeNet to recognize the hand-written digits [5]. In 2012, Krizhevsky et al. proposed the AlexNet and won the competition in ISLVRC 2012 [43]. These achievements opened the prelude to the golden decade of neural networks, a.k.a., deep learning.

2.1.1 Graph Neural Networks

Graph neural networks (GNNs) generalize conventional convolutional neural networks (CNNs) to graph data and achieve great success in the recent few years [44]. Bruna et al. first generalize convolutions on graph data from the perspective of both spatial method and spectral method [45]. However, the eigendecomposition of the graph Laplacian matrix leads to high complexity on a large graph. To improve efficiency, Defferrard et al. introduce the K-order Chebyshev polynomial to approximate spectral filter [46]. Kipf and Welling limit the graph convolution to a 1-order polynomial and achieve state-of-the-art performance with high efficiency [4]. Wu et al. further simplify the graph convolution by successively removing nonlinearities and collapsing weight between consecutive layers without a negative impact on accuracy [47]. Different from the aforementioned spectral methods, Hamilton et al. propose a spatial model that aggregates features from fixed-size local neighbors of the current node [48]. Veličković et al. leverage a masked self-attention strategy to aggregate neighborhoods information in both transductive and inductive manners [49]. Du et al. explore a K-localized filter on the spatial domain with much lower complexity against spectral methods [50]. From the perspective of the Weisfeiler-Lehman (WL) test, Xu et al. perform extensive experiments to investigate the performance of different types of aggregators [51]. In brief, GNNs are fairly a new and promising direction in graph representation learning.

2.1.2 Generative Adversarial Networks

Generative Adversarial Networks (GANs) were first introduced by Goodfellow et al. in 2014 [6]. The idea of GANs is to train two neural networks to generate instances. One neural network, a generator, is designed to generate fake data, whereas another neural network, a discriminator, is used to distinguish real data and fake data. The original GAN provided a novel solution for generative modeling but it had some limitations, such as instability during

training and difficulty in generating high-quality instances. To strengthen training stability, Arjovsky et al. employed the Wasserstein distance as a loss function to address mode collapse issues during training in the original GANs [52]. Donahue et al. enhance the training stability via an additional encoder during the training stage [53]. To improve the generation quality, in 2015, Radford et al. proposed a Deep Convolutional GAN (DCGAN) that utilizes deep convolutional neural networks in both the generator and discriminator networks [54]. In 2016, Salimans et al. further investigate to improve GAN in semi-supervised learning and image generation [55]. In 2017, Zhang et al. proposed the StackGAN to generate high-resolution images using a two-stage process [56]. Since then, GAN-based models have been widely applied for image generation. For example, CycleGAN was proposed to translate images across two domains using a cycle-consistency loss [57]. StyleGAN was proposed to generate high-resolution and diverse images by a multi-scale generator architecture [58]. BigGAN was introduced to generate high-quality images at a large scale than the previous models [59]. Overall, GAN-family models have become a crucial technique in generative modeling.

2.2 Learning with Noisy Labels

In the past few years, an increasing number of studies learns the deep-learning networks with noisy labels [60]–[62]. Sukhbaatar et al. introduce an extra noise transition matrix to adjust the networks output by noisy supervision [63]. Subsequent improvement from [12] considers noise, not only conditioning on the input image but also conditioning on the human annotation. Yao et al. propose a dynamic label regression framework that improves the prediction by embedding the noise transition into Dirichlet-distributed space [64]. Those works achieve great improvement in conventional CNNs. From the perspective of GNNs, NT et al. present a loss correction approach to handle the graph noisy label [65]. Zhong et al. employ GCNs as a label noise cleaner to acquire clean labels [66]. Both works attempt to filter noises and then acquire cleaner labels. In this thesis, we leverage noisy labels in Bayesian approaches to improve the robustness of GNNs.

2.3 Open-set Learning

Open-set learning is a subset of machine learning that recognizes unknown classes in the test data. The idea of open-set learning was introduced by Scheirer et al., who defined the problem where the test data may contain instances that belong to novel classes which are not present in the training data [67]. According to [68], open-set learning models can mainly be categorized into two types, discriminative and generative. The first type includes SVM-based methods [69] and distance-based method [70], [71]. A collection of recent open-set learning works venture towards the generative direction [72]–[75]. A subset of open-set learning methods, named non-exhaustive learning, mainly employs Bayesian methods, such as the infinite Gaussian mixture model (IGMM) [76] to learn the unknown classes. For example, Zhang et al. [77] use a non-parametric Bayesian framework with different posterior sampling strategies, such as one-sweep Gibbs sampling, for detecting novel classes in online name disambiguation. However, IGMM-type methods can only handle small datasets that follow Gaussian distribution. To address this issue, we propose a novel algorithm that can achieve high accuracy on the large sparse dataset, which does not necessarily follow the Gaussian distribution [16].

3. DEPERTURBATION OF ONLINE SOCIAL NETWORKS VIA BAYESIAN LABEL TRANSITION

3.1 Introduction

In online social networks (OSNs), such as, Facebook and Twitter, a common exercise is user profiling, which clusters users into different groups based on their interests and online behavior. Such clusters are typically annotated so that a supervised node classification model can be built, which can subsequently be used for advertisement, product recommendation, and promotion offer generation. However, a small number of OSN users, so-called perturbators, often modify the network arbitrarily—for example, business users may randomly connect with many other users for commercial promotion. Activities of perturbators generally weaken the node classification model leading to poor performance. To improve the performance of the node classification under perturbations, such models should be defended so that the ramification of the perturbators' random activities can be mitigated.

Graph Neural Networks (GNNs) have been widely used on the node classification task [4], [45]–[48] due to superior performance. However, GNNs are also shown to be vulnerable [78], [79] to adversarial attacks. According to [80], two kinds of defending methodologies have been proposed: the adversarial-based method [20]–[22], [81]–[85], and the detection-based method [30]–[32]. The former improves the robustness of GNNs by training with adversarial samples and the latter identifies the attacker nodes (or edges) and alleviates the negative impact by removing them. However, such approaches are not a good fit for defending GNNs from perturbators firstly because a small number of attack patterns are not easy to be discovered by OSN administrators, hence, adversarial samples cannot be used in the training phase. Also, perturbators are not Sybil users [86], i.e., they obey rules and don't conduct malicious attacks, and hence cannot be removed.

In this chapter, we present an alternative approach for defending GNNs from perturbators' actions. Instead of identifying perturbators and directly seeking the remedy of their actions, our proposed approach repairs the prediction of GNNs by assuming that the annotated node labels (which are observed and used for training the GNN) are noisy, which warrants the prediction of the GNN to be adjusted by using a methodological approach. Specifically,

we propose a novel Bayesian label transition model, namely GraphLT (LT stands for Label Transition), that learns a label transition matrix, which enables the substitution of the predicted label of the GNN with an alternative label, if needed. In the beginning, GraphLT trains the GNN-based node classifier with noisy labels on the train graph; the latent labels which would have been assigned to the nodes in the absence of perturbation, are unobserved. For each node, GNN returns a multinomial distribution over the label set. GraphLT then infers the label for each node by sampling from this multinomial distribution. The inference expects that inferred labels would match with the corresponding latent labels. In each iteration, the conditional label transition matrix is dynamically updated by replacing the predicted labels with the inferred labels from Gibbs sampling. With each subsequent iteration, the inferred labels increasingly align with the latent labels, making the inference more accurate. Note that, throughout the process, the latent labels are unknown, so GraphLT employs Bayesian inference to approximate the latent label distribution through conditional label transition. This is made possible by considering that the conditional label transition vector (a multinomial distribution) of each of the K labels follows a Dirichlet prior, and these vectors are iteratively updated using the Bayesian framework. The above design of GraphLT provides two main advantages. First, training a node classifier with noisy labels may cause the performance to decline, which GraphLT can reverse by applying appropriate label transition. Second, GraphLT can repair the prediction of the node classifier when the graph is perturbed with no malicious intent. Most importantly, GraphLT does not require identifying perturbators (or attackers), so it can be applied without any knowledge of the perturbator's activities or an attacker's attack model. In the experiment section, we show results to demonstrate that GraphLT can improve the GNN-based node classifier's performance substantially in either of the scenarios, unperturbed graph, and perturbed graph. Overall, our contribution can be summarized as follows:

- We propose a new Bayesian label transition model, namely GraphLT, to improve the performance of the node classifier on graph data. To the best of our knowledge, our work is the first model that adapts the Bayesian label transition method on GNNs for deperturbation in online social networks.

- GraphLT can enhance the performance of node classification by Bayesian label transition under both unperturbed and perturbed environments. We also show that this enhancement is generalizable over three classic GNNs and across various sizes of attributed graphs.

- Extensive experiments demonstrate that GraphLT is superior to the competing models on seven datasets of different sizes (two KDD Cup 2020 competition datasets and five public graph datasets).

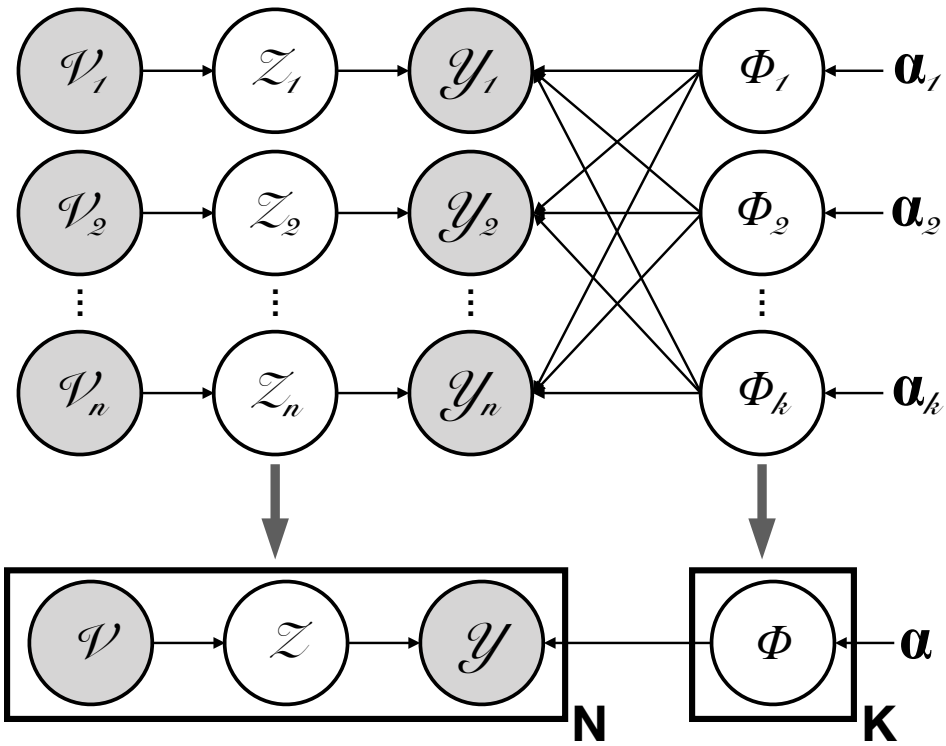


Figure 3.1. The Diagram of Bayesian Label Transition ($\mathcal{V}$, $\mathcal{Z}$, and $\mathcal{Y}$ denote the nodes, latent labels, and noisy labels, respectively. ϕ denotes the conditional label transition matrix. α denotes the Dirichlet parameter. N and K denote the number of nodes and classes, respectively.)

3.2 Methodology

In this section, we first theoretically analyze the Bayesian label transition. Furthermore, we explain the algorithm of our proposed model, GraphLT, and analyze its time complexity.

3.2.1 Bayesian Label Transition

Figure 3.1 presents the diagram of the Bayesian label transition. The unobserved latent labels ($\mathcal{Z}$) depend on the node features, whereas the observed noisy labels ($\mathcal{Y}$) depend on both $\mathcal{Z}$ and the conditional label transition matrix, ϕ, modeled by K multinomial distributions, each with a Dirichlet prior having a parameter α_k. The latent label of node v_n, $\mathbf{z}_n \sim P(\cdot \mid v_n)$, where $P(\cdot \mid v_n)$ is a *Categorical* distribution modeled by the node classifier $f_\theta(v_n)$. The noisy label $\mathbf{y}_n \sim P(\cdot \mid \phi_{\mathbf{z}_n})$, where $\phi_{\mathbf{z}_n}$ is the parameter of *Categorical* distribution $P(\cdot \mid \phi_{\mathbf{z}_n})$. The conditional label transition matrix $\phi = [\phi_1, \phi_2, , \phi_K]^T \in \mathbb{R}^{K \times K}$ consists of K transition vectors. The k-th transition vector $\phi_k \sim \textit{Dirichlet}(\alpha_k)$, where α_k is the parameter of the *Dirichlet* distribution associated with k-th transition vector. We use the symbol α to denote the set $\{\alpha_k\}_{1 \leq k \leq K}$. The goal of Bayesian label transition is to approximate the **inferred label** of a given node to the latent label of that node as identically as possible.

According to Figure 3.1, the dependency of latent labels can be formulated as follows:

$$P(\mathcal{Z} \mid \mathcal{V}, \mathcal{Y}; \alpha) = P(\mathcal{Z} \mid \mathcal{V}, \mathcal{Y}, \phi)\, P(\phi; \alpha)\,, \tag{3.1}$$

where the posterior of $\mathcal{Z}$ is conditioned on the nodes $\mathcal{V}$, the noisy labels $\mathcal{Y}$, and the Dirichlet parameter α.

The posterior of $\mathcal{Z}$ can be deduced by Bayes' theorem as follows:

$$\begin{aligned} P(\mathcal{Z} \mid \mathcal{V}, \mathcal{Y}; \alpha) &= \int_\phi \prod_{k=1}^{K} P(\phi_k; \alpha_k) \cdot \prod_{n=1}^{N} P(\mathbf{z}_n \mid v_n, \mathbf{y}_n, \phi)\, d\phi \\ &= \int_\phi \prod_{k=1}^{K} P(\phi_k; \alpha_k) \cdot \prod_{n=1}^{N} \frac{P(\mathbf{z}_n \mid v_n)\, P(\mathbf{y}_n \mid \mathbf{z}_n, \phi)}{P(\mathbf{y}_n \mid v_n)} d\phi. \end{aligned} \tag{3.2}$$

We assume the Dirichlet distribution is symmetric, i.e., the parameter vector α has the same value for all elements. Thus, Equation (3.2) can be further deduced as follows:

$$\begin{aligned} P(\mathcal{Z} \mid \mathcal{V}, \mathcal{Y}; \alpha) &= \int_\phi \prod_{k=1}^{K} \frac{\Gamma\left(\sum_{k'=1}^{K} \alpha_{k'}\right)}{\prod_{k'=1}^{K} \Gamma(\alpha_{k'})} \prod_{k'=1}^{K} \phi_{kk'}^{\alpha_{k'}-1} \prod_{n=1}^{N} \frac{P(\mathbf{z}_n \mid v_n)}{P(\mathbf{y}_n \mid v_n)} \prod_{n=1}^{N} \phi_{\mathbf{z}_n \mathbf{y}_n} d\phi \\ &= \prod_{n=1}^{N} \frac{P(\mathbf{z}_n \mid v_n)}{P(\mathbf{y}_n \mid v_n)} \int_\phi \prod_{k=1}^{K} \frac{\Gamma\left(\sum_{k'=1}^{K} \alpha_{k'}\right)}{\prod_{k'=1}^{K} \Gamma(\alpha_{k'})} \prod_{k'=1}^{K} \phi_{kk'}^{\alpha_{k'}-1} \prod_{n=1}^{N} \phi_{\mathbf{z}_n \mathbf{y}_n} d\phi, \end{aligned} \tag{3.3}$$

where the term $\prod_{n=1}^{N} \frac{P(\mathbf{z}_n|v_n)}{P(\mathbf{y}_n|v_n)}$ is constant w.r.t. ϕ, and hence we take it out of the integration. We simplify this term as Ω in the following deduction.

According to the conjugation property between the Multinomial distribution and the Dirichlet distribution, Equation (3.3) can be deduced as follows:

$$\begin{aligned} P(\mathcal{Z} \mid \mathcal{V}, \mathcal{Y}; \alpha) &= \Omega \int_{\phi} \prod_{k=1}^{K} \frac{\Gamma\left(\sum_{k'=1}^{K} \alpha_{k'}\right)}{\prod_{k'=1}^{K} \Gamma\left(\alpha_{k'}\right)} \prod_{k'=1}^{K} \phi_{kk'}^{\mathbf{C}_{kk'}+\alpha_{k'}-1} d\phi \\ &= \Omega \prod_{k=1}^{K} \frac{\Gamma\left(\sum_{k'=1}^{K} \alpha_{k'}\right)}{\prod_{k'=1}^{K} \Gamma\left(\alpha_{k'}\right)} \prod_{k=1}^{K} \frac{\prod_{k'=1}^{K} \Gamma\left(\alpha_{k'} + \mathbf{C}_{kk'}\right)}{\Gamma\left(\sum_{k'=1}^{K} \left(\alpha_{k'} + \mathbf{C}_{kk'}\right)\right)}. \end{aligned} \tag{3.4}$$

Here we denote the confusion matrix between the node prediction and the noisy labels as $\mathbf{C}$, where $\mathbf{C}_{kk'}$ is the kk'-th element in the matrix, and $\sum_{k}^{K} \sum_{k'}^{K} \mathbf{C}_{kk'} = N$. The term $\prod_{n=1}^{N} \phi_{\mathbf{z}_n \mathbf{y}_n}$ is expressed as $\prod_{k}^{K} \prod_{k'}^{K} \phi_{kk'}^{\mathbf{C}_{kk'}}$ so we can integrate the terms based on the conjugation property.

Unfortunately, Equation (3.4) can not directly be employed to infer the label. Instead, we apply Gibbs sampling here to approximate our goal. According to Gibbs sampling, for each time we sample $\mathbf{z}_n$ by fixing n-th dimension in order to satisfy the detailed balance condition of a Markov chain. Combined with Equation (3.4) and the recurrence relation of Γ function, $\Gamma(n+1) = n\Gamma(n)$, we sample a sequence of $\mathbf{z}_n$ as follows:

$$\begin{aligned} P(\mathbf{z}_n \mid \mathcal{Z}^{\neg \mathbf{z}_n}, \mathcal{V}, \mathcal{Y}; \alpha) &= \frac{P(\mathcal{Z} \mid \mathcal{V}, \mathcal{Y}; \alpha)}{P(\mathcal{Z}^{\neg \mathbf{z}_n} \mid \mathcal{V}, \mathcal{Y}; \alpha)} \\ &= \frac{P(\mathbf{z}_n \mid v_n)}{P(\mathbf{y}_n \mid v_n)} \frac{\alpha_{\mathbf{y}_n} + \mathbf{C}_{\mathbf{z}_n \mathbf{y}_n}^{\neg \mathbf{z}_n}}{\sum_{k'=1}^{K} \left(\alpha_{k'} + \mathbf{C}_{\mathbf{z}_n k'}^{\neg \mathbf{z}_n}\right)} \\ &\propto \overline{P}(\mathbf{z}_n \mid v_n) \frac{\alpha_{\mathbf{y}_n} + \mathbf{C}_{\mathbf{z}_n \mathbf{y}_n}^{\neg \mathbf{z}_n}}{\sum_{k'=1}^{K} \left(\alpha_{k'} + \mathbf{C}_{\mathbf{z}_n k'}^{\neg \mathbf{z}_n}\right)}, \end{aligned} \tag{3.5}$$

where we denote $\mathcal{Z}^{\neg \mathbf{z}_n}$ as the subset of $\mathcal{Z}$ that removes statistic $\mathbf{z}_n$. In the last row of Equation (3.5), the first term $\overline{P}(\mathbf{z}_n \mid v_n)$ is a categorical distribution of labels for the node v_n modeled by f_θ. We use the term $\overline{P}(\mathcal{Z} \mid \mathcal{V}) \in \mathbb{R}^{N \times K}$ to denote the same over all the nodes, whereas the second term represents the conditional label transition which is obtained from the posterior of the multinomial distribution corresponding to label transition from $\mathbf{y}_n$ to $\mathbf{z}_n$. We use Equation (3.5) to sample the inferred label, $\mathbf{z}_n$. Also, ϕ is updated through Bayesian

inference in each iteration. Such a process is repeated for a given number of epochs with the expectation that subsequent inferred labels can approximate latent labels.

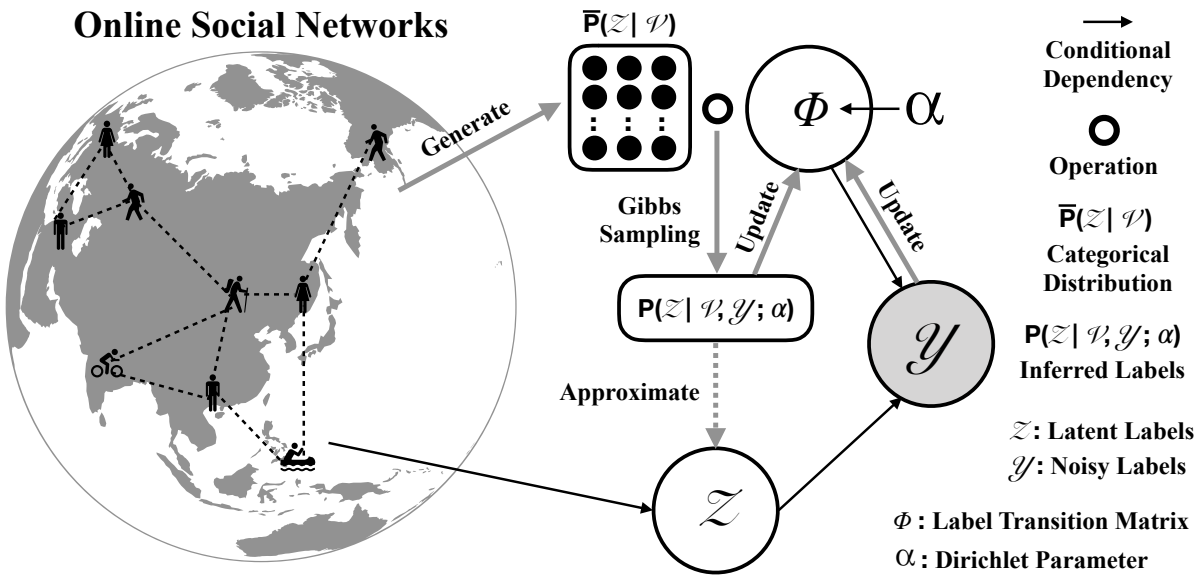


Figure 3.2. The Workflow of Bayesian Label Transition (GraphLT) (The left-hand side represents online social networks (OSN). Each human figure denotes a user of OSN. The dashed line between two users denotes the virtual connection in OSN. Note that the location of each user does not indicate the real location on Earth. The latent labels represent the ground-truth labels of the nodes, which cannot be observed (White color). The noisy labels are manually annotated, which can be observed (Gray color).)

3.2.2 GraphLT Algorithm and Pseudo-code

The total process of GraphLT is displayed in Figure 3.2. Assume that OSN presents an undirected attribute graph. GraphLT classifies the nodes and generates categorical distribution $\overline{P}(\mathcal{Z} \mid \mathcal{V}) \in \mathbb{R}^{N \times K}$ at first. After that, GraphLT applies Gibbs sampling to sample the inferred labels $P(\mathcal{Z} \mid \mathcal{V}, \mathcal{Y}; \alpha) \in \mathbb{R}^{N \times 1}$ and updates the label transition matrix ϕ parameterized by α. The information of $\mathcal{V}$ is represented by both $\mathbf{A}$ and $\mathbf{X}$. The inference will ultimately converge, approximating the inferred labels $P(\mathcal{Z} \mid \mathcal{V}, \mathcal{Y}; \alpha)$ to the latent labels $\mathcal{Z}$ as identically as possible. In brief, the goal of GraphLT is to sample the inferred labels by

supervising the categorical distribution based on dynamic conditional label transition and ultimately to approximate the inferred labels to the latent labels as identically as possible.

Algorithm 1 BAYESIAN Label Transition

Input: Graph $\mathcal{G}_{train}$ and $\mathcal{G}_{test}$, which contain corresponding symmetric adjacency matrix $\mathbf{A}$, feature matrix $\mathbf{X}$, and noisy labels $\mathcal{Y}$, Node classifier f_θ, The number of Warm-up steps WS, The number of inference epochs *Epochs*

1: Train f_θ by Equation (1.2) on $\mathcal{G}_{train}$;
2: Generate categorical distribution $\overline{P}(\mathcal{Z} \mid \mathcal{V})$ by f_θ;
3: Compute warm-up label transition matrix ϕ' on $\mathcal{G}_{train}$;
4: Define the inferred labels $P(\mathcal{Z} \mid \mathcal{V}, \mathcal{Y}; \alpha)$ and dynamic label transition matrix ϕ on $\mathcal{G}_{test}$;
5: **for** *step* := 1 **to** *Epochs* **do**
6: **if** *step* $< WS$ **then**
7: Sample $\mathbf{z}_n$ with warm-up ϕ' by Equation (3.5);
8: **else**
9: Sample $\mathbf{z}_n$ with dynamic ϕ by Equation (3.5);
10: **end if**
11: Update dynamic ϕ and $P(\mathcal{Z} \mid \mathcal{V}, \mathcal{Y}; \alpha)$;
12: **end for**
13: **return** $P(\mathcal{Z} \mid \mathcal{V}, \mathcal{Y}; \alpha)$ and dynamic ϕ.

The pseudo-code of GraphLT is shown in Algorithm (1).

Training: GraphLT trains the node classifier f_θ on the train graph $\mathcal{G}_{train}$ at first (**Line 1**) and then generates categorical distribution $\overline{P}(\mathcal{Z} \mid \mathcal{V})$ by f_θ (**Line 2**).

Inference: Before the inference, GraphLT first computes a warm-up label transition matrix ϕ' by using the prediction over the train graph (**Line 3**) and then defines (creates empty spaces) the inferred labels $P(\mathcal{Z} \mid \mathcal{V}, \mathcal{Y}; \alpha)$ and the dynamic label transition matrix ϕ based on the test graph $\mathcal{G}_{test}$ (**Line 4**). In the warm-up stage of the inference, GraphLT samples $\mathbf{z}_n$ with the warm-up label transition matrix ϕ' (**Line 7**), which is built with the categorical distribution of f_θ and the noisy labels on the train graph. The categorical distributions of both the train graph and the test graph should have high similarity if both follow a similar distribution. Thus, the warm-up ϕ' is a keystone since subsequent inference largely depends on this distribution. After the warm-up stage, GraphLT samples $\mathbf{z}_n$ with the dynamic ϕ (**Line 9**). This dynamic ϕ updates in every epoch with current sampled $\mathbf{z}_n$ and corresponding $\mathbf{y}_n \in \mathcal{Y}$. Simultaneously, the inferred labels $P(\mathcal{Z} \mid \mathcal{V}, \mathcal{Y}; \alpha)$ is also updated based on the before-mentioned $\mathbf{z}_n$ (**Line 11**). The inference will ultimately converge, approximating the

inferred labels to the latent labels as identically as possible. Note that, both the train graph and the test graph contain corresponding symmetric adjacency matrix $\mathbf{A}$, feature matrix $\mathbf{X}$, and noisy labels $\mathcal{Y}$. The categorical distributions of the test graph may change abruptly when this graph is under perturbation since the original distribution in this graph is being perturbed. In this case, GraphLT can also help recover the original categorical distribution by dynamic conditional label transition.

