BEYOND ACCURATE GRAPH NEURAL NETWORKS

A Dissertation in
Information Sciences and Technology
by
Xianfeng Tang

© 2020 Xianfeng Tang

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Doctor of Philosophy

December 2020
The dissertation of Xianfeng Tang was reviewed and approved by the following:

Prasenjit Mitra  
Professor of Information Sciences and Technology  
Dissertation Co-Advisor  
Co-Chair of Committee

Suhang Wang  
Assistant Professor of Information Sciences and Technology  
Dissertation Co-Advisor  
Co-Chair of Committee

Xiang Zhang  
Associate Professor of Information Sciences and Technology

Wang-Chien Lee  
Associate Professor of Department of Computer Science and Engineering

Mary Beth Rosson  
Professor of Information Sciences and Technology  
Program Head of the College of Information Sciences and Technology
Abstract

Graphs are ubiquitous and are widely used to represent non-Euclidean correlations among entities. A large amount of graph data are collected to study science, human society, and economics, such as academic graphs, social correlations, and transportation networks. The advances in both machine learning algorithms and computer hardware make it possible to train deep neural networks on large-scale datasets. Inspired and motivated by the success of neural networks in computer vision, natural language processing, and graphics, researchers have developed Graph Neural Networks (GNNs) that leverage the power of deep learning for graph-based machine learning tasks. In recent years, various GNN models are proposed to improve the representation learning on both node/edge features and graph topology. Though these models have provided accurate outcomes for different regression/classification tasks, some essential characteristics of GNNs are neglected including robustness, fairness, and explainability, making the models less reliable and less trustful in many applications. For example, adopting graph neural networks on social graphs contributes to credit score analysis and loan prediction. However, if the models are not robust, criminals can raise their credit score and loan approval rate by adversarial attacks; GNN models for job candidates ranking and resume screening applications will lead to Unfair competitions with biased training data regarding gender and race. While the accuracy of GNN approaches is pioneered in most tasks, their black-box designs have obstructed researchers from understanding how GNNs produce outputs and why GNNs make mistakes. It is critical to improving the robustness, fairness, and explainability of accurate GNN models.

However, building robust, unbiased, and explainable GNN models is an extremely challenging task, mainly owing to the unique message-passing mechanism of GNNs. Different from other neural network designs, GNN models not only transform feature vectors between layers but also aggregate information from neighborhoods for every node in the graph. The message-passing mechanism could broadcast perturbations injected by attackers and harm the representation learning on nodes. Besides, since many semi-supervised tasks on graphs are trained and tested on the same graph structure, GNNs need to defend against adversarial attacks in their training phase. How to design robust GNN models to jointly address these challenges is not well studied. Similarly, the fairness of GNNs is also different from other neural networks. The reasons are two folds: (1) GNNs require new definitions of fairness that not only cover node
features but also cover properties graph topological structure, such as degree distribution and centrality; (2) algorithms and approaches should be developed accordingly to train/learn GNN models that satisfy the above goals. Unfortunately, fair and unbiased GNN models are still under-explored. Finally, the unique message-passing mechanism requires us to design special explanations that can consider the non-Euclidean graph topology, as well as new approaches to deriving the explanations.

In this dissertation, I will go beyond accurate GNNs and focus on the robustness, fairness, and explainability of GNN models. I will start by introducing GNN designs, as well as the limitations of GNNs regarding robustness, fairness, and explainability. Then I will show three novel GNN models that enhance the robustness, fairness, and explainability respectively. Each GNN model is developed based on real-world observations and theoretical analysis to tackle the most challenging parts of the under-explored limitations. Besides, I validate all proposed models on benchmark datasets and in real-world industrial applications, showing that the proposed methods/models achieve significant improvements over state-of-the-art approaches.
# Table of Contents

List of Figures ........................................ viii
List of Tables ........................................... x
Acknowledgments .......................................... xi

## Chapter 1

### Introduction

1.1 A General Definition of Graph Neural Networks .................. 1
   1.1.1 Basic GNN Design ................................................... 2
   1.1.2 GNN for Classification Tasks .................................... 2
   1.1.3 GNN for Regression Tasks ........................................ 2
1.2 Limitations ................................................ 3
1.3 Challenges and Solutions ....................................... 4
   1.3.1 Robustness .......................................................... 4
   1.3.2 Fairness ............................................................ 5
   1.3.3 Explainability ...................................................... 6
1.4 Overview .................................................. 6

## Chapter 2

### Robustness of Graph Neural Networks ............................ 7

2.1 Introduction ................................................... 8
2.2 Related Work ................................................ 10
   2.2.1 Graph Neural Networks .......................................... 10
   2.2.2 Adversarial Attack and Defense on Graphs .................... 10
2.3 Problem Definition ............................................ 11
2.4 Proposed Framework .......................................... 12
   2.4.1 Penalized Aggregation Mechanism .............................. 12
   2.4.2 Transfer with