According to Algorithm (1), GraphLT applies Gibbs sampling via Equation (3.5) inside the FOR loop. The time complexity of the sampling is $\mathcal{O}(N_{test} \times K + K^2)$ since element-wise multiplication only traverses the number of elements in matrices once, where N_{test} denotes the number of nodes in the test graph. In practice, the number of test nodes is far more than the number of classes in OSN, i.e., $N_{test} \gg K$. So, the time complexity of this sampling operation is approximately equal to $\mathcal{O}(N_{test})$. Hence, the time complexity of inference except the training (**Line 1**) is $\mathcal{O}(Epochs \times N_{test})$, where $Epochs$ is the number of epochs for inference. This time complexity will be much lower than $\mathcal{O}(N_{test}^2)$ when $Epochs$ is significantly smaller than N_{test} in a realistic scenario. This algorithm could be easily extended to the online inference version. We could reload the node classifier f_θ that is pre-trained offline. To do so, the total time complexity approximates to $\mathcal{O}(Epochs \times N_{test})$.

3.3 Experiments

In this section, we present experimental results to examine GraphLT's performance. We first introduce experimental backgrounds and settings. We then investigate how node classification benefits from GraphLT. Besides, we validate the defending performance of GraphLT against competing defenders and visualize the results. Finally, we analyze model parameters, conduct an ablation study, and discuss limitations and future directions.

3.3.1 Datasets

For testing the performance of GraphLT, we need node-labeled graph datasets. In this chapter, we used seven such datasets, whose statistics are shown in Table 3.1. These datasets are discussed as follows. Both **KDD20(S1)** and **KDD20(S2)** are obtained from KDD Cup

Table 3.1. Statistics of Seven Benchmark Datasets ($|\mathcal{V}|$, $|\mathcal{E}|$, $|F|$, $|C|$, $\#Iter.$ denotes the number of nodes, edges, features, classes, and training epochs, respectively. $Avg.D$ denotes the average degree of test nodes.)

Dataset	$\|\mathcal{V}\|$	$\|\mathcal{E}\|$	$\|F\|$	$\|C\|$	#Iter.	Avg.D
KDD20(S1)	593,486	6,217,004	100	18	1,000	9.63
KDD20(S2)	659,574	5,757,154	100	18	1,000	8.07
Cora	2,708	10,556	1,433	7	200	3.85
Citeseer	3,327	9,228	3,703	6	200	2.78
AMZcobuy	13,752	574,418	767	10	500	37.07
Coauthor	18,333	327,576	6,805	15	500	10.01
Reddit	232,965	114,615,892	602	41	500	491.88

2020 regular machine learning competition track 2, Adversarial Attacks and Defense on Academic Graph (KDD20 Competition) [1]. They were released for stage one (S1) and stage two (S2) of this competition, respectively. Each node in the academic graph represents one research paper and the class label represents the research area. The edge between two nodes represents the citation relationship between two research papers. Both originally have 50,000 unlabeled nodes. We remove unlabeled nodes in our experiment. **Reddit** is a large online discussion forum where users discuss topics in different communities. Hamilton et al. [48] constructed the graph by connecting Reddit posts if the same user comments on both posts. The node label is the community that this post belongs to. For each post, its feature vector includes the embedding of the title and comments, score, and the number of comments. **Coauthor** is co-authorship graphs of computer science based on the Microsoft Academic Graph from the KDD Cup 2016 challenge [2]. Each Node represents one author. Two nodes are connected if they are co-authors of one paper. The feature represents all paper keywords for this author. Node label is the most active research area for this author. **AMZcobuy** comes from the computer segment of the Amazon co-purchase graph [87], where nodes represent the products, whereas edges indicate that two products are frequently bought together. Product reviews are encoded by bag-of-words as the feature. The node label is the

[1] ↑https://www.kdd.org/kdd2020/kdd-cup
[2] ↑https://www.kdd.org/kdd-cup/view/kdd-cup-2016

product category. Both **Cora** and **Citeseer** are well-known citation graph data [88]. Among the above datasets, KDD20(S1), KDD20(S1), and Reddit are large graphs. AMZcobuy and Coauthor are medium graphs. Cora and Citeseer are small graphs.

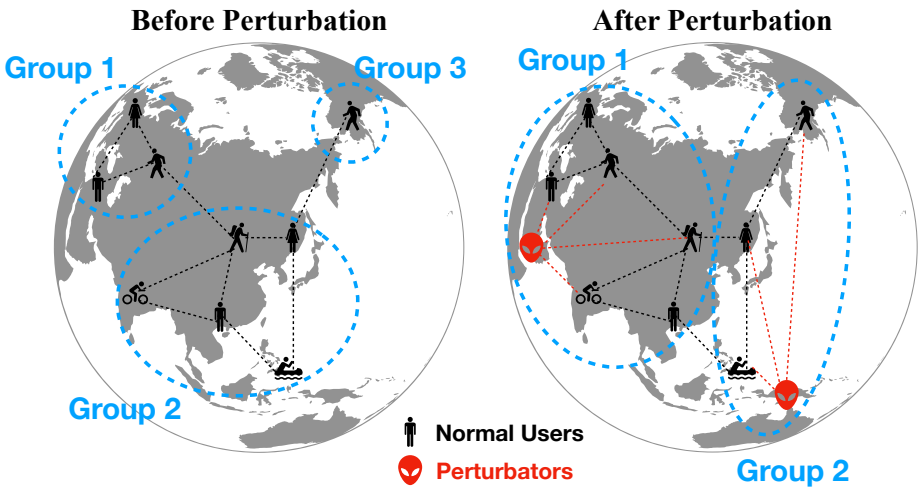


Figure 3.3. An Example of Non-malicious Perturbations (The left-hand side presents the OSN before perturbation (unperturbed environment). The node classifier can classify normal users into the correct group. The right-hand side shows the OSN after perturbation. Perturbators randomly connect with many other users and disturb the accuracy of the node classifier, causing incorrect classification. The red dashed line denotes the connection between the normal user and the perturbator.)

3.3.2 Competing Defending Models

We compare our model with state-of-the-art defending methods that work on graph convolutional networks to overcome non-malicious graph perturbations, which are described in Figure 3.3 as an example. For reproducibility purposes, we maintain the same denotation for each competing method as the corresponding original paper and present the hyper-parameters below. The competing models are trained by Adam optimizer with 200 epochs.

- **RGCN** [89] adopts Gaussian distributions as the hidden representations of nodes to mitigate the negative effects of adversarial attacks. We set up γ as 1, β_1 and β_2 as 5×10^{-4} on all datasets. The number of hidden units is 32. The dropout rate is 0.6. The learning rate is 0.01.

- **GRAND** [90] proposes random propagation and consistency regularization strategies to address the issues of over-smoothing and non-robustness of GCNs. We follow the same procedure to tune the hyper-parameters and present them in Table 3.2.

- **ProGNN** [91] jointly learns the structural graph properties and iteratively reconstructs the clean graph to reduce the effects of adversarial structure. We select α, β, γ, and λ as 5×10^{-4}, 1.5, 1.0, and 1×10^{-3}, respectively. The number of hidden units is 16. The dropout rate is 0.5. The learning rate is 0.01. Weight decay is 5×10^{-4}.

- **GNNGUARD** [25] employs neighbor importance estimation and layer-wise graph memory to quantify the message passing between nodes to defend the GCNs from adversarial attacks. We follow the same setting as [25]. The number of hidden units is 16. The dropout rate is 0.5. The learning rate is 0.01.

Table 3.2. Hyper-parameters of GRAND in This Chapter for Small (Cora, Citeseer), Medium (AMZcobuy, Coauthor), and Large graphs (KDD20(S1), KDD20(S2), and Reddit)

Hyperparameters	**Small**	**Medium**	**Large**
DropNode probability	0.5	0.5	0.5
Propagation step	8	5	5
Data augmentation times	4	4	3
CR loss coefficient	1.0	1.0	0.9
Sharpening temperature	0.5	0.2	0.4
Learning rate	0.01	0.2	0.2
Early stopping patience	200	100	100
Hidden layer size	32	32	32
L2 weight decay rate	5×10^{-4}	5×10^{-4}	5×10^{-4}
Dropout rate in input layer	0.5	0.6	0.6
Dropout rate in hidden layer	0.5	0.8	0.5

3.3.3 Experimental Settings

For all seven graphs, the nodes are partitioned into train, validation, and test, each comprising 40%, 30%, and 30% of the nodes, respectively. We select such partitions to simulate the OSNs as the size of the existing users in OSNs will not be too small compared to the new

coming users. Since GraphLT works with noisy labels, for a subset of nodes, we generate noisy labels by randomly replacing the ground-truth label of a node with another label, chosen uniformly. We denote the percentage of such replacement as noise ratio nr. We examine the generalization of GraphLT over different variants of graph convolutional networks, spectral methods (GCN [4], SGC [47]), and spatial methods (GraphSAGE [48]). GCN applies one-order polynomial on layer-wise graph convolution and achieved state-of-the-art performance with high efficiency. SGC proves that simplified graph convolution does not bring a negative impact on accuracy. GraphSAGE aggregates feature from fixed-size local neighbors of the current node. These three models are well-known graph convolutional networks and they are popular for node classification tasks. To test the defending performance, we simulate non-malicious perturbations in OSN. We assume that perturbators intend to randomly connect with many other users for commercial promotion, which is a common behavior in realistic scenarios. Note that, the perturbations apply to the validation/test graphs only. To ensure that the perturbation is unnoticeable, we limit the number of perturbators to 1% of the validation/test nodes (a.k.a. victim nodes). For each perturbator, we also limit the number of connections up to a given budget, which is 100. This simulation is similar to [78]. However, our simulation does not apply gradient-based attacks, such as FGSM [92], PGD [11], etc. Inspired by [79], we argue that feature values could be proportional to the degree of victim nodes (presented in Table 3.1) for simulating similar extent of perturbations. Note that, the goal of this simulation is not to maliciously attack the node classifier. Instead, we want to examine the defending performance of GraphLT under this non-malicious perturbation setting.

3.3.4 Hyper-parameters of Our Model

Our proposed model, GraphLT, can be applied on top of GNNs. The model architecture and hyper-parameters of GNNs are described in Table 3.4 and Table 3.3, respectively. In the training phase, the number of training epochs for all node classifiers is kept the same for each graph. The hyper-parameter of GraphLT, α, is fixed as 1.0.

Table 3.3. Hyper-parameters of GNNs (#Hidden denotes the number of neurons in each hidden layer of GNNs.)

Hyper-parameters	Values
#Layers	2
#Hidden	200
Optimizer	Adam
Learning Rate	1×10^{-3}

Table 3.4. Model Architecture of GNNs

Model	Aggregator	#Hops	Activation	Dropout
GCN	$\times$	$\times$	ReLU	0.0
SGC	$\times$	2	$\times$	0.0
GraphSAGE	mean	$\times$	ReLU	0.0

3.3.5 Node Classification Benefits from GraphLT

We first investigate how GraphLT reverses the performance's decline caused by training a node classifier with noisy labels. We examine our model with three node classifiers under different noise ratios, where $nr = [0.0, 0.1, 0.2, 0.3]$. Note that we don't apply perturbations in this examination. As shown in Table 3.7, we highlight the best performance for two scenarios, the original performance (Orig.) and the performance after label transition (LT) on each dataset. It's expected that the accuracy increases as the noise ratio decreases. We observe that label transition still works well in some cases even if we use noisy labels with a higher noise ratio. For instance, we train GraphSAGE using the noisy labels with $nr = 0.3$ and get 69.67% original accuracy on Citeseer. This accuracy increases to 76.68% after label transition (under the same noise ratio). This property is very helpful in real life. For example, the profile of new users may lack sufficient information and thus leads to inaccurate annotation. This property indicates that our model could still help improve the performance of the node classifier well even if we only acquire the annotated labels with a higher noise ratio. We also observe that GraphSAGE cannot always outperform GCN. This phenomenon shows more significance on denser graphs. For example, GCN achieves the

best classification performance on AMZcobuy. GCN also surpasses GraphSAGE after label transition ($nr = 0.2$ & 0.1) on Reddit. We argue that this phenomenon may be related to the degree of nodes. Compared to GCN, which applies one-order polynomial on layer-wise graph convolution, GraphSAGE aggregates features from fixed-size local neighbors of the current node. Such aggregation may lose more information when the degree of nodes increases. That is to say, GCN may outperform GraphSAGE when the graph is denser (higher degrees of nodes).

We also examine the generalization of our proposed model over various variants of graph convolutional networks, specifically, GCN [4], SGC [47] and GraphSAGE [48], when the graph is under perturbed. As presented in Table 3.8, we compare the test accuracy of these node classifiers over three scenarios: Before perturbation, After perturbation, and After label transition. For all datasets, the performance of all node classifiers drops after perturbation. However, they improve substantially after we apply GraphLT's label transition. On some occasions, the accuracy after label transition is even better than the accuracy before perturbation on large graphs, whereas this improvement is not so obvious on small graphs. We argue that this phenomenon is caused by the size of the initial warm-up label transition matrix ϕ'. The model can obtain a stronger ϕ' on a large graph so that its improvement gets better. Overall, all three node classifiers benefit from our model, GraphLT. The performance of GraphLT is associated with the noise ratio and the size of the graph.

Table 3.5. The Comparison of Defending Performance between GraphLT (nr=0.1) and Competing Defenders

	KDD20(S1)	**KDD20(S2)**	**Cora**	**Citeseer**	**AMZcobuy**	**Coauthor**	**Reddit**
RGCN [89]	70.97	69.62	26.12	20.09	36.16	23.96	79.30
GRAND [90]	68.51	66.84	27.92	**21.42**	53.67	24.38	86.31
ProGNN [91]	69.34	67.80	25.96	19.74	56.39	24.17	85.25
GNNGUARD [25]	71.73	70.14	24.77	20.12	57.64	23.91	86.83
GraphLT [33]	**76.34**	**74.26**	**27.95**	20.32	**60.64**	**24.64**	**95.87**

3.3.6 Comparison of The Defense

Although all node classifiers benefit from label transition, GraphSAGE shows better performance before perturbation and also after label transition. For each scenario in Table 3.8, we highlight the best performance on each dataset with different colors. For example, GraphSAGE achieves the best performance across all three scenarios on KDD20(S1). So, we employ GraphSAGE as the node classifier of GraphLT in this section and further examine the defending performance between GraphLT and competing methods. In Table 6.5, we highlight the best defending performance (Red color) across seven datasets. The defending results indicate that GraphLT achieves superior defense against competing defenders on most benchmark datasets. We observe that GRAND gains better performance on small graphs whereas works worse on larger graphs. One of the reasons for this is that GRAND assumes that the graph satisfies the homophily property. Larger graphs contain more nodes and a higher number of classes, which may make the network more difficult to satisfy homophily property leading to poor performance by GRAND. We also notice that RGCN gets worse performance on denser graphs. We argue that RGCN adopts a sampling in the hidden representation whereas this sampling may lose more information on denser graphs.

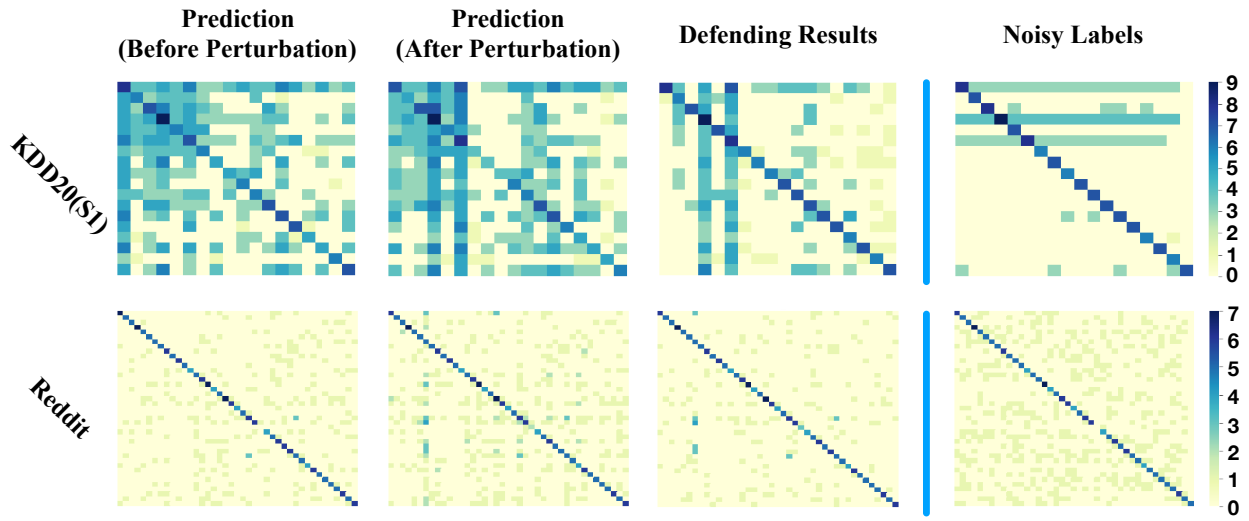


Figure 3.4. The Confusion Matrices (Heatmap) of The Defending Results on KDD20(S1) and Reddit for GraphLT (nr=0.1) (We apply log-scale to the confusion matrix for fine-grained visualization.)

We also visualize the result of three scenarios for GraphLT on two large graphs, KDD20(S1) and Reddit, as examples in Figure 3.4. The first three columns present the result of three scenarios. The last column shows the noisy labels with the noise ratio, $nr = 0.1$. KDD20(S1) suffers from a serious class-imbalanced problem. Most uniform noises are distributed in three classes. The defending result on KDD20(S1) indicates that GraphLT can largely remedy the perturbation by label transition. Similarly, GraphLT shows satisfactory remedies on Reddit as well. We observe that one of the classes almost disappeared even if the graph is not perturbed. GraphLT can partially retrieve the prediction in this class. Overall, the visualization demonstrates that GraphLT can significantly improve the prediction on both datasets.

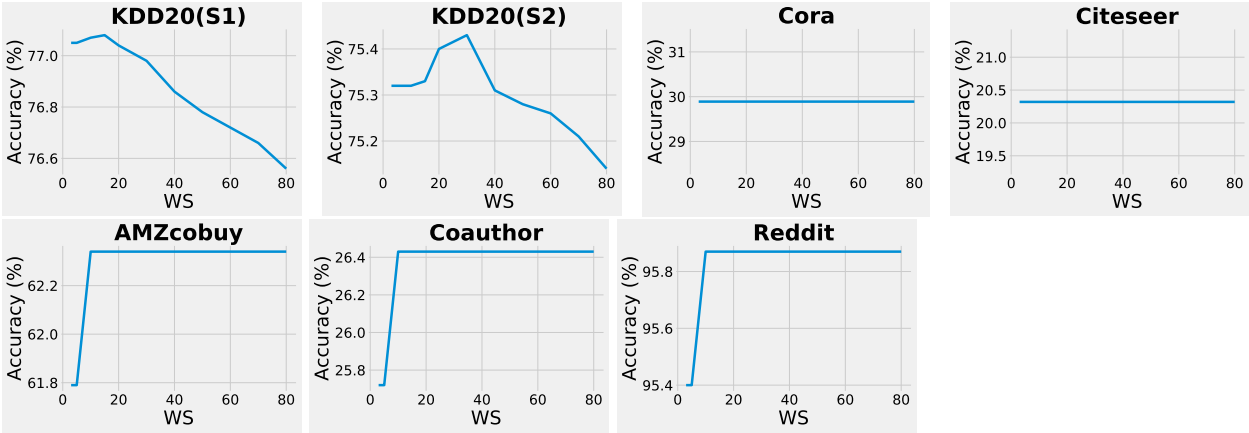


Figure 3.5. Analysis of The Number of Warm-up Steps WS for GraphLT Inference (nr=0.1)

3.3.7 Analysis of Parameters

We analyze how the number of warm-up steps WS affects accuracy. We conduct this analysis on the validation set. We notice that our model can achieve the best performance with only 100 epochs for inference. Thus, we fix the number of epochs for inference as 100 and examine the model with the different number of warm-up steps WS, where $WS \in [3, 80]$. In general, the change of WS has a limited effect on accuracy. To enlarge the difference, we display the curve separately in Figure 3.5. We observe that both larger and smaller WS have a negative effect on accuracy. Larger WS means insufficient inference, whereas smaller WS

implies inadequate epochs to build the dynamic label transition matrix ϕ. This phenomenon shows more obviously on a large graph, such as KDD20(S1) or KDD20(S2). In this chapter, we select the WS for KDD20(S1) and KDD20(S2) as 15 and 30, respectively. For the rest datasets, we fix the WS as 20 instead.

Besides the WS, we analyze the runtime of GraphLT inference. The first row of Table 3.6 presents the average runtime of GraphLT inference with different WS. It's expected that our model has longer runtime on a larger graph. We observe that the standard deviation for each dataset is stable, which indicates that the runtime stays stable no matter how the WS changes. Besides, we also present the runtime per 100 validation nodes in the second row of Table 3.6. The result specifies that this unit runtime does not increase as the size of the graph grows. In other words, the speed of inference won't slow down on a larger graph.

Table 3.6. Analysis of Average ($\mu \pm \sigma$) and Unit (Per 100 nodes) Runtime for GraphLT Inference with Different WS (nr=0.1)

Runtime(s)	KDD20(S1)	KDD20(S2)	Cora	Citeseer	AMZcobuy	Coauthor	Reddit
Average	536.04(6.73)	635.81(8.22)	3.32(0.53)	3.84(0.80)	15.26(0.87)	19.85(0.73)	224.11(9.02)
Unit	0.3289	0.3477	0.4089	0.3848	0.3699	0.3609	0.3206

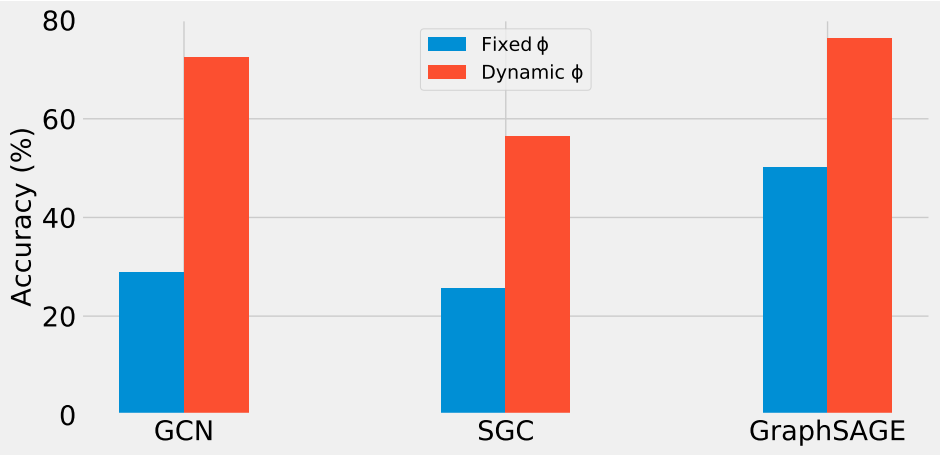


Figure 3.6. Ablation Study of GraphLT over Three Node Classifiers on KDD20(S1) ({Fixed ϕ, Dynamic ϕ} denote whether we dynamically update the transition matrix ϕ in each iteration.)

3.3.8 Ablation Study

To better understand the model architecture, we conduct an ablation study over three node classifiers on KDD20(S1) to illustrate how the label transition matrix ϕ affects the defending performance of GraphLT. We follow the same procedure as our previous defense experiments. According to Figure 3.6, we observe that the accuracy has no change without dynamically updating the transition matrix ϕ. This study reveals that dynamic transition matrix ϕ is a crucial component, which can significantly improve the performance of GraphLT.

3.3.9 Limitation and Future Improvement

GraphLT applies a one-time label transition using noisy labels to recover the test performance of the node classifier against non-malicious random perturbations. Such a scenario assumes that noisy labels are already annotated in the test data. However, this assumption may be impractical for online social networks since new users are continually coming, whereas malicious perturbations could attack these new users at any time before annotations. In the next chapter, we release this assumption. Furthermore, GraphLT can be improved to defend against malicious perturbations by iteratively updating the noisy labels. Last but not least, our work could be easily extended to detect the perturbations on dynamic graphs.

3.4 Chapter Summary

In this chapter, we present a new Bayesian label transition model, namely GraphLT, to improve the performance of OSN classifiers by approximating the inferred labels to the latent labels based on dynamic conditional label transition, which follows the Dirichlet distribution. GraphLT provides two main advantages. For one thing, GraphLT can significantly reverse the performance's decline caused by training a node classifier with noisy labels. For another, GraphLT can repair the prediction of the node classifier under a perturbed environment without identifying perturbators. Extensive experiments demonstrate these two advantages over seven benchmark datasets.

Table 3.7. The Investigation of How GraphLT Reverses The Performances Decline Caused by Training Node Classifiers f_θ with Noisy Labels under Different Noise Ratio nr (**Orig.** denotes the original accuracy. **LT** denotes the accuracy after label transition.)

	Test Acc (%)	**KDD20(S1)**		**KDD20(S2)**		**Cora**		**Citeseer**		**AMZcobuy**		**Coauthor**		**Reddit**	
		Orig.	**LT**	**Orig.**	**LT**	**Orig.**	**LT**	**Orig.**	**LT**	**Orig.**	**LT**	**Orig.**	**LT**	**Orig.**	**LT**
GCN	$nr = 0.3$	35.04	70.48	60.93	76.37	81.30	77.74	66.47	76.78	89.48	75.11	91.96	76.75	92.91	85.15
	$nr = 0.2$	35.19	79.36	61.02	85.24	82.78	88.81	67.97	85.99	89.63	85.70	92.07	87.84	93.53	92.23
	$nr = 0.1$	35.92	88.79	62.82	91.49	83.03	94.22	69.57	93.19	89.58	93.75	92.29	95.51	93.61	96.61
	$nr = 0.0$	36.48	96.52	63.19	98.20	86.23	97.90	69.77	93.89	**89.85**	**98.54**	92.74	98.50	94.34	98.24
SGC	$nr = 0.3$	35.83	64.98	59.58	73.22	83.27	79.46	68.07	76.77	82.31	75.64	90.51	76.73	91.41	85.01
	$nr = 0.2$	36.11	76.34	59.90	85.01	84.38	87.21	70.17	85.98	83.19	82.62	90.87	87.73	92.23	92.25
	$nr = 0.1$	36.21	86.47	60.11	90.15	84.87	95.20	70.47	93.59	84.25	92.85	91.36	95.46	92.44	96.71
	$nr = 0.0$	36.92	96.63	60.30	97.66	85.73	97.78	71.27	96.88	85.44	97.86	91.94	98.42	93.25	97.90
GraphSAGE	$nr = 0.3$	57.09	79.20	62.19	79.68	82.41	78.11	69.67	76.68	85.96	77.12	91.27	79.53	94.01	85.16
	$nr = 0.2$	57.76	82.34	62.67	86.51	84.50	87.95	71.57	87.29	87.20	86.89	91.78	90.38	95.02	91.89
	$nr = 0.1$	57.89	90.13	63.01	91.52	85.98	96.19	72.47	93.69	87.83	94.21	93.02	96.66	95.21	96.23
	$nr = 0.0$	**58.20**	**98.69**	**63.24**	**98.32**	**86.47**	**99.26**	**73.88**	**98.29**	89.79	98.42	**94.07**	**99.29**	**96.14**	**99.38**

Table 3.8. The Examination of The Generalization of GraphLT over Three Classic Node Classifiers (nr=0.1)

	Test Acc (%)	KDD20(S1)	KDD20(S2)	Cora	Citeseer	AMZcobuy	Coauthor	Reddit
GCN	**Before Perturbation**	35.92	62.82	83.03	69.57	**89.58**	92.29	93.61
	After Perturbation	28.89	33.54	27.80	19.02	**85.14**	**35.75**	78.45
	After Label Transition	72.46	66.02	27.81	19.92	**92.51**	48.62	87.31
SGC	**Before Perturbation**	36.21	60.11	84.87	70.47	84.25	91.36	92.44
	After Perturbation	25.56	28.68	27.91	**30.83**	74.04	35.64	39.82
	After Label Transition	56.43	57.93	27.93	**37.84**	90.21	**59.66**	62.37
GraphSAGE	**Before Perturbation**	**57.89**	**63.01**	**85.98**	**72.47**	87.83	**93.02**	**95.21**
	After Perturbation	**50.23**	**39.21**	**27.92**	19.82	50.95	21.60	**87.23**
	After Label Transition	**76.34**	**74.26**	**27.95**	20.32	60.64	24.64	**95.87**

4. BAYESIAN SELF-SUPERVISION AGAINST DYNAMIC GRAPH PERTURBATIONS

4.1 Introduction

As the previous discussion, Graph Neural Networks (GNNs) have been widely ascertained to achieve extraordinary accomplishments on various node classification tasks on large graphs [4], [45]–[48], [50]. Our aforementioned study in the previous chapter reveals that GNN-based node classification can be used for clustering users into different groups based on their interests and behavior in online social networks (OSNs), such as Facebook and Twitter. To do so, unlabeled nodes are annotated so that a supervised node classification model can be built, which can subsequently be used for advertisement, product recommendation, etc. In the previous chapter, we presented a Bayesian label transition model, GraphLT, to repair the test classification accuracy in online social networks under the scenario of noisy labeling and non-malicious random perturbations. This scenario assumes that the test data is manually annotated, which may be impractical for online social networks. To overcome this limitation, in this chapter, we leverage auto-generated labels to supervise the label transition and further perform investigations on the vulnerability of GNNs on label-scarce dynamic graphs. The label-scarce issue arises in a dynamic graph as new nodes appear, which are not annotated; lacking annotated labels undermines the training of robust GNNs [93]. Besides, temporal changes in the graph structure make GNNs vulnerable to adversarial attacks [94] on any nodes at any time. We denote such adversarial attacks as Dynamic Graph Perturbations (DGP). Our objective in this chapter is to safeguard GNNs' performance in the presence of Dynamic Graph Perturbations.

Extensive research has been pursued to improve the robustness of GNNs on node classification tasks [80]. Such efforts can be categorized into four categories: adversarial learning methods [20]–[22], [82], [95], self-supervised learning methods [96]–[98], topological denoising methods [23], [24], and mechanism designing methods [25]–[29]. The adversarial learning methods improve the robustness of GNNs by training with adversarial samples. However, these methods can barely train a robust model under the circumstance of extensive label scarcity. To overcome label scarcity, a kind of self-supervised learning method, namely pre-

training [18], [19], [99]–[101], is introduced, which constructs a pretext task to help GNNs learn transferable graph representation through the pre-training stage. Nevertheless, it is still a challenge to design a generalized pretext task that can be beneficial to arbitrary downstream tasks. Topological denoising methods prune suspicious edges on the graph before feeding this graph into GNNs. Nonetheless, such methods are not efficient on dynamic graphs as the graph structure is always changing over time. Mechanism designing methods mainly propose a message-passing mechanism, a.k.a. graph convolutional operator or node aggregator, to better classify the nodes. Such a mechanism sometimes highly depends on heuristics explorations.

Self-training [102] is an extended form of pre-training. It assigns self-generated pseudo-labels to highly confident unlabeled nodes and then adds these nodes to the set of labeled nodes for the next training iteration. Nonetheless, accurate assignment of such pseudo-labels may suffer serious degradation when the graph is undergoing dynamic graph perturbations. Learning graph representation with noisy labels is a potential solution to mitigate this issue [33], [103]. In this chapter, we assume that noisy labels include both manual-annotated labels and auto-generated labels. We argue that such noisy labels assigned to the vertices could be regarded as a kind of self-information for each node, which can help GNN to retain its performance in the presence of DGP. In fact, the usage of noisy labels is a kind of defense, which is equivalent to recovering the original label distribution of a node on a perturbed graph, for which the node's label distribution has undergone a change due to perturbation. This intuition inspires us to propose a Bayesian self-supervision model, namely GraphSS[1], which recovers the original label distribution iteratively, to improve the robustness of a GNN-based node classifier against dynamic graph perturbations. First, GraphSS utilizes self-information, i.e., noisy labels of the nodes, to build a robust node classifier (without knowing the ground-truth latent labels), helping the node classification in a label-scarce graph. Most importantly, GraphSS can recover the prediction of a node classifier by iterative label transition with auto-generated labels when the graph is undergoing dynamic graph perturbations.

We summarize the contributions of this chapter as follows:

[1]↑Source code: **https://github.com/junzhuang-code/GraphSS**

- We generalize noisy supervision as a subset of self-supervised learning methods and propose a new Bayesian self-supervision model, namely GraphSS, to improve the robustness of the node classifier on the dynamic graph. To the best of our knowledge, our work is the first model that adapts Bayesian inference with self-supervision and significantly improves the robustness of GNNs against perturbations on label-scarce dynamic graphs.

- Extensive experiments demonstrate that GraphSS can 1) effectively recover the prediction of a node classifier against dynamic graph perturbations, and 2) affirmatively alert such perturbations on dynamic graphs. These two advantages prove to be generalized over three classic GNNs across five public graph datasets.

4.2 Methodology

In the previous chapter, we theoretically analyze the Bayesian label transition. In this chapter, we derive the Bayesian self-supervision from the Bayesian label transition by using auto-generated labels. In this section, we mainly explain our algorithm and analyze its time complexity.

The total process of Bayesian Self-supervision (GraphSS) is displayed in Figure 4.1. Given an undirected attribute graph, GraphSS classifies the nodes with manual-annotated labels $\mathcal{Y}_m$ at first and then generates both categorical distribution $\overline{P}(\mathcal{Z} \mid \mathcal{V})$ and auto-generated labels $\mathcal{Y}_a$. After that, GraphSS applies Gibbs sampling to sample the inferred labels $P(\mathcal{Z} \mid \mathcal{V}, \mathcal{Y}_a; \alpha) \in \mathbb{R}^{N \times 1}$ and updates the label transition matrix ϕ parameterized by α. The information of $\mathcal{V}$ is represented by both $\mathbf{A}$ and $\mathbf{X}$. For alert, GraphSS applies a one-time label transition with $\mathcal{Y}_a$. For recovery, GraphSS iteratively re-trains the node classifier to update $\overline{P}(\mathcal{Z} \mid \mathcal{V})$. The inference will ultimately converge, approximating $P(\mathcal{Z} \mid \mathcal{V}, \mathcal{Y}_a; \alpha)$ to $\mathcal{Z}$ as close as possible. In brief, the goal of GraphSS is to sample the inferred labels by supervising the categorical distribution based on dynamic conditional label transition and ultimately to approximate the inferred labels to the latent labels as closely as possible.

The pseudo-code of GraphSS is shown in Algorithm (2).

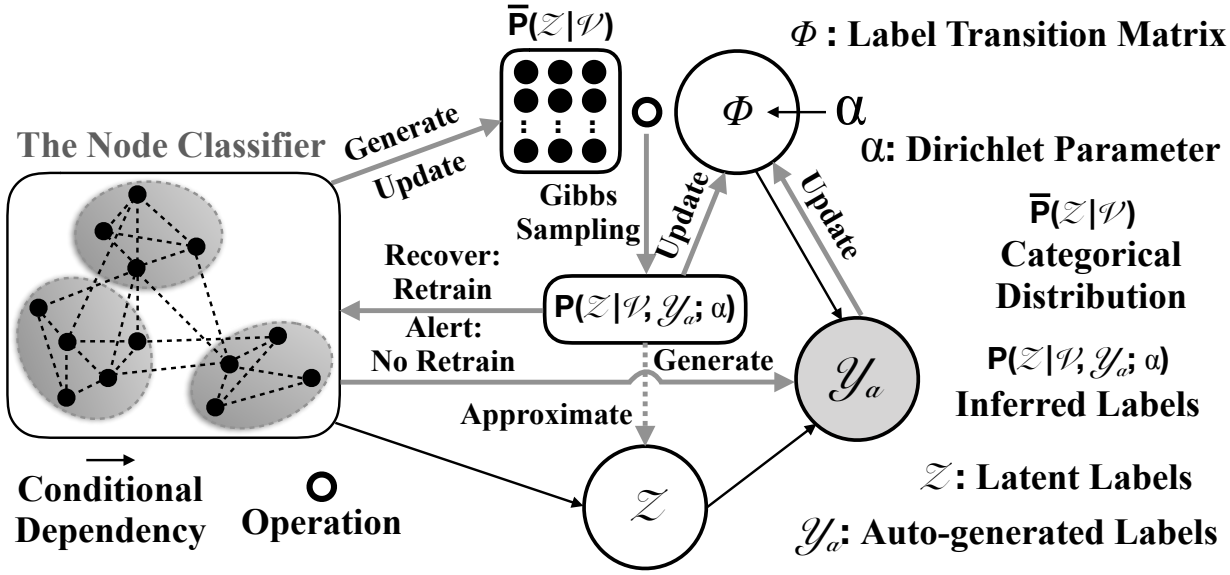


Figure 4.1. The Workflow of Bayesian Self-Supervision Model, GraphSS (The latent labels (White color) represent the unobserved ground-truth labels whereas the auto-generated labels (Gray color) are observed noisy labels.)

Training: GraphSS trains the node classifier f_θ on the train graph $\mathcal{G}_{train}$ at first (**line 1**) with manual-annotated labels $\mathcal{Y}_m$ and then generates categorical distribution $\overline{P}(\mathcal{Z} \mid \mathcal{V})$ and auto-generated labels $\mathcal{Y}_a$ by f_θ (**line 2**).

Inference: Before the inference, GraphSS first computes a warm-up label transition matrix ϕ' by using the prediction over $\mathcal{G}_{train}$ (**line 3**) and then defines (creates empty spaces) the inferred labels $P(\mathcal{Z} \mid \mathcal{V}, \mathcal{Y}_a; \alpha)$ and the dynamic label transition matrix ϕ based on the test graph $\mathcal{G}_{test}$ (**line 4**). In the warm-up stage of the inference, GraphSS samples $\mathbf{z}_n$ with the warm-up ϕ' (**line 7**), which is built with the categorical distribution of f_θ and $\mathcal{Y}_m$ on $\mathcal{G}_{train}$. The categorical distributions of both $\mathcal{G}_{train}$ and $\mathcal{G}_{test}$ should have high similarity if both follow a similar distribution. Thus, the warm-up ϕ' is a keystone since subsequent inference largely depends on this distribution. After the warm-up stage, GraphSS samples $\mathbf{z}_n$ with the dynamic ϕ (**line 9**). This dynamic ϕ updates in every epoch with current sampled $\mathbf{z}_n$ and corresponding $\mathcal{Y}_a$. Simultaneously, $P(\mathcal{Z} \mid \mathcal{V}, \mathcal{Y}_a; \alpha)$ is also updated based on the before-mentioned $\mathbf{z}_n$ (**line 11**). For recovery, GraphSS iteratively re-trains the node classifier to

Algorithm 2 BAYESIAN Self-supervision

Input: Graph $\mathcal{G}_{train}$ and $\mathcal{G}_{test}$, which contain corresponding symmetric adjacency matrix $\mathbf{A}$ and feature matrix $\mathbf{X}$, Manual-annotated labels $\mathcal{Y}_m$ in $\mathcal{G}_{train}$, Node classifier f_θ, The number of warm-up steps WS, The number of epochs for inference $Epochs$

1: Train f_θ by Equation (1.2) with $\mathcal{Y}_m$ on $\mathcal{G}_{train}$;
2: Generate categorical distribution $\overline{P}(\mathcal{Z} \mid \mathcal{V})$ and auto-generated labels $\mathcal{Y}_a$ by f_θ;
3: Compute the warm-up label transition matrix ϕ' based on $\mathcal{G}_{train}$;
4: Define the inferred labels $P(\mathcal{Z} \mid \mathcal{V}, \mathcal{Y}_a; \alpha)$ and the dynamic label transition matrix ϕ based on $\mathcal{G}_{test}$;
5: **for** $step \leftarrow 1$ to $Epochs$ **do**
6: **if** $step \leq WS$ **then**
7: Sample $\mathbf{z}_n$ with warm-up ϕ' by Equation (3.5);
8: **else**
9: Sample $\mathbf{z}_n$ with dynamic ϕ by Equation (3.5);
10: **end if**
11: Update dynamic ϕ, $P(\mathcal{Z} \mid \mathcal{V}, \mathcal{Y}_a; \alpha)$;
12: **if** not Alert **then**
13: Retrain f_θ;
14: **end if**
15: **end for**
16: **return** $P(\mathcal{Z} \mid \mathcal{V}, \mathcal{Y}_a; \alpha)$ and Dynamic ϕ.

update its parameters (**line 13**). The inference will ultimately converge, approximating the inferred labels to the latent labels as closely as possible. Note that both $\mathcal{G}_{train}$ and $\mathcal{G}_{test}$ contain corresponding symmetric adjacency matrix $\mathbf{A}$ and feature matrix $\mathbf{X}$. The categorical distributions of $\mathcal{G}_{test}$ may change abruptly when this graph is under perturbation since the original distribution in this graph is being perturbed. In this case, GraphSS can also help recover the original categorical distribution by dynamic conditional label transition.

According to Algorithm (2), GraphSS applies Gibbs sampling via Equation (3.5) inside the FOR loop. The time complexity of the sampling is $\mathcal{O}(N_{test} \times K + K^2)$ since element-wise multiplication only traverses the number of elements in matrices once, where N_{test} denotes the number of nodes in the test graph. In practice, the number of test nodes is far more than the number of classes, i.e., $N_{test} \gg K$. So, the time complexity of this sampling operation is approximately equal to $\mathcal{O}(N_{test})$. Hence, the **time complexity** of inference except the first training (**line 1**) is $\mathcal{O}(Epochs \times N_{test})$, where $Epochs$ is the number of epochs for inference.

4.3 Experiments

In this section, we present and analyze experimental results as follows. We first examine how far Bayesian self-supervision (GraphSS) can defend node classifiers against dynamic graph perturbations. We then assess whether GraphSS can alert such perturbations. Furthermore, we analyze the runtime and parameters and conduct an ablation study. At last, we discuss the limitation and future directions to illustrate how the community benefits from our work.

Table 4.1. Statistics of Datasets ($|\mathcal{V}|$, $|\mathcal{E}|$, $|F|$, and $|C|$ denote the number of nodes, edges, features, and classes, respectively. Avg.D denotes the average degree of test nodes.)

Dataset	$\|\mathcal{V}\|$	$\|\mathcal{E}\|$	$\|F\|$	$\|C\|$	Avg.D
Cora	2,708	10,556	1,433	7	3.85
Citeseer	3,327	9,228	3,703	6	2.78
Pubmed	19,717	88,651	500	3	4.49
AMZcobuy	7,650	287,326	745	8	32.32
Coauthor	18,333	327,576	6,805	15	10.01

4.3.1 Experimental Settings

Our proposed model is evaluated on five public datasets in Table 4.1. **Cora**, **Citeseer**, and **Pubmed** are famous citation graph data [88]. **AMZcobuy** comes from the photo segment of the Amazon co-purchase graph [87]. **Coauthor** is co-authorship graphs of computer science based on the Microsoft Academic Graph from the KDD Cup 2016 challenge [2]. For all five datasets, the percentage of the train, validation, and test partition comprise 40%, 20%, and 40% of the nodes, respectively. We select such partitions to simulate the OSNs as the size of the existing graph in OSNs will not be too small compared to the new coming dynamic subgraphs. In this chapter, we assume that the train graphs contain manual-annotated labels but the validation and test graphs don't. In other words, the node classifiers are trained with manual-annotated labels whereas GraphSS applies inference with auto-generated labels. To

[2]↑https://www.kdd.org/kdd-cup/view/kdd-cup-2016

simulate the manual-annotated labels, we randomly replace the ground-truth label of a node with another label, chosen uniformly. We denote the percentage of such replacement as noise ratio nr and fix $nr = 10\%$ in the experiments. We set the number of training epochs as 200 because all node classifiers converge within 200 epochs in the training phase. The model architecture and hyper-parameters of GNNs in this chapter are the same as that in Chapter 3 except that GraphSAGE uses "gcn" aggregator. The hyper-parameter of GraphSS, α, is fixed as 1.0.

4.3.2 Implementation of Dynamic Graph Perturbations

To simulate the dynamic graph perturbations, we execute node-level direct evasion non-targeted attacks [79] on the links and features of the nodes ($L\&F$) in dynamic graphs, whereas the trained node classifier remains unchanged. Similar to [104], we denote the intensity of perturbation as n_{pert}. We set n_{pert} for Cora, Citeseer, and Pubmed as 2, and for AMZcobuy, Coauthor as 5. The ratio of n_{pert} between applying on links and applying on features is $1 : 10$. For example, the attacker applies 2 perturbations on links and 20 perturbations on features for $L\&F$. To construct a group of dynamic subgraphs, we select the random seed from 1 to n_{graphs}, where n_{graphs} denotes the number of dynamic subgraphs.

4.3.3 Competing Defending Methods

We present the hyper-parameters of our competing methods below. For reproducibility, we maintain the same denotation for each competing method as the corresponding original paper. The competing models are trained by Adam optimizer with 200 epochs.

- AdvTrain [104], [105] assigns pseudo labels to generated adversarial samples and then retrains the node classifier with both noisy labeled nodes and adversarial nodes. In detail, we first generate the adversarial samples by $L\&F$. The number of adversarial samples is equal to 10% of victim nodes. After that, we add up these generated adversarial samples into the training set and then retrain the node classifier. Finally,

we evaluate this defending result by implementing $L\&F$ on the victim nodes. The approach of adversarial training can be formulated as follows:

$$\begin{aligned} \mathcal{L}_{noisy} &= \mathcal{L}(\mathbf{Y}_{noisy},\ f_\theta(\tilde{\mathbf{A}}, \mathbf{X})), \\ \mathcal{L}_{adv} &= \mathcal{L}(\mathbf{Y}_{pseudo},\ f_\theta(\mathbf{A}', \mathbf{X}')), \\ \theta^* &= \arg\min_\theta \left(\mathcal{L}_{noisy} + \mathcal{L}_{adv}\right), \end{aligned} \tag{4.1}$$

where $\mathbf{Y}_{noisy}$, $\mathbf{Y}_{pseudo}$ are the corresponding labels to compute loss functions, and $\mathbf{A}'$, $\mathbf{X}'$ are perturbed adjacency, feature matrices generated by the attacking algorithm.

- GNN-Jaccard [31] preprocesses the graph by eliminating suspicious connections, whose Jaccard similarity of nodes features is smaller than a given threshold. The similarity threshold for dropping edges is 0.01. The number of hidden units is 16. The dropout rate is 0.5.

- GNN-SVD [84] proposes another preprocessing approach with low-rank approximation on the perturbed graph to mitigate the negative effects from high-rank attacks, such as Nettack [79]. The number of singular values and vectors is 15. The number of hidden units is 16. The dropout rate is 0.5.

- RGCN [89] adopts Gaussian distributions as the hidden representations of nodes to mitigate the negative effects of adversarial attacks. We set up γ as 1, β_1 and β_2 as 5×10^{-4} on all datasets. The number of hidden units is 32. The dropout rate is 0.6. The learning rate is 0.01.

- GRAND [90] proposes random propagation and consistency regularization strategies to address the issues of over-smoothing and non-robustness of GCNs. We follow the same procedure to tune the hyper-parameters. The optimal hyper-parameters of GRAND in this chapter are reported in Table 4.2.

- ProGNN [91] jointly learns the structural graph properties and iteratively reconstructs the clean graph to reduce the effects of adversarial structure. We select α, β, γ, and

λ as 5×10^{-4}, 1.5, 1.0, and 1×10^{-3}, respectively. The number of hidden units is 16. The dropout rate is 0.5. The learning rate is 0.01. Weight decay is 5×10^{-4}.

- NRGNN [17] develops a label noise-resistant framework that brings clean label information to unlabeled nodes based on the feature similarity. We follow the corresponding settings except that we choose uniform noise with ptb_rate=0.1.

Table 4.2. Hyper-parameters of GRAND in This Chapter

Hyper-parameters	**Cora**	**Citeseer**	**Pubmed**	**AMZcobuy**	**Coauthor**
DropNode probability	0.5	0.5	0.5	0.5	0.5
Propagation step	8	2	5	5	5
Data augmentation times	4	2	4	3	3
CR loss coefficient	1.0	0.7	1.0	0.9	0.9
Sharpening temperature	0.5	0.3	0.2	0.4	0.4
Learning rate	0.01	0.01	0.2	0.2	0.2
Early stopping patience	200	200	100	100	100
Hidden layer size	32	32	32	32	32
L2 weight decay rate	5e-4	5e-4	5e-4	5e-4	5e-4
Dropout rate in input layer	0.5	0.0	0.6	0.6	0.6
Dropout rate in hidden layer	0.5	0.2	0.8	0.5	0.5

4.3.4 GraphSS Defends Node Classifiers

We examine how far GraphSS can defend node classifiers against dynamic graph perturbations. We randomly sample 20% of the nodes from the test graph to build a subgraph and repeat this process five times to construct a group of dynamic subgraphs. For each subgraph, we apply non-targeted adversarial attacks on both links and features together ($L\&F$) [79]. To assess the generalization, we apply the adversarial attacks on two groups of popular node classifiers, spectral models (GCN [4], SGC [47]) and spatial models (GraphSAGE [48]), across five public datasets. We also compare the defending performance of GraphSS with the competing methods and present the accuracy of each group of dynamic subgraphs in Table 4.4. The best defending performance on each dataset is highlighted. Originally, all three node classifiers have comparable performance. After the adversarial attacks, the results

explicitly state that GraphSS gains superior defense against the state-of-the-art competing methods and even exceeds the original performance in some cases. The extent of recovery may sometimes be influenced by the attacking consequence, i.e., GraphSS can retrieve higher accuracy when the node classifier undergoes less degradation.

In this examination, we have several observations as follows. 1) AdvTrain gets unfavorable performance across all datasets, which reveals that simply using adversarial samples to train the node classifiers has limited contributions to the defense. 2) RGCN performs poorly in comparison to AdvTrain along with the density (Avg.D) of the graph increasing because RGCN adopts a sampling in the hidden representation, whereas this sampling may lose more information on denser graphs. 3) GNN-SVD obtains lower accuracy than GNN-Jaccard across all graphs as GNN-SVD is tailored to fit the targeted attacks and may not adapt well to the non-targeted attacks. 4) GRAND deteriorates seriously on Cora. One of the reasons for this deterioration is that GRAND assumes the graph satisfies the homophily property. The assumption may break when the graph suffers serious perturbations. Besides, GRAND is sensitive to the parameters setting as we spent substantial time tuning the optimal parameters on different datasets. On the contrary, GraphSS does not assume such network property, and it is also easy to tune. 5) ProGNN is superior to GNN-Jaccard on denser graphs. We argue that ProGNN can simultaneously update the graph structure and the parameters of the node classifiers with low rank and feature smoothness constraints. Such an update reaps huge fruits for recovering better graph structures on denser graphs.

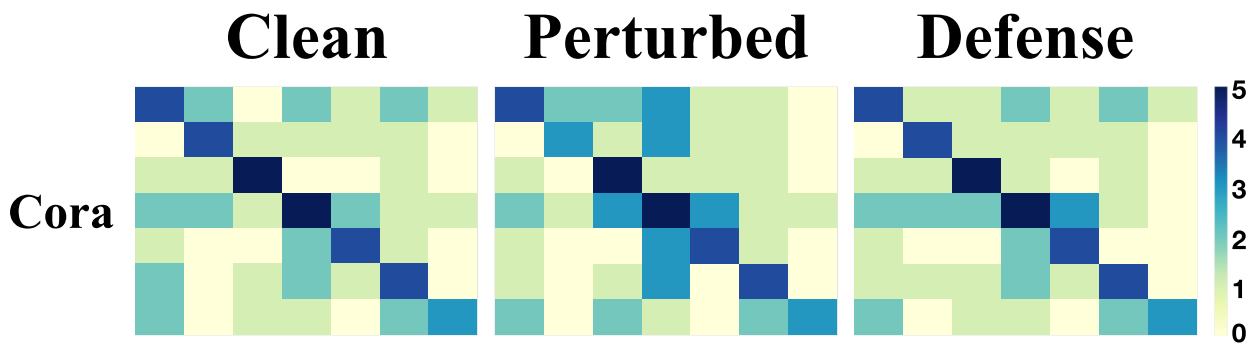


Figure 4.2. The Confusion Matrices (Heatmap) of The Defending Result on Cora by GraphSS (We apply log-scale to the confusion matrix for fine-grained visualization.)

Furthermore, we visualize our defending result (on top of GCN) on the test graph of Cora as an example in Figure 4.2, which presents the result of prediction before/after graph perturbations, and our defending result. The visualization clarifies that GraphSS can recover the prediction after perturbations as close as that in a clean environment.

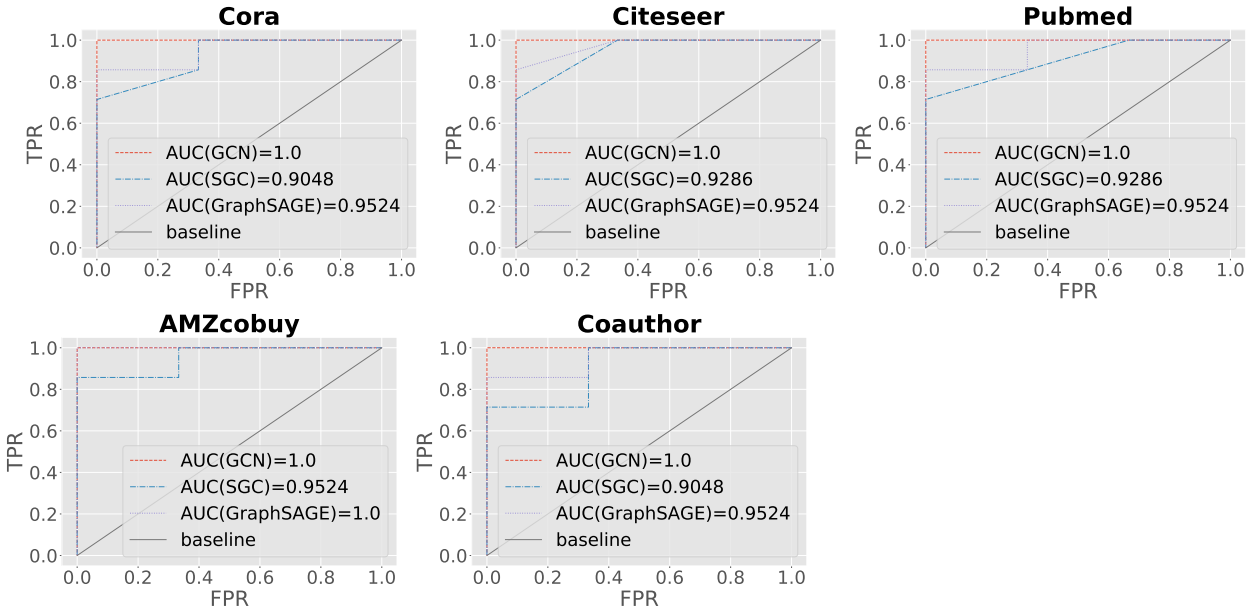


Figure 4.3. The Assessment of Alerting Dynamic Graph Perturbations by GraphSS via ROC Curve

4.3.5 GraphSS Can Alert Dynamic Graph Perturbations

We assess whether GraphSS can alert the perturbations on dynamic graphs. We repeat the aforementioned procedure ten times to construct a group of dynamic subgraphs and then randomly select three subgraphs from the group to be perturbed by non-targeted adversarial attacks ($L\&F$) [79]. We assess our model on three node classifiers across five public datasets and present the performance via ROC curve and AUC score in Figure 4.3. A higher true positive rate (TPR) indicates that GraphSS has a higher probability of successfully alerting the true perturbations. On the contrary, a higher false positive rate (FPR) implies that GraphSS has more chance to make a false alert on unperturbed subgraphs. Figure 4.3 reveals that GraphSS achieves robust alerts on top of GCN across all datasets. For the

other two node classifiers, GraphSS has higher TPR and AUC scores with GraphSAGE than with SGC. This indicates that GraphSS accomplishes better alert performance on top of GraphSAGE when the subgraph is undergoing perturb.

4.3.6 Analysis of Runtime

We analyze the runtime of GraphSS between defense and alert. For each dataset in Table 4.5, the left column presents the average runtime whereas the right column presents the unit runtime. It's expected that GraphSS has longer runtime on a larger graph. However, the unit runtime maintains a comparable level as the size of the graph increases. This result indicates that the size of the graph doesn't affect the speed of inference. We observe that the average runtime of defense is slightly higher than that of alert. We argue that such a gap is reasonable as GraphSS iteratively retrains the node classifiers in the defense scenario. Overall, this analysis illustrates that GraphSS retains a stable speed of inference regardless of the size of the graph.

Table 4.3. Runtime (s) Comparison among Five Methods on Cora

GNN-Jaccard	GNN-SVD	GraphSS	RGCN	GRAND
9.39 ($\pm$0.42)	10.72 ($\pm$0.55)	22.75 ($\pm$0.65)	27.65 ($\pm$1.36)	53.42 ($\pm$2.68)

We also sort the defense runtime of five methods on Cora as an example in Table 4.3. The sorting indicates that topological denoising methods mainly apply in the preprocessing stage and may have a faster speed on smaller graphs.

4.3.7 Analysis of Parameters

We investigate how the number of warm-up steps WS and the number of retraining epochs $Retrain$ affects the defending performance. We select GCN as the node classifier and conduct this analysis on the validation set. We fix the number of inference epochs as 100 and then apply grid search to look for the optimal parameters, where $WS \in [5, 80]$ and $Retrain \in [20, 100]$. The results turn out that WS, and $Retrain$ reach the optimum

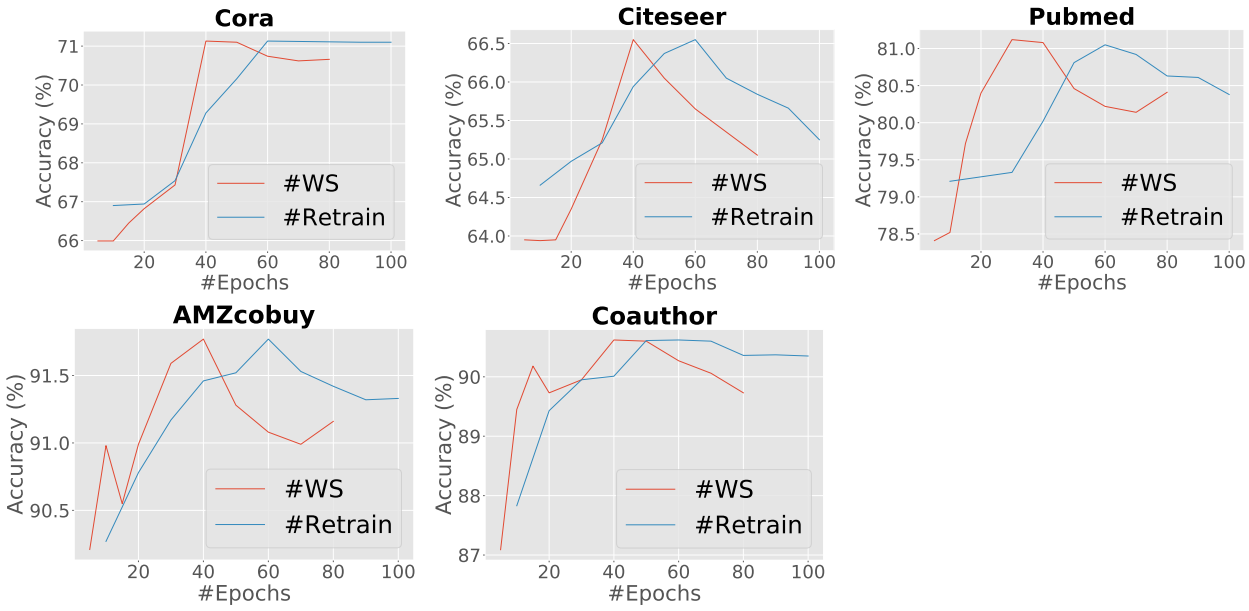


Figure 4.4. Analysis of The Number of Warm-up Steps WS (Blue) and The Number of Retraining Epochs *Retrain* (Red) for GraphSS

nearby 40, and 60, respectively. To display the trend clearly, we fix one parameter as its optimal value and present another one in a curve. For example, we fix *Retrain* as 60 and investigate how the validation accuracy changes with different WS in the blue curve. To enlarge the difference, we display these curves separately in Figure 4.4. We observe that both larger and smaller WS have a negative effect on accuracy. Larger WS means insufficient inference, whereas smaller WS implies inadequate epochs to build the dynamic label transition matrix ϕ. This property also applies to *Retrain*. A larger *Retrain* is even harmful to the performance. In this chapter, we select the aforementioned optimal values for our model.

4.3.8 Ablation Study

To better understand the model architecture, we conduct an ablation study on top of GCN to illustrate how the label transition matrix ϕ and retraining affect the defending performance of GraphSS under four scenarios. We follow the same procedure as our previous

Figure 4.5. Ablation Study of GraphSS ({Retrain, No Retrain} denote whether we retrain the node classifier in each iteration. {Fixed ϕ, Dynamic ϕ} denote whether we dynamically update the transition matrix ϕ in each iteration.)

defense experiments. According to Figure 4.5, we observe that the accuracy has no change without dynamically updating the transition matrix ϕ. In the last two scenarios (Dynamic ϕ), we notice that retraining the node classifier can help increase the accuracy further. This study reveals that the dynamic transition matrix ϕ is a crucial component of GraphSS. Retraining can further promote the robustness of the node classifier with the help of dynamic ϕ.

4.3.9 Limitation and Future Improvement

GraphSS employs Gibbs sampling to sample the node labels in each iteration followed by GNN re-training. During this process, however, Gibbs sampling may suffer slow convergence issues as the posterior distribution adapts incrementally over a sequence of label transitions. In the next chapter, we proposed a new sampling method to speed up the convergence of the Bayesian label transition.

4.4 Chapter Summary

In this chapter, we generalize noisy supervision as a subset of self-supervised learning methods. This generalization regards the noisy labels, including both manual-annotated labels and auto-generated labels, as one kind of self-information for each node. The robustness of the node classifier can be improved by utilizing such self-information. We then present a new Bayesian self-supervision model, namely GraphSS, to accomplish this improvement by supervising the categorical distribution of the latent labels based on dynamic conditional label transition, which follows the Dirichlet distribution. Extensive experiments demonstrate that GraphSS can not only affirmatively alert the perturbations on dynamic graphs but also effectively recover the prediction of a node classifier when the graph is under such perturbations. These two advantages prove to be generalized over three classic GNNs across five public graph datasets.

Table 4.4. The Comparison of Defending Results (Test accuracy) between GraphSS and The Competing Methods (Original/Attack denote the test accuracy before/after the non-target attacks ($L\&F$).)

	Test Acc. (%)	Cora	Citeseer	Pubmed	AMZcobuy	Coauthor
GCN	Original	81.46 (±0.89)	69.95 (±0.53)	81.83 (±0.62)	93.14 (±0.96)	92.01 (±0.85)
	Attack [79]	17.01 (±5.11)	35.42 (±1.67)	41.49 (±1.99)	46.71 (±1.95)	42.85 (±1.11)
	AdvTrain [104]	25.64 (±2.16)	37.66 (±1.80)	44.06 (±0.79)	52.50 (±1.40)	62.18 (±0.55)
	GNN-Jaccard [31]	62.31 (±2.14)	68.79 (±1.66)	69.73 (±1.24)	82.76 (±2.38)	79.42 (±0.57)
	GNN-SVD [84]	54.65 (±1.82)	63.92 (±2.91)	63.75 (±1.10)	78.99 (±1.48)	73.64 (±0.99)
	RGCN [89]	31.72 (±2.03)	46.93 (±1.60)	48.19 (±2.32)	50.90 (±1.84)	57.51 (±1.22)
	GRAND [90]	34.78 (±3.12)	63.68 (±1.75)	52.67 (±1.66)	57.56 (±1.65)	68.11 (±3.35)
	ProGNN [91]	58.72 (±2.22)	67.74 (±3.90)	69.02 (±1.95)	83.71 (±0.67)	81.99 (±2.57)
	NRGNN [17]	57.15 (±1.93)	62.53 (±1.44)	64.20 (±2.35)	69.18 (±2.01)	70.05 (±2.41)
	GraphSS [34]	**65.38 (±2.59)**	**69.18 (±0.99)**	**81.16 (±1.41)**	**90.88 (±1.61)**	**89.51 (±1.09)**
SGC	Original	83.21 (±1.52)	69.65 (±0.86)	83.43 (±1.12)	90.56 (±1.25)	91.04 (±1.31)
	Attack [79]	36.55 (±8.16)	44.79 (±2.75)	42.60 (±1.95)	45.34 (±2.02)	41.59 (±1.14)
	AdvTrain [104]	46.28 (±3.84)	47.43 (±2.58)	53.65 (±1.69)	50.42 (±1.30)	62.52 (±1.82)
	GNN-Jaccard [31]	84.35 (±3.74)	69.77 (±1.59)	72.48 (±1.28)	79.72 (±2.22)	79.61 (±1.16)
	GNN-SVD [84]	65.42 (±4.20)	54.39 (±1.64)	68.40 (±1.70)	75.65 (±2.16)	72.54 (±1.24)
	RGCN [89]	41.32 (±0.67)	45.17 (±3.07)	46.56 (±3.22)	48.42 (±1.68)	56.33 (±1.29)
	GRAND [90]	32.64 (±3.57)	67.18 (±3.44)	67.34 (±1.09)	56.87 (±1.75)	68.39 (±1.97)
	ProGNN [91]	75.92 (±4.62)	68.55 (±1.68)	69.84 (±1.74)	82.26 (±0.83)	81.57 (±2.05)
	NRGNN [17]	72.15 (±3.51)	66.37 (±3.23)	68.70 (±2.04)	79.02 (±2.91)	79.16 (±2.74)
	GraphSS [34]	**88.51 (±2.68)**	**70.78 (±3.17)**	**86.06 (±1.29)**	**88.70 (±1.39)**	**88.47 (±0.91)**
GraphSAGE	Original	84.50 (±1.61)	72.58 (±0.94)	84.99 (±1.08)	91.99 (±0.96)	91.98 (±1.02)
	Attack [79]	36.48 (±7.72)	42.09 (±3.94)	44.63 (±2.19)	45.87 (±2.20)	42.70 (±1.10)
	AdvTrain [104]	47.78 (±7.58)	42.66 (±1.77)	53.54 (±1.99)	51.60 (±1.09)	62.70 (±1.83)
	GNN-Jaccard [31]	84.97 (±3.85)	69.99 (±1.96)	74.52 (±1.25)	81.81 (±2.06)	78.72 (±0.71)
	GNN-SVD [84]	66.03 (±4.19)	57.92 (±0.42)	69.87 (±1.33)	77.42 (±1.63)	72.25 (±1.47)
	RGCN [89]	41.98 (±0.69)	45.08 (±1.67)	47.04 (±2.50)	50.29 (±1.95)	57.23 (±1.43)
	GRAND [90]	32.38 (±3.35)	65.07 (±3.18)	68.25 (±1.13)	57.48 (±1.50)	67.54 (±2.34)
	ProGNN [91]	75.84 (±4.60)	68.29 (±2.02)	71.59 (±1.40)	82.53 (±0.96)	81.36 (±2.80)
	NRGNN [17]	73.02 (±4.26)	65.76 (±2.53)	69.08 (±1.85)	80.23 (±2.17)	78.54 (±2.66)
	GraphSS [34]	**89.29 (±2.77)**	**73.43 (±2.32)**	**87.55 (±1.16)**	**90.20 (±1.79)**	**88.75 (±0.85)**

Table 4.5. Analysis of The Average Runtime and The Unit Runtime (Per 100 nodes) in Seconds of GraphSS between Defense and Alert

		Cora		**Citeseer**		**Pubmed**		**AMZcobuy**		**Coauthor**	
		Average	**Unit**	**Average**	**Unit**	**Average**	**Unit**	**Average**	**Unit**	**Average**	**Unit**
GCN	Defense	22.75 (±0.65)	0.84	27.86 (±0.66)	0.84	163.57 (±3.29)	0.83	61.19 (±0.66)	0.80	146.90 (±1.07)	0.80
	Alert	21.28 (±0.31)	0.79	26.14 (±0.44)	0.79	155.84 (±1.57)	0.79	61.05 (±0.60)	0.80	146.27 (±1.83)	0.80
SGC	Defense	22.07 (±1.13)	0.81	28.06 (±0.45)	0.84	163.29 (±3.28)	0.83	61.51 (±0.55)	0.80	146.71 (±0.84)	0.80
	Alert	22.13 (±2.17)	0.82	26.20 (±0.54)	0.79	156.06 (±1.51)	0.79	61.01 (±0.61)	0.80	145.36 (±2.10)	0.79
GraphSAGE	Defense	22.04 (±0.91)	0.81	28.51 (±0.59)	0.86	165.44 (±0.86)	0.84	61.75 (±0.45)	0.81	147.28 (±0.77)	0.80
	Alert	21.38 (±0.22)	0.79	26.06 (±0.36)	0.78	154.94 (±1.04)	0.79	61.12 (±0.70)	0.80	147.14 (±1.36)	0.80

5. ROBUST NODE CLASSIFICATION ON GRAPHS: JOINTLY FROM BAYESIAN LABEL TRANSITION AND TOPOLOGY-BASED LABEL PROPAGATION

5.1 Introduction

Among the various machine learning tasks on the graph data, node classification is probably the most popular with numerous real-world applications, such as user profiling in online social networks (OSNs) [33], [103], [106], community detection [86], [107], expert finding in community-based question answering [108], [109], recommendation systems [110]–[112], and epidemiology study [113], [114]. While many traditional approaches have been used for solving a node classification task [115]–[118], in this deep learning era various graph neural network models (GNNs) [4], [45], [46], [48], [50] have become popular for solving this task. The enormous appeal of GNN models for node classification is due to their ability to learn effective node-level representation by performing locality-based non-linear aggregation jointly over node attributes and adjacency information leading to superior prediction performance.

However, our previous experimental results indicate that the performance of GNN-based node classification may be substantially deteriorated by changes in a graph structure [80], [119], which raises concern regarding the application of GNNs in various real-life scenarios. As in our previous example, in an online social network, non-malicious selfish users may randomly connect with many other nodes for promoting their business activities [33], [120]; such random connections exacerbate over-smoothing [121], [122], a well-known cause of the poor performance of GNNs. On the other extreme, the message-passing mechanism of GNNs performs poorly on nodes with sparse connections [123], [124], such as the nodes representing new users of OSNs, which lack sufficient links (edges) or profile information (feature). Third, GNNs may be vulnerable to adversarial attacks [78], [79], [94], [125], [126]. Such attacks may significantly deteriorate the performance of node classifications with slight and unnoticeable modifications of graph structures and node features. In this chapter, we refer to all such scenarios as **topological perturbations**.

Many recent works have been proposed to develop robust GNN-based node classification models against topological perturbations. Most of these works can be divided into two categories: the topological denoising methods, which design methodologies for denoising perturbed graphs in a pre-processing stage [31], [84] or in the training stage [23], [24], [127]–[129], and the mechanism design methods, which make GCN's prediction robust by message-passing-based aggregation over the nodes [25], [27], [29], [90], [130] or through regularization [28], [89], [91], [122], [131]–[134]. Nevertheless, none of these works can fully address the topological perturbations. For one thing, topological denoising methods may fail to prune suspicious edges when the graph structure is sparse. For another, mechanism designing methods sometimes highly depend on heuristics explorations of topological connections among nodes. The mechanism may perform worse when the graph structure is under severe perturbations.

In the previous chapters, we present the works [33], [34] that tackle the topological perturbation issues by developing a Bayesian label transition mechanism, which fixes the poor performance of GNNs by post-processing the prediction; specifically, their approaches learn a label transition matrix through Bayesian inferences, which fixes the erroneous prediction of GNNs by replacing the GNN inferred labels to a different label. A key benefit of post-processing is that it is effective for dynamic network scenarios, in which the network structure and node features of test data may be different than the train data. However, Bayesian label transition [34] suffers slow convergence as the posterior distribution from which the Gibbs sampling samples the node labels adapt incrementally over a sequence of label-propagation followed by GNN re-training iterations. Slow convergence also leads to inferior performance because the labels on which the GNN is re-trained may not be accurate as the Bayesian label transition may have not yet been converged.

Although label transition is a relatively new idea in the node classification task, label propagation (LP) has been widely used in the semi-supervised learning paradigm [135]–[144]. Recently, Huang et al. [144] demonstrated that graph semi-supervised learning using LP can exceed or match the performance of some variants of GNNs. Zhu and Ghahramani [135] have shown that label propagation is similar to mean-field approximation, which has fast convergence when the number of instances is large. The main assumption of LP is that closer

data points tend to have similar class labels—this assumption is analogous to the graph Homophily assumption, which denotes that adjacent vertices have similar labels. Specifically, on noisy and perturbed graphs, the label prediction on a given (or a targeted) node can be affected severely, however, the prediction of other neighboring nodes may still be sufficiently good, and by the virtue of homophily, those predictions can help recover the true label of the affected node. Besides label propagation, another similar technique is called relaxation labeling (RL), which associates a label with nodes in the graph to reduce the prediction or classification ambiguities [145]–[148]. Inspired by these observations, we improve the robustness of GNN-based node classification through an integrated framework combining Bayesian label transition and graph topology-based label propagation. We name our proposed model **LInDT** [1].

The main advantage of LInDT over a label-transition-based method, such as GraphSS [34], is that the former effectively utilizes LP when the sampling distribution of Bayesian label transition is uncertain. This immediately improves the label convergence and subsequently improves the label prediction performance. Besides, LInDT adopts a better Bayesian prior than GraphSS; specifically, LInDT follows an informative asymmetric Dirichlet distribution, but GraphSS follows a symmetric Dirichlet distribution, which allows LInDT to take advantage of the most up-to-date label distribution, leading to better performance. We evaluate LInDT's performance under random perturbations, information sparsity, and adversarial attack scenarios, across five public graph datasets. The experimental results verify the following facts: First, the GNN-based node classifier benefits strongly from LInDT's proposed label inference as post-processing; Second, LInDT converges faster than a pure Bayesian label transition which only uses Gibbs-based posterior sampling; Third, LInDT outperforms eight popular competing methods. We also perform an ablation study to verify the effectiveness of LInDT's design choices. For example, our experiment validates that asymmetric Dirichlet distributions of LInDT help better label inference using Bayesian label transition.

[1]↑LInDT is built by taking the bold letters of "**L**abel **In**ference using **D**irichlet and **T**opological sampling", and its pronunciation is similar to the popular chocolate brand name. Source code is available at **https://github.com/junzhuang-code/LInDT**

Overall, in this chapter, we elaborate on the justification of our proposed model, LInDT, including the model architecture and experimental results, and also summarize our contributions as follows:

- We propose a new label inference mechanism that can infer node labels jointly from a Bayesian and label propagation framework. Our model can converge faster than the Bayesian label transition using Gibbs sampling.
- We release the constraint of symmetric Dirichlet distributions on the Bayesian label transition and verify that dynamically updating the α vector (Dirichlet prior) can help better label inference.
- Extensive experiments demonstrate that our model achieves robust node classification on three topological perturbation scenarios and outperforms eight popular defending models across five public graph datasets.

5.2 Methodology

In this section, we first formally state the problem. Besides, we discuss asymmetric Dirichlet distributions and introduce our proposed model. Lastly, we present the pseudo-code and time complexity of our model.

5.2.1 Problem Statement

In general node classification tasks, a GNN-based node classifier f_θ takes both $\mathbf{A}$ and $\mathbf{X}$ as inputs and is trained with ground-truth labels in train data. To obtain robust classification, some studies attempt to train the model using noisy labels as a regularization scheme [149], [150]. In this chapter, we borrow this idea and train the GNN node classifier, f_θ, after assigning a uniform random label to a percentage (in this chapter we use 10%) of the train nodes. The higher the percentage of noise ratio, the stronger the regularization. After training the model with noisy labels $\mathcal{Y}$, we want the GNN to be able to predict the unobserved latent labels $\mathcal{Z}$. The model's performance is optimized to predict $\mathcal{Z}$ by using a validation set. It may seem counter-intuitive to train a model with noisy labels and still expect it to

perform well on the ground-truth label; however, this is feasible because both $\mathcal{Z}$ and $\mathcal{Y}$ take values from the same closed category set. Also, there is a strong correlation between $\mathcal{Y}$ and $\mathcal{Z}$ as only a fraction of the node's labels has been altered. But, as GNN is trained on the noisy labels, it becomes noise-resilient and robust and performs well when the network is perturbed.

In this chapter, we assume the test graph is perturbed through various structural changes. Thus, a GNN suffers performance loss in the perturbed test data. To overcome this, we apply the Bayesian inference approach on top of the GNN to fix erroneous predictions by learning a label transition matrix on the fly (from the test data). Specifically, after training the GNN with manual-annotated labels $\mathcal{Y}_m$ on unperturbed train graphs to generate auto-generated labels $\mathcal{Y}_a$, we employ our proposed label inference model to iteratively recover the perturbed predictions by $\mathcal{Y}_a$ with several epochs of transition. After the label transition is converged, we expect that the **inferred labels** $\bar{\mathcal{Z}}$ will be as identical as the latent labels. Overall, the above-mentioned problem could be formally described as follows:

Problem 1. *Given a graph $\mathcal{G} = (\mathbf{A}, \mathbf{X})$ and the manual-annotated labels $\mathcal{Y}_m$, which are used for training a GNN-based node classifier f_θ on unperturbed $\mathcal{G}_{train}$, we aim to develop a label inference model to improve the robustness of the node classifier f_θ on perturbed $\mathcal{G}_{test}$ by approximating the inferred labels $\bar{\mathcal{Z}}$ to the latent labels $\mathcal{Z}$ as identical as possible.*

5.2.2 Asymmetric Dirichlet Distributions

In Eq. 3.1, GraphLT [33] and GraphSS [34] assume the Dirichlet distribution is symmetric, i.e., the parameter vector α has the same value, 1.0, to all classes. In this chapter, we release this constraint and investigate whether adopting asymmetric Dirichlet distributions as a prior can contribute to the label transition. Fig. 5.1 visualizes the Dirichlet distributions by equilateral triangles. The first triangle represents the symmetric Dirichlet distribution. The second triangle shows that data points tend to move to the corner with the same probability. The third triangle presents that data points have a higher probability to be assigned to the class whose α value is higher. The intuition is that the ith inferred label $\bar{\mathbf{z}}_{\mathrm{i}}^{t}$ may have higher

probability to be transited to the kth class in the tth transition when α_k^t is higher. Following this intuition, we dynamically update the kth-class α value, α_k^t, in the tth transition as follows:

$$\alpha_k^t = \alpha_k^{t-1} \frac{\sum_{\mathrm{i}=1}^{N} I(\bar{\mathbf{z}}_{\mathrm{i}}^t = k)}{\sum_{\mathrm{i}=1}^{N} I(\bar{\mathbf{z}}_{\mathrm{i}}^{t-1} = k)}, \tag{5.1}$$

where $I(\cdot)$ denotes an indicator function.

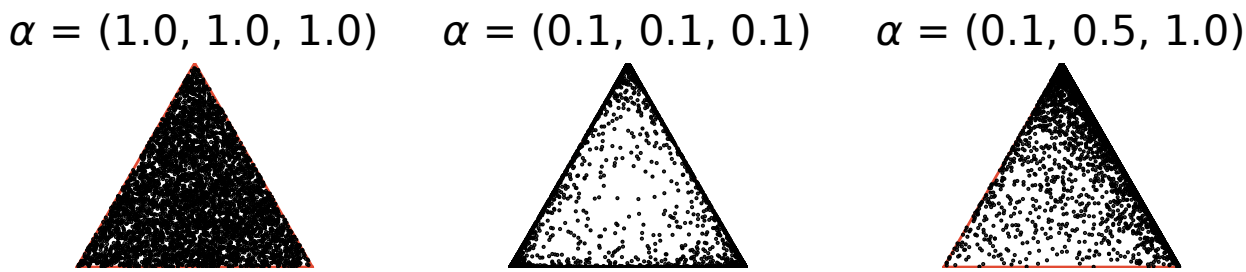


Figure 5.1. Toy Examples of Dirichlet Distributions

5.2.3 Label Inference Jointly from Bayesian Label Transition and Topology-based Label Propagation

GraphSS [34] applies Gibbs sampling to recover the multinominal distribution of nodes on perturbed graphs. However, such sampling may converge slowly and thus leads to inferior inference. This drawback can be mitigated by considering the fact that the majority of real-life networks exhibit homophily properties, i.e., labels of the adjacent nodes are similar. So, if the inferred label of a node during label transition is uncertain, we can sample its label from the labels of adjacent nodes. This idea is inspired by well-known Label propagation algorithms [139]. Thus, LInDT integrates both Bayesian label transition and topology-based label propagation. More specifically, for each node on $\mathcal{G}_{test}$, our model first infers the label from Bayesian label transition and then substitutes this label using topology-based label propagation when this inferred label is uncertain. We define the node label uncertainty during the inference as follows:

Definition 5.2.1. *In the tth transition, an inferred label $\bar{\mathbf{z}}^t$ is uncertain, such that $\bar{\mathbf{z}}^t \neq \bar{\mathbf{z}}^{t-1}$ or $\bar{\mathbf{z}}^t \neq \mathbf{y}$.*

Note that $\bar{\mathbf{z}}^t \neq \bar{\mathbf{z}}^{t-1}$ will happen when the inference is not converged yet, whereas $\bar{\mathbf{z}}^t \neq \mathbf{y}$ indicates that the inferred labels $\bar{\mathcal{Z}}$ still deviates from the auto-generated labels $\mathcal{Y}_a$. The intuition of our model is that unnoticeable perturbations may severely deteriorate the predictions of GNN-based node classifiers, but fortunately, such perturbations only alter a small number of topological structures. Based on the homophily assumption of the graph structure, i.e., intra-class nodes tend to connect with each other, sampling labels with topology-based label propagation can decrease the node label uncertainty by utilizing topological information of nodes. Note that we present the edge homophily ratio (**EHR** $\in [0, 1]$) [151], the percentage of intra-class edges, in Tab. 5.1 to verify the homophily property in the graph datasets. Higher EHR indicates stronger homophily properties.

We present how our model samples labels jointly from Bayesian Label Transition and Topology-based Label Propagation in Algo. 3. In the tth transition, we first sample a label $\bar{\mathbf{z}}_{\mathrm{i}}^t$ of the ith node from the categorical distribution $\overline{P}(\bar{\mathbf{z}}_{\mathrm{i}}^{t-1} \mid v_{\mathrm{i}}) \in \mathbb{R}^{N \times K}$ updated with a given label transition matrix ϕ^{t-1} in the $(t-1)^{th}$ transition (**Line 2**). We then update $\bar{\mathbf{z}}_{\mathrm{i}}^t$ with a given topology-based label sampler when this inferred label is uncertain (**Line 3-4**). In the end, we obtain the inferred labels in the tth transition.

Algorithm 3 LABEL Sampling Jointly from Bayesian Label Transition and Topology-based Label Propagation

Input: Categorical distribution $\overline{P}\left(\bar{\mathcal{Z}}^{t-1} \mid \mathcal{V}\right)$, Transition matrix ϕ^{t-1}, Topology-based label sampler.

1: **for** $\mathrm{i} \leftarrow 0$ **to** N **do**
2: $\quad \bar{\mathbf{z}}_{\mathrm{i}}^t \sim \arg\max \overline{P}\left(\bar{\mathbf{z}}_{\mathrm{i}}^{t-1} \mid v_{\mathrm{i}}\right) \phi^{t-1}$;
3: $\quad$ **if** $\bar{\mathbf{z}}_{\mathrm{i}}^t$ is uncertain **then**
4: $\quad\quad$ Update $\bar{\mathbf{z}}_{\mathrm{i}}^t$ with the topology-based label sampler;
5: $\quad$ **end if**
6: **end for**
7: **return** Inferred labels $\bar{\mathcal{Z}}^t$ in the tth transition.

Besides Algo. 3, we further use a toy example to explain our proposed sampling method in Fig. 5.2. In the tth transition, our model first samples the inferred label of node A as the red class via Bayesian label transition. However, this inferred label is different from the auto-generated label of node A; let's say it's blue. In this case, we say that this node label is

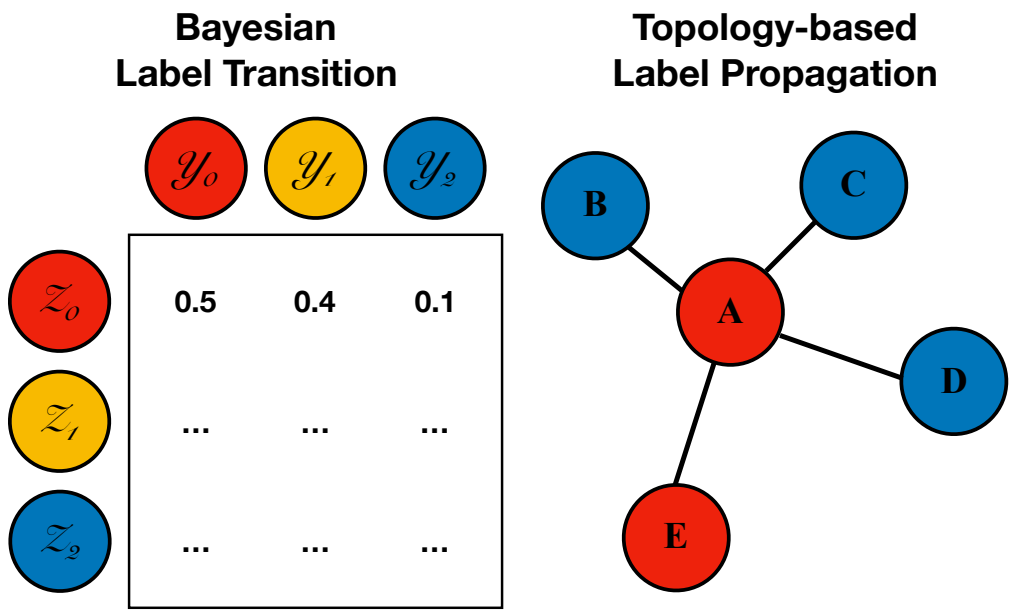


Figure 5.2. Toy Example of Our Proposed Sampling Method

uncertain. Based on the assumption of homophily graphs and unnoticeable perturbations, i.e., the labels of majority neighbor nodes are predicted correctly in $\mathcal{Y}_a$, we then apply the topology-based label sampler to substitute the label, i.e., replacing it from red to blue based on the topological information of node A.

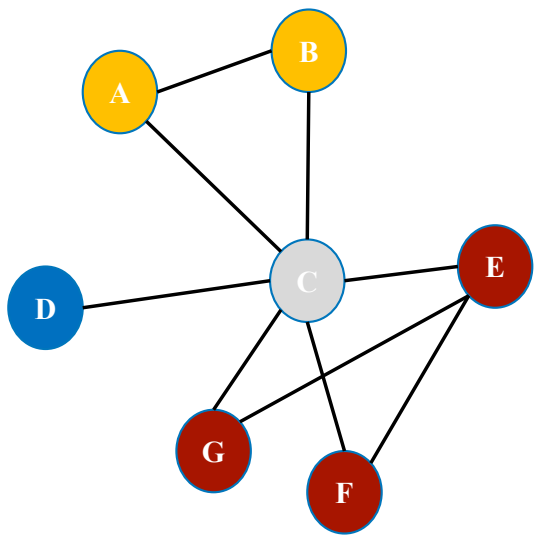
Based on the above-mentioned intuition, we investigate three types of topology-based label samplers, random sampler (**Random**), majority sampler (**Major**), and degree-weighted sampler (**Degree**). All of these samplers sample a label from one-hop neighborhoods of the current node. We denote the set of classes from the 1-hop neighborhood as $K_{n\mathrm{ei}}$ and the number of neighbor nodes as $N_{n\mathrm{ei}}$. We highlight the distinctions of these candidate samplers as follows and further explain them with toy examples in Fig. 5.3.

- Random sampler randomly samples a label. In the tth transition, the probability that the ith inferred label $\bar{\mathbf{z}}_{\mathrm{i}}^{t}$ is equal to k^{th} class can be formulated as Eq. 5.2.

$$P\left(\bar{\mathbf{z}}_{\mathrm{i}}^{t} = k \mid v_{\mathrm{i}}\right) = \frac{\sum_{\mathrm{i}=1}^{N_{n\mathrm{ei}}} I(\bar{\mathbf{z}}_{\mathrm{i}}^{t} = k)}{\sum_{\mathrm{i}=1}^{N_{n\mathrm{ei}}} I(\bar{\mathbf{z}}_{\mathrm{i}}^{t} \in K_{n\mathrm{ei}})}. \tag{5.2}$$

- Majority sampler selects the majority class k_{mj} as the label, such that the number of nodes in majority class $|v_{mj}| = max\{\sum_{\bar{\mathbf{z}}_j=k} \mathbf{A}_{ij} : k \in K_{nei}\}$.

- Degree-weighted sampler chooses a label from the class k_{dw}, such that the sum of the neighbor nodes' total degrees in k_{dw} is maximum. This maximum sum can be expressed as $max\{\sum_{\bar{\mathbf{z}}_i=k} d_i : k \in K_{nei}\}$, where $d_i = \sum_j \mathbf{A}_{ij}$ is the total degree of the ith node.



For example, we sample the node c using:

Random:

$$\bar{z}_c = K_{red} \text{ or } K_{yellow} \text{ or } K_{blue}$$

Major:

$$|v_{red}| = v_E + v_F + v_G = 3;$$
$$|v_{yellow}| = v_A + v_B = 2;$$
$$|v_{blue}| = v_D = 1.$$
$$\bar{z}_c = K_{red}$$

Degree:

$$d_{red} = d_E + d_F + d_G = 3 + 2 + 2 = 7;$$
$$d_{yellow} = d_A + d_B = 2 + 2 = 4;$$
$$d_{blue} = d_D = 1.$$
$$\bar{z}_c = K_{red}$$

Figure 5.3. Toy Examples of Our Three Proposed Samplers (We use red, yellow, and blue colors to represent three classes.)

Analysis of Convergence. To analyze the convergence of our proposed sampling method, we discuss two conjectures as follows.

Conjecture 1. *Given large enough iterations of transition T, the transition of inferred labels $\bar{\mathcal{Z}}$ will eventually converge to $\bar{\mathcal{Z}}^{\pi}$.*

Analysis: We formally state the transition as follows:

$$\bar{\mathcal{Z}}^{\pi} = \lim_{T\to\infty} \arg\max \overline{P}\left(\bar{\mathcal{Z}}^{T} \mid \mathcal{V}\right) = \lim_{T\to\infty} \arg\max \overline{P}\left(\bar{\mathcal{Z}}^{T-1} \mid \mathcal{V}\right) \phi^{T-1}. \tag{5.3}$$

Theoretical proof of this conjecture is difficult because both the categorical distribution $\overline{P}\left(\bar{\mathcal{Z}} \mid \mathcal{V}\right)$ and the label transition matrix ϕ in each transition will be dynamically updated. Instead, we empirically examine this conjecture in the experiment. Note that the converged

inferred labels $\bar{\mathcal{Z}}^{\pi}$ are not guaranteed to be the same as the latent labels $\mathcal{Z}$, which translates to the possibility of erroneous prediction.

Conjecture 2. *Label sampling jointly from Bayesian Label Transition and Topology-based Label Propagation helps the inference converge with fewer iterations of transition than the Bayesian-based sampling.*

Analysis: Our proposed sampling method will substitute the inferred labels of uncertain nodes based on the topological information. On the homophily graphs, we can get a smaller Total Variation Distance between current node label distributions of inferred labels in t^{th} transition $\bar{\mathcal{Z}}^{t}$ and the convergence distributions of inferred labels $\bar{\mathcal{Z}}^{\pi}$ as follows:

$$\|\bar{\mathcal{Z}}^{t}_{DT} - \bar{\mathcal{Z}}^{\pi}\|_{TV} \leq \|\bar{\mathcal{Z}}^{t}_{Bayes} - \bar{\mathcal{Z}}^{\pi}\|_{TV}, \tag{5.4}$$

where $\bar{\mathcal{Z}}^{t}_{DT}$ denotes the inferred labels using our proposed sampling method, whereas $\bar{\mathcal{Z}}^{t}_{Bayes}$ denotes the inferred labels using the Bayesian-based sampling method. $\|\cdot\|_{TV}$ denotes the Total Variation Distance.

We argue that this conjecture is true as such substitutions can force the inferred labels to get closer to the convergence based on the homophily assumption, shortening the iterations of transition compared to using the Bayesian-based sampling method, e.g., Gibbs sampling. Our experiments empirically verify this conjecture.

5.2.4 Pseudo-code of Our Proposed Model

To warp up our model in Algo. 4, we discuss the pseudo-code and analyze the time complexity in this section.

Training: Our model trains the node classifier f_θ on the train graph $\mathcal{G}_{train}$ at first with manual-annotated labels $\mathcal{Y}_m$ (**line 1**) and then generates initial categorical distribution $\overline{P}(\mathcal{Z} \mid \mathcal{V})$ and auto-generated labels $\mathcal{Y}_a$ by f_θ (**line 2**).

Inference: Before the inference, our model first computes a warm-up label transition matrix ϕ' by using the prediction over $\mathcal{G}_{train}$ (**line 3**). Our model then defines (creates empty spaces) the inferred labels $\bar{\mathcal{Z}}$, the dynamic label transition matrix ϕ based on the test graph $\mathcal{G}_{test}$, and also initializes the α vector with a given initial α value (**line 4**). In the warm-up stage,

Algorithm 4 LInDT's Pseudo-code

Input: Graph $\mathcal{G}_{train}$ and $\mathcal{G}_{test}$, which contain corresponding symmetric adjacency matrix $\mathbf{A}$ and feature matrix $\mathbf{X}$, Manual-annotated labels $\mathcal{Y}_m$ in $\mathcal{G}_{train}$, Node classifier f_θ, Initial α, The number of warm-up steps WS, The number of transition T.

1: Train f_θ with $\mathcal{Y}_m$ on $\mathcal{G}_{train}$;
2: Generate initial categorical distribution $\overline{P}(\mathcal{Z} \mid \mathcal{V})$ and auto-generated labels $\mathcal{Y}_a$ by f_θ;
3: Compute warm-up label transition matrix ϕ' based on $\mathcal{G}_{train}$;
4: Define inferred labels $\bar{\mathcal{Z}}$, dynamic label transition matrix ϕ based on $\mathcal{G}_{test}$, initial α vector;
5: **for** $t \leftarrow 1$ **to** T **do**
6: **if** $t < WS$ **then**
7: Sample $\bar{\mathcal{Z}}^t$ with warm-up ϕ' by Algo. 3;
8: **else**
9: Sample $\bar{\mathcal{Z}}^t$ with dynamic ϕ by Algo. 3;
10: **end if**
11: Update α by Eq. 5.1 and dynamic ϕ;
12: Retrain f_θ and update $\overline{P}\left(\bar{\mathcal{Z}}^t \mid \mathcal{V}\right)$;
13: **end for**
14: **return** Inferred labels $\bar{\mathcal{Z}}$ and Dynamic ϕ.

our model samples the inferred labels in the tth transition with the warm-up ϕ', which is built with the categorical distribution of f_θ and $\mathcal{Y}_m$ on $\mathcal{G}_{train}$, via corresponding topology-based label sampler using Algo. 3 (**line 7**). After the warm-up stage, our model applies the same sampling method with the dynamic ϕ (**line 9**). This dynamic ϕ updates in each transition with current inferred labels $\bar{\mathcal{Z}}^t$ and corresponding $\mathcal{Y}_a$. Simultaneously, our model dynamically updates the α vector by Eq. 5.1 and the dynamic ϕ matrix (**line 11**). Besides, our model iteratively re-trains the node classifier with the current inferred labels $\bar{\mathcal{Z}}^t$ in the tth transition, and then updates the categorical distribution $\overline{P}\left(\bar{\mathcal{Z}}^t \mid \mathcal{V}\right)$ modeled by the re-trained f_θ (**line 12**). The transition will eventually reach convergence, approximating the inferred labels to the latent labels as identically as possible. In other words, our model can help recover the original predictions by dynamic conditional label transition on perturbed graphs.

According to Algo. 4, our model applies T transitions in the inference. For each transition, our proposed sampling method traverses all test nodes once. Overall, the **time complexity**

of the inference approximates to $\mathcal{O}(T \cdot N_{test})$, where T denotes the number of transitions, whereas N_{test} denotes the number of nodes on the test graph.

5.3 Experiments

In this section, we want to answer the following questions for evaluating our model.

- Can node classification benefit from our model against topological perturbations?
- How's the convergence of our model?
- How's our performance compared to competing methods?
- Can asymmetric Dirichlet distributions contribute to the label inference?

Before answering these questions, we first introduce the experimental settings: Dataset, Implementation of topological perturbations, Competing methods, and Evaluation metrics.

Table 5.1. Statistics of Datasets ($|\mathcal{V}|$, $|\mathcal{E}|$, $|F|$, and $|C|$ denote the number of nodes, edges, features, and classes, respectively. Avg.D denotes the average degree of test nodes. EHR denotes the edge homophily ratio.)

Dataset	$\|\mathcal{V}\|$	$\|\mathcal{E}\|$	$\|F\|$	$\|C\|$	Avg.D	EHR(%)
Cora	2,708	10,556	1,433	7	4.99	81.00
Citeseer	3,327	9,228	3,703	6	3.72	73.55
Pubmed	19,717	88,651	500	3	5.50	80.24
AMZcobuy	7,650	287,326	745	8	32.77	82.72
Coauthor	18,333	327,576	6,805	15	10.01	80.81

5.3.1 Datasets

Tab. 5.1 presents statistics of five public node-labeled graph datasets. **Cora**, **Citeseer**, and **Pubmed** are famous citation graph data [88]. **AMZcobuy** comes from the photo segment of the Amazon co-purchase graph [87]. **Coauthor** is co-authorship graphs of computer science based on the Microsoft Academic Graph from the KDD Cup 2016 challenge [2]. For all datasets, the percentage of the train, validation, and test partition comprise 10%,

[2]↑https://www.kdd.org/kdd-cup/view/kdd-cup-2016

20%, and 70% of the nodes, respectively. Similar to GraphSS [34], only the train graphs contain manual-annotated labels, but the validation and test graphs don't. We simulate the manual-annotated labels by randomly replacing the ground-truth labels of 10% (noise ratio nr) nodes with another label, chosen uniformly.

5.3.2 Implementation of Topological Perturbations

We examine our model under three scenarios of topological perturbations as follows.

- Random perturbations (**rdmPert**): perturbators in OSNs intend to randomly connect with many other normal users for commercial purposes, e.g., new brand promotion. In the experiments, we limit the number of perturbators to 1% of the validation/test nodes (a.k.a. victim nodes) and restrict the number of connections from each perturbator to 100. Note that rdmPert doesn't apply gradient-based attacks, such as FGSM [92], PGD [11], etc.

- Information sparsity (**infoSparse**): we sparse the 90% links and 100% features of the victim nodes ($L\&F$) on validation/test graphs.

- Adversarial attacks (**advAttack**): we execute node-level direct evasion non-targeted attacks [79] on both links and features ($L\&F$) of the victim nodes on sparse graphs, whereas the trained node classifier remains unchanged. To ensure the sparsity, we sparse the denser graphs (AMZcobuy and Coauthor) and select the victim nodes whose total degrees are within (0, 10) for attacks. The intensity of perturbation n_{pert} is set as 2 for all datasets. The ratio of n_{pert} between applying on links and applying on features is 1: 10.

5.3.3 Competing Methods

We first compare our model with seven popular competing methods which develop robust GNN-based node classifiers using topological information. Also, we consider MC Dropout as one of the competing methods for measuring uncertainty purposes. The hyper-parameters of GNN in our model follow the same settings as that in Chapter 4. We present the hyper-

parameters of competing methods for reproducibility purposes. All models are trained by Adam optimizer.

- **GNN-Jaccard** [31] preprocesses the graph by eliminating suspicious connections, whose Jaccard similarity of nodes features is smaller than a given threshold. The similarity threshold is 0.01. The hidden units are 16. The dropout rate is 0.5. The training epochs are 300.

- **GNN-SVD** [84] proposes another preprocessing approach with low-rank approximation on the perturbed graph to mitigate the negative effects from high-rank attacks, such as Nettack [79]. The number of singular values is 15. Hidden units are 16. The dropout rate is 0.5. The training epochs are 300.

- **DropEdge** [23] randomly removes a number of edges from the input graph in each training epoch. We choose GCN as the base model with 1 base block layer and train this model with 300 epochs. The rest of the parameters are mentioned in Tab. 5.2.

- **GRAND** [90] proposes random propagation and consistency regularization strategies to address the issues of over-smoothing and non-robustness of GCNs. The model is trained with 200 epochs. The hidden units, drop node rate, and L2 weight decay are 32, 0.5, and 5×10^{-4}, respectively. The rest of the parameters are the same as that in Chapter 4.

- **RGCN** [89] adopts Gaussian distributions as the hidden representations of nodes to mitigate the negative effects of adversarial attacks. We set up γ as 1, β_1 and β_2 as 5×10^{-4} on all datasets. The hidden units for each dataset are 64, 64, 128, 1024, and 1024, respectively. The dropout rate is 0.6. The learning rate is 0.01. The training epochs are 400.

- **ProGNN** [91] jointly learns the structural graph properties and iteratively reconstructs the clean graph to reduce the effects of adversarial structure. α, β, γ, and λ are 5×10^{-4}, 1.5, 1.0, and 0.0, respectively. The hidden units are 16. The dropout

rate is 0.5. The learning rate is 0.01. Weight decay is 5×10^{-4}. The training epochs are 100.

- **GDC** [122] proposes an adaptive connection sampling method using stochastic regularization for training GNNs. This method can learn with uncertainty on graphs. The number of blocks and layers is 2 and 4, respectively. The hidden units are 32. The dropout rate is 0.5. Both learning rate and weight decay are 5×10^{-3}. The training epochs are 400.

- **MC Dropout** [152] develops a dropout framework that approximates Bayesian inference in deep Gaussian processes. We use the same GCN architecture and training setting as ours except for the optimal dropout rate for each dataset as 0.7, 0.6, 0.9, 0.6, and 0.8, respectively. Note that MC Dropout will apply dropout in test data.

Table 5.2. Hyper-parameters of DropEdge in This Chapter

Hyper-parameters	**Cora**	**Citeseer**	**Pubmed**	**AMZcobuy**	**Coauthor**
Hidden units	128	128	128	256	128
Dropout rate	0.8	0.8	0.5	0.5	0.5
Learning rate	0.01	9e-3	0.01	0.01	0.01
Weight decay	5e-3	1e-3	1e-3	0.01	1e-3
Use BN	×	×	×	✓	×

5.3.4 Evaluation Metrics

We use both accuracy (Acc.) and average normalized entropy (Ent.) to measure the robustness of a node classifier. More specifically, Acc. evaluates the node classification performance (the higher the better), whereas Ent. represents the uncertainty of nodes' categorical distributions (the lower the better).

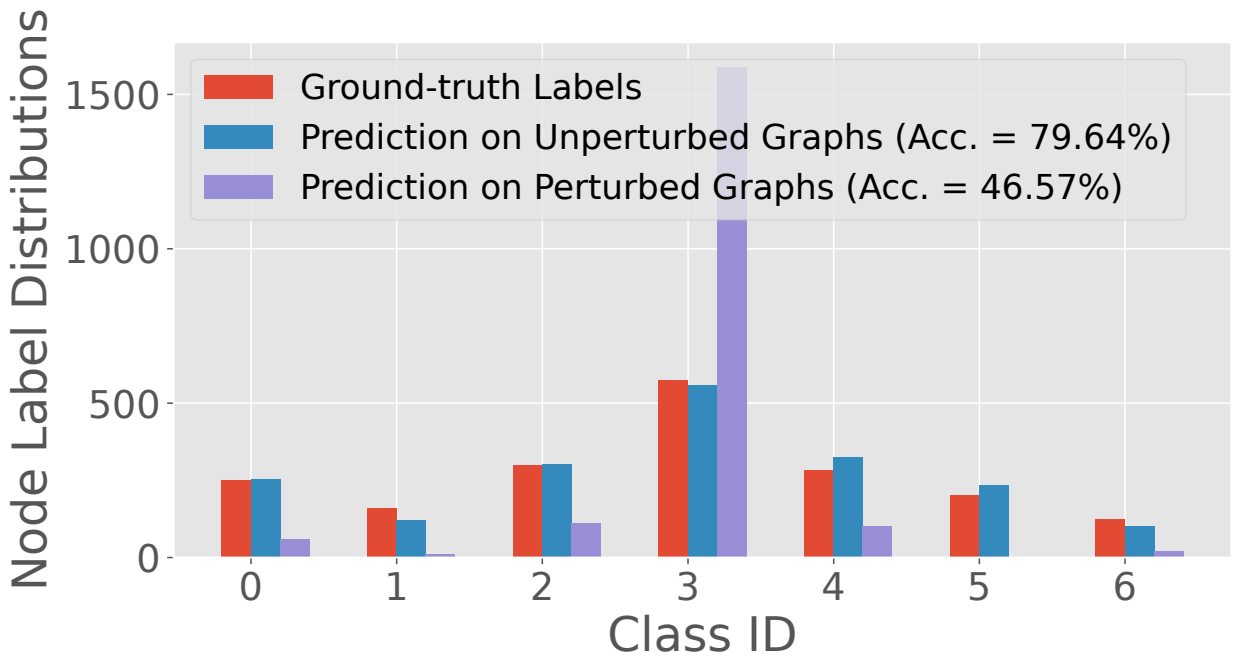


Figure 5.4. Node Label Distributions of Cora (Test nodes)

5.3.5 Node Classification Benefits from Our Model against Topological Perturbations

The performance of a node classifier may degrade on perturbed graphs. We present the comparison of node label distributions of test nodes among the ground-truth labels, the prediction on unperturbed graphs, and the prediction on perturbed graphs (under rdmPert) in Fig. 5.4 as an example. This comparison indicates that node label distributions may significantly change on perturbed graphs, causing performance degradation. In this chapter, our goal is to improve the robustness of the node classifier on perturbed graphs. In other words, we aim to increase the classification accuracy (Acc.) while maintaining the uncertainty (Ent.) at a low level.

To answer the first question, we examine whether the node classifier can benefit from our model under three scenarios of topological perturbations, random perturbations (rdmPert), information sparsity (infoSparse), and adversarial attacks (advAttack). We compare the performance between the baseline model, GraphSS [34], and the variants of our model archi-

tectures. The denotations of related methods are mentioned in the caption of Tab. 5.3. The results affirmatively verify that our model can outperform the baseline model and successfully achieve our goal in most cases. Besides, we have several observations as follows. At first, topological samplers could further boost accuracy under both rdmPert and advAttack but may fail under infoSparse because most links and features are sparsified under this scenario. Moreover, the majority sampler attains higher accuracy than the degree-weighted sampler on sparse graphs (Cora, Citeseer, and Pubmed) in most cases. Such a situation may be largely reversed on denser graphs (AMZcobuy and Coauthor) because the degree-weighted sampler can further take advantage of degree information from neighbors. Furthermore, the uncertainty may be lower under some scenarios, e.g., Ent. is 1.43% under advAttack on Cora. However, such kind of low uncertainty reveals that the node classifier may suffer severe perturbations, i.e., many nodes are certainly assigned to incorrect classes. Thus, lower uncertainty couldn't fully indicate robust performance. Last but not least, our model couldn't significantly decrease the uncertainty under rdmPert on Citeseer.

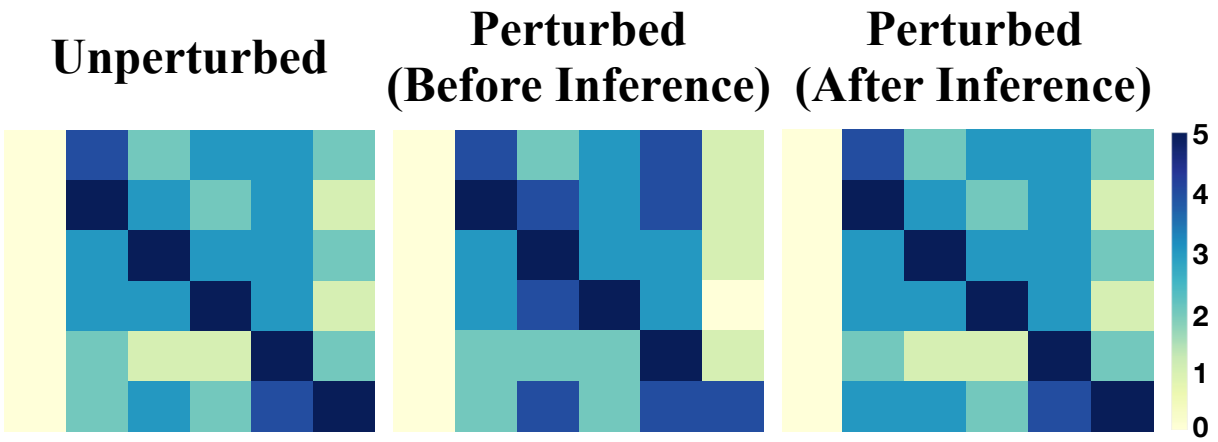


Figure 5.5. Visualization of Node Label Distributions on Citeseer via Log-scale Fine-grained Confusion Matrices

We further investigate this unusual case by visualizing the node label distributions via confusion matrices in Fig. 5.5. The left matrix presents the predicted node label distributions on unperturbed graphs, whereas the middle and right matrices present the predicted and inferred distributions on perturbed graphs, respectively. The visualization first shows that

our model can recover the distribution as close as that on unperturbed graphs when the node label distribution is perturbed. These confusion matrices also indicate that the node classification on the unperturbed graphs already misses one class, leading to inaccurate prior distribution. The label inference may probably fail to decrease the uncertainty due to this reason. Besides, this case reveals two limitations of our model. First, the inference highly depends on an accurate prior distribution. Second, our model couldn't recover the missing class, which implies that our model currently cannot handle the open-set classification.

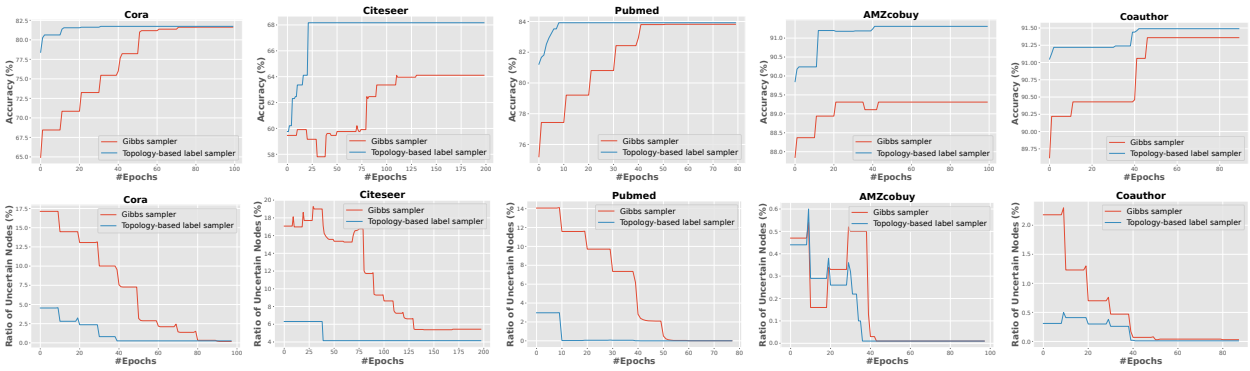


Figure 5.6. Empirical Analysis of Convergence between Gibbs Sampler and Topology-based Label Sampler (Major)

5.3.6 Empirical Analysis of Convergence

To answer the second question, we conduct an empirical analysis of convergence to examine two proposed conjectures. Fig. 5.6 displays the empirical comparison between using the Gibbs sampler and using the topology-based label sampler (Major) during the label inference under rdmPert. The first row presents the curves of validation accuracy (%). The second row displays the curves of the ratio of uncertain nodes (%) on inferred labels. According to the results, the label inference using both sampling methods can eventually converge across five datasets. The convergence verifies that the first conjecture holds true. Also, the results show that the label inference using the topology-based label sampler reaches the convergence with fewer iterations of transition compared to using the Gibbs sampler. In other words, using the topology-based label sampler can help decrease the ratio of uncertain nodes faster, and further reduces the fluctuations of the curve. The comparison verifies the second conjecture.

5.3.7 Comparison with Competing Methods

Tab. 5.4 presents the comparison of performance between competing methods and our model under rdmPert. For our model, we present the average values of three topological samplers. The results verify that our model is superior to competing methods across five datasets. Most competing methods fail to decrease the uncertainty (Ent.) but MC Dropout obtains robust performance among them. Our model can outperform MC Dropout, achieving higher classification accuracy with lower uncertainty. We notice that DropEdge and RGCN perform higher accuracy on larger graphs. For DropEdge, randomly dropping a certain number of edges is equivalent to data augmentation, which will be performed stronger when the size of the input graph is getting larger. RGCN adopts Gaussian distributions in hidden layers to reduce the negative impacts that come from the shift of node label distribution. The Gaussian-based hidden matrix has a higher capability to mitigate the negative impacts on larger graphs.

From the perspective of total runtime (in second), we observe that topological denoising methods (GNN-Jaccard, GNN-SVD, and DropEdge) and MC dropout consume much less time than other methods. These are reasonable because topological denoising methods mainly process the input graph structure, and MC dropout can speed up the training by dropping some hidden units. On the contrary, ProGNN consumes much more time as the size of the graphs increases. One of the reasons is that ProGNN iteratively reconstructs the graph by preserving low rank and sparsity properties. Such reconstructions will take a much longer time on larger graphs.

5.3.8 Analysis of Parameters

In this chapter, we maintain the same values of both warm-up steps WS, 40, and re-training epochs $Retrain$, 60, as that in GraphSS [34]. To avoid over-fitting, we retrain the node classifier every 10 epochs during the inference. In the meanwhile, we analyze how the initial α value affects accuracy, and conduct this analysis on the validation set. We fix the number of transition states T as [100, 200, 80, 100, 90] for five datasets, respectively. Fig. 5.7 presents two curves of accuracy along different initial α values. The blue curve denotes the

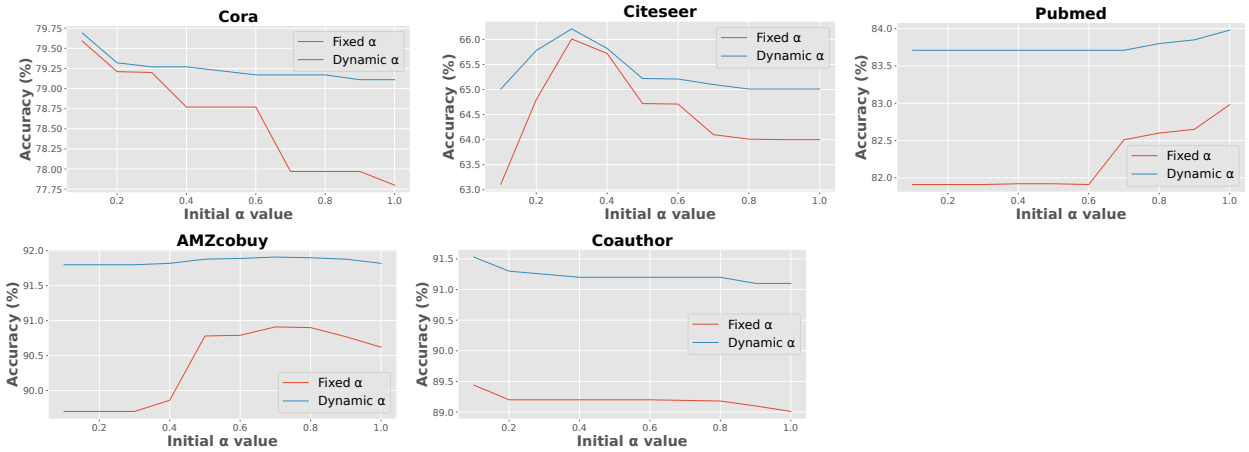


Figure 5.7. Analysis of The Initial α Value for The Dynamic α Vector

case that we dynamically update the α vector, whereas the red curve denotes the case with fixed α values. The results affirmatively answer the fourth question that adopting asymmetric Dirichlet distributions as a prior can contribute to the label inference using Bayesian label transition. Besides, the comparison between the two curves verifies that the dynamic α mechanism can further boost the accuracy in most cases. Based on this analysis, we select the initial α values as [0.1, 0.3, 1.0, 0.7, 0.1] for five datasets, respectively.

5.3.9 Limitation and Future Directions

1) Using the topology-based sampler may fail to boost the classification accuracy on extremely sparse graphs where most links and features are sparsified or missing. 2) Our label inference model highly depends on an accurate prior distribution. In other words, our model couldn't handle the poisoning attacks. In the future, integrating a denoising approach on the prior distribution would be a possible solution to mitigate this issue. 3) Our model couldn't recover the missing class from the prior distribution. This limitation implies that our model currently cannot handle the open-set classification. In the future, we could jointly utilize a distance-based label transition matrix to detect potential unseen classes during the inference.

5.4 Chapter Summary

In this chapter, we aim to improve the robustness of a GNN-based node classifier against topological perturbations given that the node classifier is trained with manual-annotated labels. To achieve this goal, we propose a new label inference model, namely LInDT, that integrates both Bayesian label transition and topology-based label propagation. Extensive experiments demonstrate the following conclusions. Firstly, GNN-based node classification can benefit from our model under three scenarios of topological perturbations. Furthermore, our model converges faster, achieves higher accuracy on node classification while maintaining normalized entropy at a low level, and surpasses eight popular competing methods across five public graph datasets. Lastly, adopting asymmetric Dirichlet distributions as a prior can contribute to label inference.

Table 5.3. Examination of Our Model on Top of GCN under Three Scenarios of Topological Perturbations across Five Datasets ("Original" denotes the original performance of GCN on perturbed graphs (Before inference). GraphSS is our baseline model with a fixed α value, 1.0. "Vanilla" denotes that we dynamically update the α vector using Gibbs samplers. "Random", "Major", and "Degree" denote that we employ the corresponding topology-based label sampler based on our vanilla architecture instead of using Gibbs samplers. All methods are evaluated by classification accuracy (Acc.) and average normalized entropy (Ent.) on victim nodes.)

	Meth-ods	**Cora**		**Citeseer**		**Pubmed**		**AMZcobuy**		**Coauthor**	
		Acc.	**Ent.**	**Acc.**	**Ent.**	**Acc.**	**Ent.**	**Acc.**	**Ent.**	**Acc.**	**Ent.**
rdmPert	Original	48.95	**9.24**	45.92	62.17	25.13	**3.59**	81.25	43.83	38.63	17.10
	GraphSS [34]	82.61	18.34	31.33	**9.35**	81.33	25.25	83.56	7.93	88.42	6.60
	Vanilla	83.68	18.34	56.22	40.35	82.45	23.86	84.68	**7.41**	89.66	**5.78**
	Random	**84.74**	21.49	**71.24**	53.71	83.54	32.41	91.91	12.61	90.58	6.86
	Major	84.21	22.22	66.95	66.99	**83.64**	32.97	**91.93**	12.79	90.59	7.19
	Degree	83.63	26.92	65.32	66.81	83.52	32.32	**91.93**	12.24	**90.62**	7.20
infoSparse	Original	71.73	47.87	62.26	90.13	76.26	57.65	90.01	13.48	86.62	38.34
	GraphSS [34]	79.32	**27.96**	**69.77**	**46.81**	84.03	29.64	90.81	7.10	90.15	8.77
	Vanilla	**79.48**	**27.96**	**69.77**	**46.81**	84.05	29.64	91.90	7.25	91.06	8.54
	Random	**79.48**	**27.96**	**69.77**	**46.81**	84.09	29.74	91.91	7.26	91.20	8.71
	Major	**79.48**	**27.96**	**69.77**	**46.81**	**84.10**	29.56	91.91	**6.88**	91.22	**8.29**
	Degree	**79.48**	**27.96**	**69.77**	**46.81**	84.07	**28.38**	**91.93**	6.89	**91.24**	8.63
advAttack	Original	33.86	1.43	4.31	37.29	23.55	11.83	77.17	49.25	58.69	36.05
	GraphSS [34]	38.10	**1.24**	54.31	**13.04**	83.04	**9.89**	81.50	13.55	74.52	8.35
	Vanilla	39.68	**1.24**	66.38	13.42	85.13	**9.89**	82.61	13.72	76.39	8.23
	Random	76.72	8.59	69.84	15.47	85.70	11.48	84.78	13.47	80.28	8.05
	Major	78.84	9.69	**71.98**	15.25	**85.87**	10.87	84.78	**12.36**	**80.31**	6.98
	Degree	**80.95**	7.15	70.26	15.23	85.51	11.21	**84.79**	13.44	79.53	**6.90**

Table 5.4. Comparison between Competing Methods and Our Model under The Random Perturbations Scenario (Acc. (%) denotes classification accuracy. Ent. (%) denotes the average normalized entropy. Time (s) denotes total runtime.)

Methods	Cora			Citeseer			Pubmed			AMZcobuy			Coauthor		
	Acc.	**Ent.**	**Time**	**Acc.**	**Ent.**	**Time**	**Acc.**	**Ent.**	**Time**	**Acc.**	**Ent.**	**Time**	**Acc.**	**Ent.**	**Time**
GNN-Jaccard [31]	66.32	93.24	2.83	56.65	95.47	2.34	60.68	81.77	3.71	53.12	96.17	5.83	88.47	96.88	7.44
GNN-SVD [84]	50.53	93.01	3.05	31.76	95.20	3.48	74.22	88.39	12.96	70.02	93.52	5.01	74.22	97.21	11.51
DropEdge [23]	67.88	95.28	**1.89**	46.78	96.44	**1.46**	77.34	76.66	2.56	63.42	96.25	2.61	71.42	97.48	**2.90**
GRAND [90]	52.34	94.99	8.72	35.02	95.35	5.69	49.75	89.89	12.15	40.22	96.23	27.14	55.56	97.73	154.17
RGCN [89]	62.63	93.72	5.04	62.23	96.39	5.17	83.20	89.91	27.71	77.87	97.36	30.85	89.33	98.39	179.65
ProGNN [91]	52.63	92.09	174.75	36.18	96.32	294.93	50.11	88.35	2135.67	45.28	98.63	1914.42	57.39	96.55	2366.04
GDC [122]	71.18	85.77	12.30	43.15	93.73	17.52	48.95	32.94	258.34	45.58	98.18	80.84	62.65	94.02	205.22
MC Dropout [152]	80.53	40.25	2.41	64.81	89.80	2.74	79.51	72.73	**1.99**	90.68	39.21	**1.81**	88.40	43.30	5.79
LInDT [35]	**84.19**	**23.54**	29.35	**67.84**	**62.50**	70.73	**83.57**	**32.57**	158.25	**91.92**	**12.55**	75.28	**90.60**	**7.08**	184.15

6. NON-EXHAUSTIVE LEARNING USING GAUSSIAN MIXTURE GENERATIVE ADVERSARIAL NETWORKS

6.1 Introduction

In earlier chapters, we presented a series of Bayesian label transition models to address GNNs' robustness issues in label scarcity and perturbations. In this chapter, we further present a work that studies another robustness problem in an open-set environment, a new machine-learning paradigm. In this paradigm, several classes in the test data are unseen during training, making conventional supervised-learning models vulnerable during inference on the test data. Thus, studying this paradigm is substantially significant since many realistic machine learning problems originate in non-stationary environments where instances of unseen classes may emerge naturally. The presence of such instances weakens the robustness of conventional machine learning algorithms, as these algorithms do not account for instances from unknown classes, either in the train or the test environments. To overcome this challenge, a series of related research activities has become popular in recent years; examples include anomaly detection (AD) [153]–[156], few-shot learning (FSL) [157], [158], zero-shot learning (ZSL) [159], [160], open set recognition (OSR) and open-world classification (OWC) [69]–[75], [161]–[167]. Collectively, each of these works belongs to one of the four different categories [68], differing on the kind of instances observed by the model during training and testing. If L refers to labeling and I refers to self-information (e.g., semantic information in image dataset), the categories C can be denoted as the Cartesian product of two sets L and I, as shown below:

$$C = L \times I = \{(l, \mathrm{i}) \ : \ l \in L \ \& \ \mathrm{i} \in I\}, \tag{6.1}$$

both L and I have two elements: known (K) and unknown (U). Thus, there are four categories in C: (K, K), (K, U), (U, K), (U, U). For example, (U, U) refers to the learning problem in which instances belonging to unknown classes and having no self-information.

Conventional supervised learning task belongs to the first category, as for such a task all instances in train and test datasets belong to (K, K). The anomaly detection (AD) task, a.k.a. one-class classification or outlier detection, detects a few (U, U) instances from the

Table 6.1. The Background of Related Tasks (Conv. for conventional method)

Tasks	Training Set	Testing Set	GOAL
Conv.	(K, K)	(K, K)	Supervised learning with (K, K)
AD	(K, K) w./wo. outliers	(K, K) w. outliers	Detect outliers
FSL	(K, K) w. limited (U, K)	(U, K)	Identify (U, K) in test set
ZSL	(K, K) w. self-info.	(U, K)	Identify (U, K) in test set
OSR	(K, K)	(K, K) & (U, U)	Distinguish (U, U) from (K, K)
NEL	(K, K)	(K, K) & (U, U)	Incrementally learn (U, U)

majority of (K, K) instances; for AD, the (U, U) instances may only (but not necessary) exist in the test set. FSL and ZSL are employed to identify (U, K) instances in the test set. The main difference between FSL and ZSL is that the training set of FSL contains a limited number of (U, K) instances while for the case of ZSL, the number of (U, K) instances in the train set is zero. In other words, ZSL identifies (U, K) instances in the test set only by associating (K, K) instances with (U, K) instances through self-information. Finally, works belonging to open set recognition (OSR) identify (U, U) instances in the test set. These works are the most challenging; unlike AD, whose objective is to detect only one class (outlier), OSR handles both (K, K) and (U, U) in the test set. Similar to OSR, OWC also incrementally learns the new classes and rejects the unseen class. Nevertheless, most existing methods of OSR or OWC do not distinguish the test instances among incremental unseen classes, which is closer to the realistic scenario. The scope of our work falls in the OSR category which only deals with (K, K) and (U, U) instances. In Table 6.1, we present a summary of the discussion of this paragraph.

Several works belonging to OSR have also been referred as Non-Exhaustive Learning (NEL). The term, Non-Exhaustive, means that the training data does not have instances of all classes that may be expected in the test data. The majority of early research works of NEL employ Bayesian methods with Gaussian mixture model (GMM) or infinite Gaussian mixture model (IGMM) [76], [77]. However, these works suffer from some limitations; for instance, they assume that the data distribution in each class follows a mixture of Gaussian, which may not be true in many realistic datasets. Also, in the case of GMM, its ability to

recognize unknown classes depends on the number of initial clusters that it uses. IGMM can mitigate this restriction by allowing cluster count to grow on the fly, but the inference mechanism of IGMM is time-consuming, no matter what kind of sampling method it uses for inferring the probabilities of the posterior distribution.

To address these issues, in this chapter, we propose a new non-exhaustive learning model, Non-exhaustive Gaussian mixture Generative Adversarial Networks (NE-GM-GAN), which synthesizes the Bayesian method and deep learning technique. Compared to the existing methods for OSR, our proposed method has several advantages: First, NE-GM-GAN takes multi-modal prior as input to better fit the real data distribution; Second, NE-GM-GAN can deal with class-imbalance problem with end-to-end offline training; Finally, NE-GM-GAN can achieve accurate and robust online detection on large sparse dataset while avoiding noisy distraction. Extensive experiments demonstrate that our proposed model has superior performance over competing methods on benchmark datasets. The contribution of this chapter can be summarized as follows:

- We propose a new model for non-exhaustive learning, namely NE-GM-GAN, which can detect novel classes in online test data accurately and defy the class imbalance problem effectively.

- NE-GM-GAN integrates Bayesian inference with distance-based and threshold-based methods to estimate the number of emerging classes in the test data. It also devises a novel scoring method to distinguish the UCs (unknown classes) from KCs (known classes).

- Extensive experiments on four datasets (3 real and 1 synthetic) demonstrate that our model is superior to existing methods for accurate and robust online detection of emerging classes in streaming data.

6.2 Background

6.2.1 Generative Adversarial Networks (GAN)

Vanilla GAN [6] consists of two key components, a generator $\mathcal{G}$, and a discriminator $\mathcal{D}$. Given a prior distribution Z as input, $\mathcal{G}$ maps an instance $\mathbf{z} \sim Z$ from the latent space to the data space as $\mathcal{G}(\mathbf{z})$. On the other hand, $\mathcal{D}$ attempts to distinguish a data instance $\mathbf{x}$ from a synthetic instance $\mathcal{G}(\mathbf{z})$, generated by $\mathcal{G}$. We use the terminology $p_Z(\mathbf{z})$ to denote that $\mathbf{z}$ is a sampled instance from the distribution Z. The training process is set up as if $\mathcal{G}$ and $\mathcal{D}$ are playing a zero-sum game, a.k.a. minimax game; $\mathcal{G}$ tries to generate the synthetic instances that are as close as possible to actual data instances; on the other hand, $\mathcal{D}$ is responsible for distinguishing the real instances from the synthetic instances. In the end, GAN converges when both $\mathcal{G}$ and $\mathcal{D}$ reach a Nash equilibrium; at that stage, $\mathcal{G}$ learns the data distribution and is able to generate data instances that are very close to the actual data instances. The objective function of GAN can be written as follows:

$$\min_{\mathcal{G}} \max_{\mathcal{D}} V(\mathcal{D}, \mathcal{G}) = \mathop{\mathbb{E}}_{\mathbf{x} \sim X}[\log \mathcal{D}(\mathbf{x})] + \mathop{\mathbb{E}}_{\mathbf{z} \sim Z}[\log(1 - \mathcal{D}(\mathcal{G}(\mathbf{z})))], \tag{6.2}$$

where X is the distribution of $\mathbf{x}$ and Z is the distribution from which $\mathcal{G}$ samples.

6.2.2 Bidirectional Generative Adversarial Networks (BiGAN)

Besides training a generator $\mathcal{G}$, BiGAN [53] also trains an encoder $\mathcal{E}$, that maps real instances $\mathbf{x}$ into latent feature space $\mathcal{E}(\mathbf{x})$. Its discriminator $\mathcal{D}$ takes both $\mathbf{x}$ and $p_Z(\mathbf{z})$ as input in order to match the joint distribution $p_{\mathcal{G}}(\mathbf{x}, \mathbf{z})$ and $p_{\mathcal{E}}(\mathbf{x}, \mathbf{z})$. The objective function of BiGAN can be written as follows:

$$\min_{\mathcal{G},\mathcal{E}} \max_{\mathcal{D}} V(\mathcal{D}, \mathcal{E}, \mathcal{G}) = \mathop{\mathbb{E}}_{\mathbf{x} \sim X}[\log \mathcal{D}(\mathbf{x}, \mathcal{E}(\mathbf{x}))] + \mathop{\mathbb{E}}_{\mathbf{z} \sim Z}[\log(1 - \mathcal{D}(\mathcal{G}(\mathbf{z}), \mathbf{z}))]. \tag{6.3}$$

The objective function achieves the global minimum if and only if the distribution of both generator and encoder matches., i.e., $p_{\mathcal{G}}(\mathbf{x}, \mathbf{z}) = p_{\mathcal{E}}(\mathbf{x}, \mathbf{z})$.

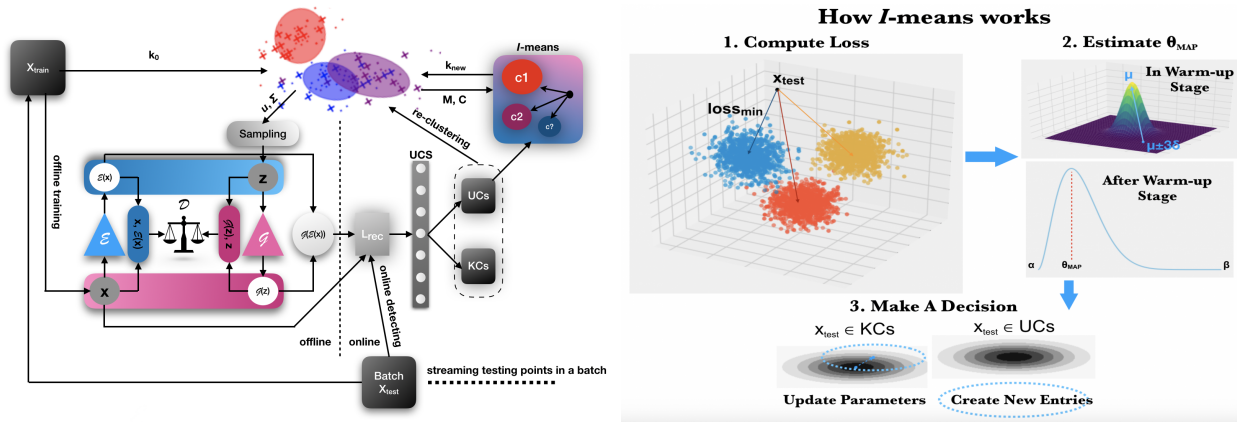


Figure 6.1. The Model Architecture of NE-GM-GAN (Left-hand side) and The Workflow of *I*-means in Algorithm (6) (Right-hand side)

6.3 Methodology

In this chapter, we propose a novel model, Non-Exhaustive Gaussian Mixture Generative Adversarial Networks (NE-GM-GAN) for online non-exhaustive learning. The whole process is displayed in Figure 6.1. Given a training set X_{train} with k_0 KCs, in the training step (offline), the proposed NE-GM-GAN employs a bidirectional GAN to train its encoder $\mathcal{E}$ and generator $\mathcal{G}$, by matching the joint distribution of encoder (X, Z) with the same of the generator. Note that the prior distribution Z of $\mathcal{G}$ is a multi-modal Gaussian (shown as Gaussian clusters on the top-middle part of the figure). After training, the generator and encoder of the GAN can take $\mathbf{z}$ and $\mathbf{x}$ as input and generate $\mathcal{G}(\mathbf{z})$ and $\mathcal{E}(\mathbf{x})$ as output, respectively.

The test step (online) is shown on the right side of the model architecture and it is run on a batch of input instances, X_{test}. For all data instances from a batch (say, $\mathbf{x}$ is one such instance), NE-GM-GAN computes the $UCS(\mathbf{x})$ (unknown class score) of all instances in that batch; UCS score is derived from the reconstruction loss $L_{rec} = |\mathbf{x} - \mathcal{G}(\mathcal{E}(\mathbf{x}))|$. Using this score, the instances of a batch are partitioned into two groups: KCs and UCs. Elements in KCs belong to the known class, whereas the elements in UCs are potential UC instances. Using instances of the UCs group, the model estimates the number of emerging classes, k_{new}.

After estimation, the model updates the prior of the $\mathcal{G}$ by adding the number of new classes k_{new} to k_0 as shown in the top right part of the model architecture. The GMM is then retrained for clustering both KCs and UCs. At this stage, the online test process for one test batch is finished.

In the subsequent discussion, $X_{train} \in \mathbb{R}^{r \times d}$ is considered to be training data, containing r data instances, each of which is represented as a d-dimensional vector. K is the total number of known classes in X_{train}. X_{test} is test data that may contain instances of KCs and also instances of UCs. The dimensionality of latent space is denoted by p.

6.3.1 Offline Training: Computing Multi-modal Prior Distribution

In the vanilla form, generators of both GAN and BiGAN have a unimodal distribution as prior; in other words, the random variables $p_Z(\mathbf{z})$ is an instance from a unimodal distribution. Enlightened by [168], in this chapter, we consider a multi-modal distribution as prior since this prior can better fit the real-life distribution of multi-class datasets. Thus,

$$p_Z(\mathbf{z}) = \sum_{k=1}^{K} \alpha^{\{k\}} \cdot p_k(\mathbf{z}). \tag{6.4}$$

We assume that the number of initial clusters in the Gaussian distribution matches with the number of known classes (K) in X_{train}. $\alpha^{\{k\}}$ is the mixing parameter, $p_k(\mathbf{z})$ denotes the multivariate Normal distribution $\mathcal{N}(u^{\{k\}}, \Sigma^{\{k\}})$, where $u^{\{k\}}$ and $\Sigma^{\{k\}}$ are mean vector and co-variance matrix, respectively.

The model assumes that the number of instances and the number of known classes in the training set are given at the beginning. During training (offline), the parameters $u^{\{k\}}$ and $\Sigma^{\{k\}}$ of each Gaussian cluster are learned by GMM and they are used as the sampling distribution of the latent variable for generating the adversarial instances. As suggested by [168], we also use the re-parameterization trick in this chapter. Instead of sampling the latent variable $\mathbf{z} \sim N(u^{\{k\}}, \Sigma^{\{k\}})$, the model samples $\mathbf{z} = A^{\{k\}}\epsilon + u^{\{k\}}$, where $\epsilon \sim N(0, I)$, $A \in \mathbb{R}^{p \times p}$, $u^{\{k\}} \in \mathbb{R}^p$. In this scenario, $u(\mathbf{z}) = u^{\{k\}}$ and $\Sigma(\mathbf{z}) = A^{\{k\}}A^{\{k\}T}$.

Similar to [53], the GM-GAN (Gaussian Mixture-GAN) learning proceeds as follows. The model takes sampled instance $\mathbf{z}$, sampled from the Gaussian multi-modal prior, and real instances $\mathbf{x}$ as input. Generator $\mathcal{G}$ attempts to map this sampled $p_Z(\mathbf{z})$ to data space as $\mathcal{G}(\mathbf{z})$. Encoder $\mathcal{E}$ maps real instances $\mathbf{x}$ into latent feature space as $\mathcal{E}(\mathbf{x})$. Discriminator $\mathcal{D}$ takes both $p_Z(\mathbf{z})$ and $\mathbf{x}$ as input for matching their joint distributions. After the model converges, theoretically, $\mathcal{G}(\mathbf{z}) \sim \mathbf{x}$ and $\mathcal{E}(\mathbf{x}) \sim p_Z(\mathbf{z})$. Note that NE-GM-GAN encodes X_{train} for offline training. To do so, GMM takes encoded X_{train} as input and then generates encoded u and Σ.

Algorithm 5 *UCS* for multi-modal prior

Input: Matrix $X_{train} \in \mathbb{R}^{r \times d}$ and $X_{test} \in \mathbb{R}^{b \times d}$

Compute $L_{test}(x_{test})$ with Equation (6.5);
for i $\leftarrow 1$ **to** b **do**
  for $k \leftarrow 1$ **to** K **do**
    Compute $L_{train}(x_{train})^{\{k\}}$ with Equation (6.5);
    Select the median of $L_{train}(x_{train})^{\{k\}}$;
    $UCS(x_{test})^{\{k\}} = |L_{test}(x_{test})^{\{\mathrm{i}\}} - L_{train}(x_{train})^{\{k\}}_{median}|$;
  end for
  $UCS^{\{\mathrm{i}\}}_{min} = \min\left(UCS(x_{test})^{\{1\}}, ..., UCS(x_{test})^{\{K\}}\right)$;
end for
$UCS = [UCS^{\{1\}}_{min}, ..., UCS^{\{b\}}_{min}]$;
return Vector $UCS \in \mathbb{R}^{b \times 1}$.

6.3.2 Extracting Potential Unknown Class

UC extraction of NE-GM-GAN is an online process that works on unlabeled data. During online detection, the model assumes that the test instance $\mathbf{x}$ is coming in a batch of the test set $X_{test} \in \mathbb{R}^{b \times d}$, where b is batch size and d is the dimension of feature space. Unlike [53], whose purpose is to generate fake images as real as possible, our model aims at extracting the UC as accurately as possible. More specifically, our model generates the reconstructed instance $\mathcal{G}(\mathcal{E}(\mathbf{x}))$ at first and then computes the reconstruction loss between $\mathbf{x}$ and $\mathcal{G}(\mathcal{E}(\mathbf{x}))$.

This step returns a size-b 1-D vector, consisting of reconstruction losses of the b points in the current batch, which is defined below:

$$L_{rec} = \|\mathbf{x} - \mathcal{G}(\mathcal{E}(\mathbf{x}))\|. \tag{6.5}$$

To distinguish the UC from KC in each test batch, we propose a metric, unknown class score, in short, UCS; the larger the score for an instance, the more likely that the instance belongs to an unknown class. To compute UCS of a test instance $\mathbf{x}$, NE-GM-GAN first computes, for each KC (out of K KCs), a baseline reconstruction loss, which is equal to the median of reconstruction losses of all train objects belonging to that known class. Then, UCS of $\mathbf{x}$ is equal to the minimum of the differences between $\mathbf{x}$'s reconstruction loss and each of the K baseline reconstruction losses. The pseudo-code of UCS computation is shown in Algorithm 5.

The intuition of the UCS function is that GAN models instances of KCs with smaller reconstruction loss than the instances of UCs, but different known classes may have different baseline reconstruction loss, so we want an unknown class's reconstruction loss larger than the worst loss among all the KCs. This mechanism is inspired by [169]. Nevertheless, unlike [169], which assumes the prior as unimodal distribution and the UC must be far away from KC, our approach considers a multi-modal prior. After computing the UCS, the model extracts the potential UC from KC with a given threshold. For online detection, the threshold for the first test batch is empirically given whereas subsequent thresholds are decided by the percentage of UCs from previous test batches. Note that, the UCs objects may belong to multiple classes, but the model has no knowledge yet about the number of classes.

6.3.3 Estimating The Number of Emerging Class

The previous extraction only extracts potential UCs. In practice, a small number of anomalous KC instances may be selected as UC instances. So, we use a subsequent step that distinctly identifies instances of unknown classes together with the number of UC and their parameters (mean, and covariance matrix of each of the UCs). We name this step as

Infinite Means (I-means); the name reflects the fact that the number of unknown classes can increase as large as needed based on the test instances. Using I-means, a test instance is assigned to a new class if it is positioned far from the mean of all the KCs, and discovered novel classes prior to seeing that instance. To achieve this, for i-th test instance $x_{test}^{\{i\}}$, as shown in Equation (6.6), I-means computes the distance $L_{\mu}^{\{k\}}$ between $x_{test}^{\{i\}}$ and the mean vector $\mu^{\{k\}}$ for the k-th KC and then selects the minimum of these values as $loss_{min}$ in Equation (6.7).

$$L_{\mu}^{\{k\}} = \|x_{test}^{\{i\}} - \mu^{\{k\}}\|, \forall k \in [1..K]. \tag{6.6}$$

$$loss_{min} = \min\left(L_{\mu}^{\{1\}}, L_{\mu}^{\{2\}}, ..., L_{\mu}^{\{K\}}\right), \ \mathrm{i}dx = \arg\min\left(L_{\mu}^{\{1\}}, L_{\mu}^{\{2\}}, ..., L_{\mu}^{\{K\}}\right). \tag{6.7}$$

A small value of $loss_{min}$ indicates that $x_{test}^{\{i\}}$ may potentially be a member of class idx; on the other hand, a large value $loss_{min}$ indicates that $x_{test}^{\{i\}}$ possibly belongs to a UC. To make the determination, we use a Bayesian approach, which dynamically adjusts the probability that a test point that is closest to cluster idx's mean vector belongs to cluster idx or not. The process is described below.

For a test instance, $x_{test}^{\{i\}}$ for which i$dx = k$, the binary decision whether the instance belongs to k-th existing cluster or an emerging cluster follows Bernoulli distribution with parameter θ_k, which is modeled by using a Beta prior with parameter α_k, and β_k, where $\alpha_k, \beta_k \geq 1$ and $\theta_k = \frac{\alpha_k}{\alpha_k+\beta_k}$. The value of α_k and β_k are updated using the Bayes rule. Based on the Bayes' theorem, the posterior distribution $p(\theta_k|x_{test}^{\{i\}})$, where $\theta_k \in [0, 1]$, is proportional to the prior distribution $p(\theta_k)$ multiplied by the likelihood function $p(x_{test}^{\{i\}}|\theta_k)$:

$$p(\theta_k|x_{test}^{\{i\}}) \propto p(x_{test}^{\{i\}}|\theta_k) \cdot p(\theta_k). \tag{6.8}$$

The posterior $p(\theta_k|x_{test}^{\{i\}})$ in Equation (6.8) can be re-written as following:

$$\begin{aligned} p(\theta_k|x_{test}^{\{i\}}) &\propto \theta_k^{\alpha_{k0}}(1-\theta_k)^{\beta_{k0}} \cdot \theta_k^{\alpha_k-1}(1-\theta_k)^{\beta_k-1} \\ &= \theta_k^{\alpha_{k0}+\alpha_k-1} \cdot (1-\theta_k)^{\beta_{k0}+\beta_k-1} \\ &= beta(\theta_k|\alpha_{k0}+\alpha_k, \beta_{k0}+\beta_k). \end{aligned} \tag{6.9}$$

As the test instances are coming in streaming fashion, for any subsequent test instance for which $idx = k$, the posterior $p(\theta_k|x_{test}^{\{i\}})$ will act as prior for the next update. For the very first iteration, α_{k0} and β_{k0} are shape parameters of beta prior, which we learn in a warm-up stage. In the warm-up stage, we apply the three-sigma rule to compute the beta priors, α_{k0}, and β_{k0}. Each test point in the warm-up stage, for which $idx = k$, contributes a count of 1 to α_{k0} if the point is further than 3 standard deviation away from the mean, otherwise it contributes a count of 1 to β_{k0}. After the warm-up stage, we employ the Maximum-A-Posteriori (MAP) estimation to obtain the θ_{MAP_k} at which the posterior $p(\theta_k|x_{test}^{\{i\}})$ reaches its maximum value. According to the property of beta distribution, the θ_{MAP_k} is most likely to occur at the mean of posterior $p(\theta_k|x_{test}^{\{i\}})$. Thus, we can estimate the θ_{MAP_k} by:

$$\theta_{MAP_k} = \arg\max_{\theta_k} p(\theta_k|x_{test}^{\{i\}}) = \frac{\alpha_{k0} + \alpha_k}{\alpha_{k0} + \alpha_k + \beta_{k0} + \beta_k}. \tag{6.10}$$

After estimating the θ_{MAP_k} by Equation (6.10), *I*-means makes a cluster membership decision for each $x_{test}^{\{i\}}$ based on θ_{MAP_k}. This decision simulates the Bernoulli process, i.e., among the test instances which are close to the k-th cluster, approximately θ_{MAP_k} fraction of those will belong to the emerging cluster, whereas the remaining $(1-\theta_{MAP_k})$ fractions of such instances will belong to the k-th cluster. After each decision, the corresponding parameters will be updated. If $x_{test}^{\{i\}}$ is clustered as a member of $KC^{\{k\}}$, we update the parameters $\mu_k^{\{i\}} \in \mathbb{R}^{1\times d}$, $\sigma_k^{\{i\}} \in \mathbb{R}^{d\times d}$ of the k-th cluster by Equation (6.11) and Equation (6.12), respectively. The shape parameter β_k is increased by 1. Otherwise, if $x_{test}^{\{i\}}$ is considered as a member of UC, the shape parameter α_k, k_{new} are increased by 1, and the mean and covariance matrix of this new class are initialized by assigning current $x_{test}^{\{i\}}$ as new mean vector and creating a zero vector with the same shape of $x_{test}^{\{i\}}$ as new standard deviation vector.

$$\mu_k^{\{i\}} = \mu_k^{\{i-1\}} + \frac{x_{test}^{\{i\}} - \mu_k^{\{i-1\}}}{i}. \tag{6.11}$$

$$v_k^{\{i\}} = v_k^{\{i-1\}} + \left(x_{test}^{\{i\}} - \mu_k^{\{i-1\}}\right)\left(x_{test}^{\{i\}} - \mu_k^{\{i\}}\right),\ \sigma_k^{\{i\}} = \sqrt{\frac{v_k^{\{i\}}}{(i-1)}}. \tag{6.12}$$

The entire process of this paragraph is summarized *I*-means in Algorithm 6.

Algorithm 6 INFINITE Means (I-means)

Input: Testing batch $X_{test} \in \mathbb{R}^{b \times d}$, mean and co-variance matrices.

for all $x^{\{i\}} \in X_{test}$ **do**
  for all $\mu^{\{k\}} \in M$ **do**
    Compute $L_{\mu}^{\{k\}}$ by Equation (6.6);
  end for
  Get the index, idx, of minimum loss by Equation (6.7);
  if warm-up stage **then**
    Select beta prior α_{idx0} and β_{idx0} based on Three-sigma Rule;
  else
    Estimate the θ_{MAP_k} by Equation (6.10);
  end if
  if Uniform $(0, 1) \leq \theta_{MAP_k}$ **then**
    Update corresponding μ and σ by Equation (6.11) and Equation (6.12);
    $\beta_{idx} \leftarrow \beta_{idx} + 1$;
  else
    $\alpha_{idx} \leftarrow \alpha_{idx} + 1$;
    $k_{new} \leftarrow k_{new} + 1$;
  end if
end for
return The number of new emerging clusters k_{new}.

6.4 Experiments

In this section, we show experimental results for validating the superior performance of our proposed NE-GM-GAN over different competing methods for multiple capabilities. Firstly, we compare the performance of potential UCs extraction. Furthermore, we compare the estimation of the number of distinct unknown classes. Finally, we show some experimental results for studying the effect of user-defined parameters on the algorithm's performance.

6.4.1 Dataset

We evaluate NE-GM-GAN on four datasets. Three of the datasets are real-life network intrusion datasets and the remaining one is a synthetic dataset. Network intrusion is very common for non-exhaustive classification because attackers constantly update their attack methods, so the classification model must adapt to novel class scenarios. The datasets are:

- KDD Cup 1999 network intrusion dataset **(KDD99)**, which contains 494,021 instances and 41 features with 23 different classes. One of the classes represents Normal activity and the rest 22 represent various network attacks;
- NSL-KDD dataset **(NSL-KDD)** [170], which is also a network intrusion dataset built by filtering some records from KDD99;
- UNSW-NB15 dataset **(UNSW-NB15)** [171], which hybridizes real normal network activities with synthetic attack;
- Synthetic dataset **(Synthetic)**, which contains non-isotropic Gaussian clusters.

Many of the features in the intrusion datasets are categorical or binary, so we employ one-hot embedding for such features. We also drop some columns which are redundant or whose values are almost zero or missing along the column. After that, we select eight of the classes as unknown classes (UCs) for each dataset.

The test set is constructed from two parts. The first part is randomly-sampled 20% of KCs instances and the second part is all the instances of the UCs. The rest 80% of KC instances are left for the training set. In the synthetic dataset, noises are injected into Gaussian clusters, each cluster representing a class. The injected noise is homocentric to the corresponding normal class but with a larger variance. The detailed statistics of the datasets are provided in Table 6.2.

6.4.2 Competing Methods

The performance of UCs extraction is evaluated with three competing methods, AnoGAN [155], DAGMM [156], and ALAD [169]. AnoGAN is the first GAN-based model for UC detection. Similarly, ALAD is another GAN-based model, which uses reconstructed errors to determine the UC. In contrast, DAGMM implements the autoencoder for the same task instead. The experimental setting follows [169] for this experiment. On the other hand, the capability of estimating the number of new emerging classes is compared against two competing methods, X-means [172], and IGMM [76], [77]. X-means is a classical distance-based algorithm that can efficiently search the data space without knowing the initial number

Table 6.2. Statistics of Datasets (#Inst. denotes the number of instances; #F. denotes the number of features after one-hot embedding or dropping for network intrusion dataset; #C. denotes the number of classes.)

Dataset	#Inst.	#F.	#C.	Selected UCs
KDD99	494,021	121	23	neptune, normal, back, satan, ipsweep, portsweep, warezclient, teardrop
NSL-KDD	148,517	121	40	neptune, satan, ipsweep, smurf, portsweep, nmap, back, guess_passwd
UNSW-NB15	175,341	169	10	generic, exploits, fuzzers, DoS, reconnaissance, analysis, backdoor, shellcode
Synthetic	100,300	121	16	No. 3, 4, 5, 6, 7, 8, 9, 10

of clusters. On the contrary, IGMM is a Bayesian mixture model which uses the Dirichlet process prior and the Gibbs sampler to efficiently identify new emerging entities. This experiment uses one sweep Gibbs sampler for IGMM [77]. For IGMM, we select the tunable parameters as follows; $h = 10$, $m = h + 100$, $\kappa = 100$ and $\alpha = 100$, which is identical to the parameter values in [77]. Both models can return the number of online classes as NE-GM-GAN does, so they are selected as competing methods.

6.4.3 Evaluation Metrics

We use an external clustering evaluation metric, such as F1-score, to evaluate the performance of UCs extraction. For evaluating the prediction of the number of UCs (a regression task), we propose a new metric, Symmetrical R-squared (S-R^2). To obtain this, the root mean square error ($RMSE$) for both NE-GM-GAN and a competing method are computed and plugged into Equation 6.13. S-$R^2 \in$ [-1, 1] gets more close to 1 if NE-GM-GAN defeats the competing method. On the contrary, its value will become more close -1. S-R^2 is exactly equal to 1 when the proposed model gets perfect prediction while the competing method doesn't. S-R^2 is zero when both methods have similar performance. The motivation to propose a new metric rather than using R-squared (R^2) is that R^2 would be less distinctive if

two methods get much worse predictions because of using mean square error (MSE) inside. Besides, baseline sometimes achieves better performance, but R^2 cannot reflect this scenario as its range is from negative infinity to positive one.

$$S\text{-}R^2 = \begin{cases} 1 - \dfrac{RMSE_m}{RMSE_{bl}}, & RMSE_m < RMSE_{bl} \\ \dfrac{RMSE_{bl}}{RMSE_m} - 1, & RMSE_m > RMSE_{bl} \end{cases}, \tag{6.13}$$

where $RMSE_m$ and $RMSE_{bl}$ denote the $RMSE$ of our model and baseline model, respectively.

Table 6.3. The F1-score of Four Models for UCs Extraction

Data	NE-GM-GAN	AnoGAN	DAGMM	ALAD
KDD99	**0.99**	0.87	0.97	0.94
NSL-KDD	0.75	0.68	**0.79**	0.73
UNSW-NB15	**0.57**	0.49	0.53	0.51
Synthetic	**0.74**	0.51	0.70	0.56

6.4.4 The Capability of Unknown Class Extraction

In Table 6.3, we show the F1-score values of NE-GM-GAN and the competing methods for detecting the unknown class instances (the best results are shown in bold font). The result is computed by running each model 10 times and then taking the average. Out of the four datasets, NE-GM-GAN has the best performance in three with a healthy margin over the second-best method. In the largest dataset, our model received a 0.99 F1-score, a very good performance considering the fact that unknown class instances are assembled from 8 different classes. Only in the NSL-KDD dataset, NE-GM-GAN came out as the second-best. The performance of the other three models is mixed without a clear winner. One observation is that all the methods perform better on the larger dataset (KDD99).

To understand NE-GM-GAN's performance further, we perform an ablation study by switching the prior, as shown in Table 6.4. As we can see, the Gaussian multi-modal prior used in NE-GM-GAN is better suited than the Unimodal prior generally used in traditional

Table 6.4. F1 Score from Our Proposed Model by Using Different Prior

Prior	KDD99	NSL-KDD	UNSW-NB15	Synthetic
Unimodal	0.98	0.74	0.55	0.72
Multi-modal	**0.99**	**0.75**	**0.57**	**0.74**

GAN. For all datasets multi-modal prior has 1% to 2% better F-score. A possible reason is that the multi-modal prior is closer to the real distribution of the training data.

Table 6.5. The S-R^2 between NE-GM-GAN and Baselines on Four Datasets (We denote "UCs" as the number of unknown classes in this table.)

Datasets	Methods	UCs=2	UCs=3	UCs=4	UCs=5	UCs=6
KDD99	X-means	0.8301	0.8805	0.8628	0.9105	0.8812
	IGMM	0.9528	0.8991	0.8908	0.9303	0.9248
NSL-KDD	X-means	0.8892	0.8604	0.9539	0.9228	0.9184
	IGMM	0.8771	0.8647	0.9517	0.9285	0.9238
UNSW-NB15	X-means	0.8892	0.8604	0.9539	0.9228	0.9184
	IGMM	0.8771	0.8647	0.9517	0.9285	0.9238
Synthetic	X-means	0	0	0	0	0
	IGMM	1	1	1	1	1

6.4.5 The Estimation of The Number of New Classes

In this experiment, we compare NE-GM-GAN against two competing methods on all four datasets. To extend the scope of experiments, we vary the number of unknown classes from 2 to 6 by choosing all possible combinations of UCs and build multiple copies of one dataset. We report performance results over all those copies. The motivation for using a combination of different UCs is to validate the robustness of the methods against varying numbers of UC counts. The result is shown in Table 6.5 using S-R^2 metric discussed earlier. The result close to 1 (the majority of the values in the table are between 0.85 and 0.95) means NE-GM-GAN substantially outperforms the competing methods. We argue that both competing methods assume that data distribution in each class follows a mixture of Gaussian and thus fails to achieve good performance on realistic datasets. In only one dataset (Synthetic), X-means

was able to obtain identical performance as ours' method, as both methods have the perfect prediction. The same results are also shown in Figure 6.2 as bar charts. In this Figure, the y-axis is the number of predicted clusters, and each group of bars denotes the number of actual clusters for different methods. As we can see, NE-GM-GAN's prediction is very close to the actual prediction, whereas the results of the completing methods are way-off, except for the X-means method on the synthetic dataset. These experimental results demonstrate that our NE-GM-GAN outperforms the competing methods in terms of accuracy and robustness.

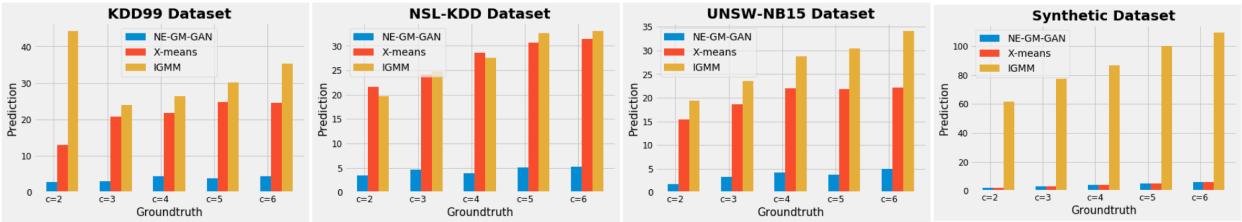


Figure 6.2. Comparison on The Estimation of New Emerging Class among Three Methods

Table 6.6. Test of Three-sigma Rule (%)

Range	KDD99	NSL-KDD	UNSW-NB15	Synthetic
$\mu \pm 1\sigma$	94.37	61.17	58.67	56.64
$\mu \pm 2\sigma$	99.58	99.87	99.92	99.81
$\mu \pm 3\sigma$	99.60	100.00	100.00	100.00

6.4.6 Study of User-defined Parameters

We perform a few experiments to justify some of our parameter design choices. For instance, to build the initial beta priors we used the three-sigma rule. In Table 6.6, we present the percentage of instances of points that falls within the three standard deviations of the mean. The four columns correspond to the four datasets. As can be seen in the third row of the table, for all datasets, almost 100% of the points fall within the three standard deviations away from the mean. So, the priors selected in the warm-up stage based on the three-sigma rule can sufficiently distinguish the UCs from the known class instances.

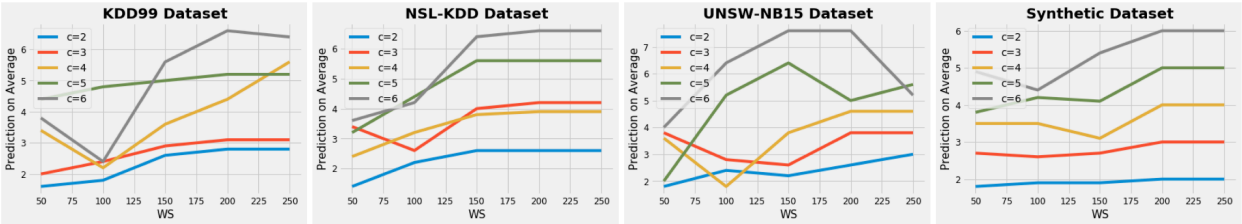


Figure 6.3. Investigation on The Number of Epochs in The Warm-up Stage (WS) for I-means on Four Datasets

We also show unknown class prediction results over different values of WS (epochs of the warm-up stage) for different (between 2 to 6) unknown class that counts for all datasets. In Figure 6.3, each curve represents a specific UC count. As can be seen, the prediction of the unknown class gets better with a larger number of WS. In most cases, the prediction converges when the number of epochs in the warm-up stage (WS) reaches 200 or above. In all our experiments, we select the WS value 200 for all datasets.

6.4.7 Reproducibility of The Work

The model is implemented using Python 3.6.9 and Keras 2.2.4. For optimization, Adam is used with $\alpha = 10^{-5}$ and $\beta = 0.5$; mini-batch size is 50, latent dimension is 32, and the number of training epochs equal 1000. The source code is available at https://github.com/junzhuang-code/NEGMGAN. The details of the BiGAN model architecture are given in Table 6.7.

Table 6.7. Model Architectures

	Layers	Units	Activation	Batch Norm.	Dropout
$\mathcal{E}(\mathbf{x})$	Dense	64	LReLU(0.2)	×	0.0
	Dense	1	None	×	0.0
$\mathcal{G}(\mathbf{z})$	Dense	64	LReLU(0.2)	×	0.0
	Dense	128	LReLU(0.2)	×	0.0
	Dense	121	Tanh	×	0.0
$\mathcal{D}(\mathbf{x}, \mathbf{z})$	Dense	128	LReLU(0.2)	✓	0.5
	Dense	128	LReLU(0.2)	✓	0.5
	Dense	1	Sigmoid	×	0.0

6.5 Chapter Summary

In this chapter, we present a new online non-exhaustive model, Non-Exhaustive Gaussian Mixture Generative Adversarial Network (NE-GM-GAN), that synthesizes the Bayesian method and deep learning technique for incremental learning the new emerging classes. NE-GM-GAN consists of three main components: (1) Gaussian mixture clustering generating multi-modal prior and re-clusters both KCs and UCs for parameter updating. (2) Bidirectional adversarial learning reconstructs the loss for extracting imbalanced UCs from KCs in an online testing batch. (3) A novel algorithm, *I*-means, estimates the number of new emerging classes for incremental learning the UCs on large sparse datasets. Experimental results illustrate that NE-GM-GAN significantly outperforms the competing methods for online detection across several benchmark datasets.

7. SUMMARY

In this thesis, we aim to address the robustness issues of Artificial Neural Networks (ANNs) from two aspects. First, we propose a series of Bayesian label transition models to improve the robustness of Graph Neural Networks (GNNs) in the presence of label scarcity and perturbations. Specifically, in this aspect, we first present a Bayesian label transition model, GraphLT, to repair the classification accuracy in online social networks under the environment of noisy labeling and random perturbations. We further develop a Bayesian self-supervision model, GraphSS, to recover the classification performance against dynamic graph perturbations, a kind of adversarial attack on dynamic graphs, on label-scarce dynamic graphs. To speed up the convergence of both GraphLT and GraphSS, we propose a new topology-based label sampling method in LInDT. Extensive experiments indicate that our proposed models can successfully recover the performance of node classifications over several public graph datasets. Second, we propose a new online non-exhaustive learning model, named NE-GM-GAN, to detect novel classes in the test data and to defy the class imbalance problem effectively. The experimental results demonstrate that our model is superior to existing methods for accurate and robust online detection of emerging classes in streaming data over four datasets.

Although the above-mentioned models have been empirically ascertained as effective, they still have some limitations. For the Bayesian label transition models, for example, the inference highly depends on the warm-up transition matrix ϕ', which is built with the categorical distribution on the train graph. Such dependence indicates that the model can only handle the evasion perturbations at this point. Leveraging the Gaussian distribution as the prior distribution could be a potential solution to overcome this limitation. Also, such dependence implies that the model cannot handle a class that doesn't exist in the train data. We categorize this issue as a subset of the open-set learning paradigm. In this paradigm, researchers aim to develop models to handle test instances that may belong to multiple unseen classes. Such a paradigm is very common in our daily lives. For example, in online social networks, such as Reddit, new posts may belong to a new community that doesn't exist. However, it's still challenging to classify such novel posts nowadays. Furthermore,

this paradigm targets label space and could be extended to feature space. In other words, the number of feature dimensions could be non-exhaustive. Such an extended paradigm is even more challenging as conventional neural networks cannot handle inputs with infinite feature dimensions. Quantum computing could be a potential solution to this challenging issue. In brief, open-set learning in the label space and non-exhaustive exploration in the feature space could be two promising and challenging paradigms related to the robustness of ANNs.

Besides the above-mentioned two paradigms, our proposed Bayesian label transition models can benefit large ANNs in transfer learning paradigms. In this paradigm, researchers first pre-train the large models for generating the embedding. The generated embedding is then used for various downstream tasks, such as classification, prediction, and retrieval. However, the pre-trained large models may suffer from robust issues. For instance, the performance of the models on downstream tasks may deteriorate during inference when the distribution of downstream datasets shifts. This domain shift issue could be caused by perturbations. In this case, our proposed Bayesian label transition models can be applied to the downstream tasks to recover the classification performance. Overall, our model can benefit the robustness of large ANNs in various domains, such as image, language, and graph. These attempts are left for future research.

REFERENCES

[1] E. Micheli-Tzanakou, "Artificial neural networks: An overview," *Network: Computation in Neural Systems*, vol. 22, no. 1-4, pp. 208–230, 2011.

[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[3] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

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

[5] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[6] I. Goodfellow, J. Pouget-Abadie, M. Mirza, *et al.*, "Generative adversarial networks," *Communications of the ACM*, vol. 63, no. 11, pp. 139–144, 2020.

[7] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *2017 ieee symposium on security and privacy (sp)*, Ieee, 2017, pp. 39–57.

[8] L. Jing and Y. Tian, "Self-supervised visual feature learning with deep neural networks: A survey," *IEEE transactions on pattern analysis and machine intelligence*, vol. 43, no. 11, pp. 4037–4058, 2020.

[9] A. Chakraborty, M. Alam, V. Dey, A. Chattopadhyay, and D. Mukhopadhyay, "Adversarial attacks and defences: A survey," *arXiv preprint arXiv:1810.00069*, 2018.

[10] C. Geng, S.-j. Huang, and S. Chen, "Recent advances in open set recognition: A survey," *IEEE transactions on pattern analysis and machine intelligence*, vol. 43, no. 10, pp. 3614–3631, 2020.

[11] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," *arXiv preprint arXiv:1706.06083*, 2017.

[12] I. Misra, C. Lawrence Zitnick, M. Mitchell, and R. Girshick, "Seeing through the human reporting bias: Visual classifiers from noisy human-centric labels," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 2930–2939.

[13] M. M. Ahsan, S. A. Luna, and Z. Siddique, "Machine-learning-based disease diagnosis: A comprehensive review," in *Healthcare*, MDPI, vol. 10, 2022, p. 541.

[14] J. Zhuang and D. Wang, "Geometrically matched multi-source microscopic image synthesis using bidirectional adversarial networks," in *Proceedings of 2021 International Conference on Medical Imaging and Computer-Aided Diagnosis (MICAD 2021) Medical Imaging and Computer-Aided Diagnosis*, Springer, 2022, pp. 79–88.

[15] C. Gong, T. Ren, M. Ye, and Q. Liu, "Maxup: A simple way to improve generalization of neural network training," *arXiv preprint arXiv:2002.09024*, 2020.

[16] J. Zhuang and M. Al Hasan, "Non-exhaustive learning using gaussian mixture generative adversarial networks," in *Machine Learning and Knowledge Discovery in Databases. Research Track: European Conference, ECML PKDD 2021, Bilbao, Spain, September 13–17, 2021, Proceedings, Part II 21*, Springer, 2021, pp. 3–18.

[17] E. Dai, C. Aggarwal, and S. Wang, "Nrgnn: Learning a label noise-resistant graph neural network on sparsely and noisily labeled graphs," *arXiv preprint arXiv:2106.04714*, 2021.

[18] L. Galke, I. Vagliano, and A. Scherp, "Can graph neural networks go" online"? an analysis of pretraining and inference," *arXiv preprint arXiv:1905.06018*, 2019.

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

[20] H. Jin and X. Zhang, "Latent adversarial training of graph convolution networks," in *ICML Workshop on Learning and Reasoning with Graph-Structured Representations*, 2019.

[21] Z. Deng, Y. Dong, and J. Zhu, "Batch virtual adversarial training for graph convolutional networks," *arXiv preprint arXiv:1902.09192*, 2019.

[22] F. Feng, X. He, J. Tang, and T.-S. Chua, "Graph adversarial training: Dynamically regularizing based on graph structure," *IEEE Transactions on Knowledge and Data Engineering*, 2019.

[23] Y. Rong, W. Huang, T. Xu, and J. Huang, "Dropedge: Towards deep graph convolutional networks on node classification," *arXiv preprint arXiv:1907.10903*, 2019.

[24] D. Luo, W. Cheng, W. Yu, *et al.*, "Learning to drop: Robust graph neural network via topological denoising," in *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*, 2021, pp. 779–787.

[25] X. Zhang and M. Zitnik, "Gnnguard: Defending graph neural networks against adversarial attacks," *arXiv preprint arXiv:2006.08149*, 2020.

[26] S. Geisler, D. Zügner, and S. Günnemann, "Reliable graph neural networks via robust aggregation," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[27] L. Chen, J. Li, Q. Peng, Y. Liu, Z. Zheng, and C. Yang, "Understanding structural vulnerability in graph convolutional networks," *arXiv preprint arXiv:2108.06280*, 2021.

[28] X. Liu, W. Jin, Y. Ma, *et al.*, "Elastic graph neural networks," in *International Conference on Machine Learning*, PMLR, 2021, pp. 6837–6849.

[29] W. Jin, T. Derr, Y. Wang, Y. Ma, Z. Liu, and J. Tang, "Node similarity preserving graph convolutional networks," in *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*, 2021, pp. 148–156.

[30] Y. Zhang, S. Khan, and M. Coates, "Comparing and detecting adversarial attacks for graph deep learning," in *Proc. Representation Learning on Graphs and Manifolds Workshop, Int. Conf. Learning Representations, New Orleans, LA, USA*, 2019.

[31] H. Wu, C. Wang, Y. Tyshetskiy, A. Docherty, K. Lu, and L. Zhu, "Adversarial examples on graph data: Deep insights into attack and defense," *arXiv preprint arXiv:1903.01610*, 2019.

[32] C. Zheng, B. Zong, W. Cheng, *et al.*, "Robust graph representation learning via neural sparsification," 2019.

[33] J. Zhuang and M. Al Hasan, "Deperturbation of online social networks via bayesian label transition," in *Proceedings of the 2022 SIAM International Conference on Data Mining (SDM)*, SIAM, 2022, pp. 603–611.

[34] J. Zhuang and M. Al Hasan, "Defending graph convolutional networks against dynamic graph perturbations via bayesian self-supervision," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, no. 4, pp. 4405–4413, Jun. 2022. DOI: 10.1609/aaai.v36i4.20362. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/20362.

[35] J. Zhuang and M. Al Hasan, "Robust node classification on graphs: Jointly from bayesian label transition and topology-based label propagation," in *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, 2022, pp. 2795–2805.

[36] W. Köhler, "Gestalt psychology today.," *American psychologist*, vol. 14, no. 12, p. 727, 1959.

[37] T. M. Mitchell *et al.*, *Machine learning*. McGraw-hill New York, 2007, vol. 1.

[38] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th international conference on machine learning (ICML-10)*, 2010, pp. 807–814.

[39] A. L. Maas, A. Y. Hannun, A. Y. Ng, *et al.*, "Rectifier nonlinearities improve neural network acoustic models," in *Proc. icml*, Atlanta, Georgia, USA, vol. 30, 2013, p. 3.

[40] D. H. Ackley, G. E. Hinton, and T. J. Sejnowski, "A learning algorithm for boltzmann machines," *Cognitive science*, vol. 9, no. 1, pp. 147–169, 1985.

[41] G. E. Hinton, T. J. Sejnowski, *et al.*, "Learning and relearning in boltzmann machines," *Parallel distributed processing: Explorations in the microstructure of cognition*, vol. 1, no. 282-317, p. 2, 1986.

[42] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[43] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

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

[45] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and locally connected networks on graphs," *arXiv preprint arXiv:1312.6203*, 2013.

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

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

[48] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in neural information processing systems*, 2017, pp. 1024–1034.

[49] P. Velikovi, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *International Conference on Learning Representations*, 2018.

[50] J. Du, S. Zhang, G. Wu, J. M. Moura, and S. Kar, "Topology adaptive graph convolutional networks," *arXiv preprint arXiv:1710.10370*, 2017.

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

[52] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein generative adversarial networks," in *International conference on machine learning*, PMLR, 2017, pp. 214–223.

[53] J. Donahue, P. Krähenbühl, and T. Darrell, "Adversarial feature learning," *arXiv preprint arXiv:1605.09782*, 2016.

[54] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," *arXiv preprint arXiv:1511.06434*, 2015.

[55] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training gans," *Advances in neural information processing systems*, vol. 29, 2016.

[56] H. Zhang, T. Xu, H. Li, *et al.*, "Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 5907–5915.

[57] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2223–2232.

[58] T. Karras, S. Laine, and T. Aila, "A style-based generator architecture for generative adversarial networks," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 4401–4410.

[59] A. Brock, J. Donahue, and K. Simonyan, "Large scale gan training for high fidelity natural image synthesis," *arXiv preprint arXiv:1809.11096*, 2018.

[60] S. Reed, H. Lee, D. Anguelov, C. Szegedy, D. Erhan, and A. Rabinovich, "Training deep neural networks on noisy labels with bootstrapping," *arXiv preprint arXiv: 1412.6596*, 2014.

[61] J. Goldberger and E. Ben-Reuven, "Training deep neural-networks using a noise adaptation layer," *International Conference on Learning Representations*, 2017.

[62] G. Patrini, A. Rozza, A. Krishna Menon, R. Nock, and L. Qu, "Making deep neural networks robust to label noise: A loss correction approach," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 1944–1952.

[63] S. Sukhbaatar, J. Bruna, M. Paluri, L. Bourdev, and R. Fergus, "Training convolutional networks with noisy labels," *arXiv preprint arXiv:1406.2080*, 2014.

[64] J. Yao, H. Wu, Y. Zhang, I. W. Tsang, and J. Sun, "Safeguarded dynamic label regression for noisy supervision," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 9103–9110.

[65] H. NT, C. J. Jin, and T. Murata, "Learning graph neural networks with noisy labels," *arXiv preprint arXiv:1905.01591*, 2019.

[66] J.-X. Zhong, N. Li, W. Kong, S. Liu, T. H. Li, and G. Li, "Graph convolutional label noise cleaner: Train a plug-and-play action classifier for anomaly detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 1237–1246.

[67] W. J. Scheirer, A. de Rezende Rocha, A. Sapkota, and T. E. Boult, "Toward open set recognition," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 7, pp. 1757–1772, 2012.

[68] C. Geng, S.-j. Huang, and S. Chen, "Recent advances in open set recognition: A survey," *arXiv preprint arXiv:1811.08581*, 2018.

[69] W. J. Scheirer, L. P. Jain, and T. E. Boult, "Probability models for open set recognition," *IEEE transactions on pattern analysis and machine intelligence*, vol. 36, no. 11, pp. 2317–2324, 2014.

[70] A. Bendale and T. Boult, "Towards open world recognition," *arXiv preprint arXiv: 1412.5687*, 2014.

[71] A. Bendale and T. Boult, "Towards open set deep networks," *arXiv preprint arXiv: 1511.06233*, 2015.

[72] Z. Ge, S. Demyanov, Z. Chen, and R. Garnavi, "Generative openmax for multi-class open set classification," *arXiv preprint arXiv:1707.07418*, 2017.

[73] L. Neal, M. Olson, X. Fern, W.-K. Wong, and F. Li, "Open set learning with counterfactual images," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 613–628.

[74] I. Jo, J. Kim, H. Kang, Y.-D. Kim, and S. Choi, "Open set recognition by regularising classifier with fake data generated by generative adversarial networks," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, 2018, pp. 2686–2690.

[75] Y. Yang, C. Hou, Y. Lang, D. Guan, D. Huang, and J. Xu, "Open-set human activity recognition based on micro-doppler signatures," *Pattern Recognition*, vol. 85, pp. 60–69, 2019.

[76] C. E. Rasmussen, "The infinite gaussian mixture model," in *NIPS*, 2000, pp. 554–560.

[77] B. Zhang, M. Dundar, and M. A. Hasan, "Bayesian non-exhaustive classification a case study: Online name disambiguation using temporal record streams," *arXiv preprint arXiv:1607.05746*, 2016.

[78] X. Wang, J. Eaton, C.-J. Hsieh, and F. Wu, "Attack graph convolutional networks by adding fake nodes," *arXiv preprint arXiv:1810.10751*, 2018.

[79] D. Zügner, A. Akbarnejad, and S. Günnemann, "Adversarial attacks on neural networks for graph data," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018, pp. 2847–2856.

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

[81] S. Wang, Z. Chen, J. Ni, *et al.*, "Adversarial defense framework for graph neural network," *arXiv preprint arXiv:1905.03679*, 2019.

[82] A. Zhang and J. Ma, "Defensevgae:defending against adversarial attacks on graph data via a variational graph autoencoder," *arXiv preprint arXiv:2006.08900*, 2020.

[83] K. Xu, H. Chen, S. Liu, *et al.*, "Topology attack and defense for graph neural networks: An optimization perspective," *arXiv preprint arXiv:1906.04214*, 2019.

[84] N. Entezari, S. A. Al-Sayouri, A. Darvishzadeh, and E. E. Papalexakis, "All you need is low (rank) defending against adversarial attacks on graphs," in *Proceedings of the 13th International Conference on Web Search and Data Mining*, 2020, pp. 169–177.

[85] P. Elinas, E. V. Bonilla, and L. Tiao, "Variational inference for graph convolutional networks in the absence of graph data and adversarial settings," *arXiv preprint arXiv:1906.01852*, 2019.

[86] A. Breuer, R. Eilat, and U. Weinsberg, "Friend or faux: Graph-based early detection of fake accounts on social networks," in *Proceedings of The Web Conference 2020*, 2020, pp. 1287–1297.

[87] O. Shchur, M. Mumme, A. Bojchevski, and S. Günnemann, "Pitfalls of graph neural network evaluation," *Relational Representation Learning Workshop, NeurIPS 2018*, 2018.

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

[89] D. Zhu, Z. Zhang, P. Cui, and W. Zhu, "Robust graph convolutional networks against adversarial attacks," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019, pp. 1399–1407.

[90] W. Feng, J. Zhang, Y. Dong, *et al.*, "Graph random neural networks for semi-supervised learning on graphs," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[91] W. Jin, Y. Ma, X. Liu, X. Tang, S. Wang, and J. Tang, "Graph structure learning for robust graph neural networks," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 66–74.

[92] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *arXiv preprint arXiv:1412.6572*, 2014.

[93] W. Hu, B. Liu, J. Gomes, *et al.*, "Strategies for pre-training graph neural networks," in *International Conference on Learning Representations*, 2019.

[94] H. Dai, H. Li, T. Tian, *et al.*, "Adversarial attack on graph structured data," *arXiv preprint arXiv:1806.02371*, 2018.

[95] X. Liu, Y. Li, C. Wu, and C.-J. Hsieh, "Adv-bnn: Improved adversarial defense through robust bayesian neural network," in *International Conference on Learning Representations*, 2018.

[96] Y. You, T. Chen, Y. Sui, T. Chen, Z. Wang, and Y. Shen, "Graph contrastive learning with augmentations," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[97] K. Hassani and A. H. Khasahmadi, "Contrastive multi-view representation learning on graphs," in *International Conference on Machine Learning*, PMLR, 2020, pp. 4116–4126.

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

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

[100] J. Qiu, Q. Chen, Y. Dong, *et al.*, "Gcc: Graph contrastive coding for graph neural network pre-training," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 1150–1160.

[101] J. Shang, T. Ma, C. Xiao, and J. Sun, "Pre-training of graph augmented transformers for medication recommendation," *arXiv preprint arXiv:1906.00346*, 2019.

[102] K. Sun, Z. Zhu, and Z. Lin, "Multi-stage self-supervised learning for graph convolutional networks," *arXiv preprint arXiv:1902.11038*, 2019.

[103] J. Zhuang and M. A. Hasan, "How does bayesian noisy self-supervision defend graph convolutional networks?" *Neural Processing Letters*, pp. 1–22, 2022.

[104] Y. You, T. Chen, Z. Wang, and Y. Shen, "When does self-supervision help graph convolutional networks?" In *International Conference on Machine Learning*, PMLR, 2020, pp. 10 871–10 880.

[105] X. Wang, X. Liu, and C.-J. Hsieh, "Graphdefense: Towards robust graph convolutional networks," *arXiv preprint arXiv:1911.04429*, 2019.

[106] W. Chen, Y. Gu, Z. Ren, *et al.*, "Semi-supervised user profiling with heterogeneous graph attention networks.," in *IJCAI*, vol. 19, 2019, pp. 2116–2122.

[107] Z. Chen, X. Li, and J. Bruna, "Supervised community detection with line graph neural networks," *arXiv preprint arXiv:1705.08415*, 2017.

[108] Z. Zhao, Q. Yang, D. Cai, X. He, and Y. Zhuang, "Expert finding for community-based question answering via ranking metric network learning.," in *Ijcai*, vol. 16, 2016, pp. 3000–3006.

[109] H. Fang, F. Wu, Z. Zhao, X. Duan, Y. Zhuang, and M. Ester, "Community-based question answering via heterogeneous social network learning," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 30, 2016.

[110] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, "Graph convolutional neural networks for web-scale recommender systems," in *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, 2018, pp. 974–983.

[111] W. Fan, Y. Ma, Q. Li, *et al.*, "Graph neural networks for social recommendation," in *The world wide web conference*, 2019, pp. 417–426.

[112] C. Gao, X. Wang, X. He, and Y. Li, "Graph neural networks for recommender system," in *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining*, 2022, pp. 1623–1625.

[113] V. La Gatta, V. Moscato, M. Postiglione, and G. Sperli, "An epidemiological neural network exploiting dynamic graph structured data applied to the covid-19 outbreak," *IEEE Transactions on Big Data*, vol. 7, no. 1, pp. 45–55, 2020.

[114] K. Hsieh, Y. Wang, L. Chen, *et al.*, "Drug repurposing for covid-19 using graph neural network with genetic, mechanistic, and epidemiological validation," *Research Square*, 2020.

[115] C. F. Mountain and C. M. Dresler, "Regional lymph node classification for lung cancer staging," *Chest*, vol. 111, no. 6, pp. 1718–1723, 1997.

[116] S. Bhagat, G. Cormode, and S. Muthukrishnan, "Node classification in social networks," in *Social network data analytics*, Springer, 2011, pp. 115–148.

[117] J. Tang, C. Aggarwal, and H. Liu, "Node classification in signed social networks," in *Proceedings of the 2016 SIAM international conference on data mining*, SIAM, 2016, pp. 54–62.

[118] B. Li and D. Pi, "Learning deep neural networks for node classification," *Expert Systems with Applications*, vol. 137, pp. 324–334, 2019.

[119] W. Jin, Y. Li, H. Xu, *et al.*, "Adversarial attacks and defenses on graphs: A review, a tool and empirical studies," *arXiv preprint arXiv:2003.00653*, 2020.

[120] M. Hahn-Klimroth, G. S. Maesaka, Y. Mogge, S. Mohr, and O. Parczyk, "Random perturbation of sparse graphs," *arXiv preprint arXiv:2004.04672*, 2020.

[121] M. Liu, H. Gao, and S. Ji, "Towards deeper graph neural networks," in *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, 2020, pp. 338–348.

[122] A. Hasanzadeh, E. Hajiramezanali, S. Boluki, *et al.*, "Bayesian graph neural networks with adaptive connection sampling," in *International conference on machine learning*, PMLR, 2020, pp. 4094–4104.

[123] E. Tam and D. Dunson, "Fiedler regularization: Learning neural networks with graph sparsity," in *International Conference on Machine Learning*, PMLR, 2020, pp. 9346–9355.

[124] Y. Ye and S. Ji, "Sparse graph attention networks," *IEEE Transactions on Knowledge and Data Engineering*, 2021.

[125] D. Zügner and S. Günnemann, "Adversarial attacks on graph neural networks via meta learning," *arXiv preprint arXiv:1902.08412*, 2019.

[126] J. Dai, W. Zhu, and X. Luo, "A targeted universal attack on graph convolutional network by using fake nodes," *Neural Processing Letters*, pp. 1–17, 2022.

[127] J. Xu, Y. Yang, J. Chen, *et al.*, "Unsupervised adversarially-robust representation learning on graphs," *arXiv preprint arXiv:2012.02486*, 2020.

[128] X. Tang, Y. Li, Y. Sun, H. Yao, P. Mitra, and S. Wang, "Transferring robustness for graph neural network against poisoning attacks," in *Proceedings of the 13th International Conference on Web Search and Data Mining*, 2020, pp. 600–608.

[129] C. Zheng, B. Zong, W. Cheng, *et al.*, "Robust graph representation learning via neural sparsification," in *International Conference on Machine Learning*, PMLR, 2020, pp. 11 458–11 468.

[130] M. E. A. Seddik, C. Wu, J. F. Lutzeyer, and M. Vazirgiannis, "Node feature kernels increase graph convolutional network robustness," *arXiv preprint arXiv:2109.01785*, 2021.

[131] H. Chang, Y. Rong, T. Xu, *et al.*, "Not all low-pass filters are robust in graph convolutional networks," *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[132] F. Regol, S. Pal, J. Sun, Y. Zhang, Y. Geng, and M. Coates, "Node copying: A random graph model for effective graph sampling," *Signal Processing*, vol. 192, p. 108 335, 2022.

[133] H. Xu, L. Xiang, J. Yu, A. Cao, and X. Wang, "Speedup robust graph structure learning with low-rank information," in *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, 2021, pp. 2241–2250.

[134] K. Xu, S. Liu, P.-Y. Chen, *et al.*, "Towards an efficient and general framework of robust training for graph neural networks," in *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, 2020, pp. 8479–8483.

[135] X. Zhu and Z. Ghahramani, "Learning from labeled and unlabeled data with label propagation," 2002.

[136] H. Wang and J. Leskovec, "Unifying graph convolutional neural networks and label propagation," *arXiv preprint arXiv:2002.06755*, 2020.

[137] F. Wang and C. Zhang, "Label propagation through linear neighborhoods," *IEEE Transactions on Knowledge and Data Engineering*, vol. 20, no. 1, pp. 55–67, 2007.

[138] M. Speriosu, N. Sudan, S. Upadhyay, and J. Baldridge, "Twitter polarity classification with label propagation over lexical links and the follower graph," in *Proceedings of the First workshop on Unsupervised Learning in NLP*, 2011, pp. 53–63.

[139] B. Wang, Z. Tu, and J. K. Tsotsos, "Dynamic label propagation for semi-supervised multi-class multi-label classification," in *Proceedings of the IEEE international conference on computer vision*, 2013, pp. 425–432.

[140] M. Karasuyama and H. Mamitsuka, "Multiple graph label propagation by sparse integration," *IEEE transactions on neural networks and learning systems*, vol. 24, no. 12, pp. 1999–2012, 2013.

[141] J. Ugander and L. Backstrom, "Balanced label propagation for partitioning massive graphs," in *Proceedings of the sixth ACM international conference on Web search and data mining*, 2013, pp. 507–516.

[142] C. Gong, D. Tao, W. Liu, L. Liu, and J. Yang, "Label propagation via teaching-to-learn and learning-to-teach," *IEEE transactions on neural networks and learning systems*, vol. 28, no. 6, pp. 1452–1465, 2016.

[143] A. Iscen, G. Tolias, Y. Avrithis, and O. Chum, "Label propagation for deep semi-supervised learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 5070–5079.

[144] Q. Huang, H. He, A. Singh, S.-N. Lim, and A. R. Benson, "Combining label propagation and simple models out-performs graph neural networks," *arXiv preprint arXiv: 2010.13993*, 2020.

[145] M. Turner and J. Austin, "Graph matching by neural relaxation," *Neural computing & applications*, vol. 7, pp. 238–248, 1998.

[146] M. Skomorowski, "Use of random graph parsing for scene labeling by probabilistic relaxation," *Pattern Recognition Letters*, vol. 20, no. 9, pp. 949–956, 1999.

[147] A. Kostin, J. Kittler, and W. Christmas, "Object recognition by symmetrised graph matching using relaxation labeling with an inhibitory mechanism," *Pattern Recognition Letters*, vol. 26, no. 3, pp. 381–393, 2005.

[148] H. Wang and E. R. Hancock, "Probabilistic relaxation labelling using the fokker–planck equation," *Pattern Recognition*, vol. 41, no. 11, pp. 3393–3411, 2008.

[149] R. Tanno, A. Saeedi, S. Sankaranarayanan, D. C. Alexander, and N. Silberman, "Learning from noisy labels by regularized estimation of annotator confusion," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 11 244–11 253.

[150] J. Li, R. Socher, and S. C. Hoi, "Dividemix: Learning with noisy labels as semi-supervised learning," *arXiv preprint arXiv:2002.07394*, 2020.

[151] J. Zhu, Y. Yan, L. Zhao, M. Heimann, L. Akoglu, and D. Koutra, "Beyond homophily in graph neural networks: Current limitations and effective designs," *Advances in Neural Information Processing Systems*, vol. 33, pp. 7793–7804, 2020.

[152] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning," in *international conference on machine learning*, PMLR, 2016, pp. 1050–1059.

[153] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *2008 Eighth IEEE International Conference on Data Mining*, IEEE, 2008, pp. 413–422.

[154] L. M. Manevitz and M. Yousef, "One-class svms for document classification," *Journal of machine Learning research*, vol. 2, no. Dec, pp. 139–154, 2001.

[155] T. Schlegl, P. Seeböck, S. M. Waldstein, U. Schmidt-Erfurth, and G. Langs, "Unsupervised anomaly detection with generative adversarial networks to guide marker discovery," *arXiv preprint arXiv:1703.05921*, 2017.

[156] B. Zong, Q. Song, M. R. Min, *et al.*, "Deep autoencoding gaussian mixture model for unsupervised anomaly detection," 2018.

[157] G. Koch, R. Zemel, and R. Salakhutdinov, "Siamese neural networks for one-shot image recognition," in *ICML deep learning workshop*, Lille, vol. 2, 2015.

[158] S. Ravi and H. Larochelle, "Optimization as a model for few-shot learning," 2016.

[159] R. Socher, M. Ganjoo, C. D. Manning, and A. Ng, "Zero-shot learning through cross-modal transfer," in *Advances in neural information processing systems*, 2013, pp. 935–943.

[160] M. Palatucci, D. Pomerleau, G. E. Hinton, and T. M. Mitchell, "Zero-shot learning with semantic output codes," in *Advances in neural information processing systems*, 2009, pp. 1410–1418.

[161] Y. Wang, W. Liu, X. Ma, *et al.*, "Iterative learning with open-set noisy labels," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 8688–8696.

[162] C. Geng and S. Chen, "Collective decision for open set recognition," *IEEE Transactions on Knowledge and Data Engineering*, 2020.

[163] M. Hassen and P. K. Chan, "Learning a neural-network-based representation for open set recognition," in *Proceedings of the 2020 SIAM International Conference on Data Mining*, SIAM, 2020, pp. 154–162.

[164] P. Oza and V. M. Patel, "C2ae: Class conditioned auto-encoder for open-set recognition," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 2307–2316.

[165] P. Perera, V. I. Morariu, R. Jain, *et al.*, "Generative-discriminative feature representations for open-set recognition," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 11 814–11 823.

[166] T. Mensink, J. Verbeek, F. Perronnin, G. Csurka, *et al.*, "Distance-based image classification: Generalizing to new classes at near zero cost," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, 2013.

[167] Z. Liu, Z. Miao, X. Zhan, J. Wang, B. Gong, and S. X. Yu, "Large-scale long-tailed recognition in an open world," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 2537–2546.

[168] D. W. Matan Ben-Yosef, "Gaussian mixture generative adversarial networks for diverse datasets, and the unsupervised clustering of images," *arXiv preprint arXiv: 1808.10356*, 2018.

[169] H. Zenati, M. Romain, C.-S. Foo, B. Lecouat, and V. Chandrasekhar, "Adversarially learned anomaly detection," in *2018 IEEE International Conference on Data Mining (ICDM)*, IEEE, 2018, pp. 727–736.

[170] L. Dhanabal and S. Shantharajah, "A study on nsl-kdd dataset for intrusion detection system based on classification algorithms," *International Journal of Advanced Research in Computer and Communication Engineering*, vol. 4, no. 6, pp. 446–452, 2015.

[171] N. Moustafa and J. Slay, "Unsw-nb15: A comprehensive data set for network intrusion detection systems (unsw-nb15 network data set)," in *2015 military communications and information systems conference (MilCIS)*, IEEE, 2015, pp. 1–6.

[172] D. Pelleg, A. W. Moore, *et al.*, "X-means: Extending k-means with efficient estimation of the number of clusters.," in *Icml*, vol. 1, 2000, pp. 727–734.

VITA

Jun Zhuang received his B.E. in Safety Engineering from the South China University of Technology (SCUT) in July 2011. He then obtained an M.S. in Finance from the Rochester Institute of Technology (RIT) in August 2013. After that, he worked at China Merchants Bank Co., Ltd. from January 2014 to July 2016. In 2018, after receiving an M.S. in computer science from the State University of New York at Buffalo (SUNY Buffalo), he joined the Indiana University-Purdue University Indianapolis (IUPUI) to pursue his Ph.D. degree in computer science under the supervision of Dr. Mohammad Al Hasan. In 2021, he earned an M.S. degree in Computer & Information Science in the same department. He is mainly interested in addressing the robustness issues in the graph domain via Bayesian and deep-learning approaches. During this period, he interned at the University of Tennessee, Knoxville, Roche Diabetes Care, Inc., and Uber Technologies, Inc. in the summer of 2020, 2021, and 2022, respectively. In the Fall of 2023, he will join the computer science department as a tenure-track assistant professor at Boise State University.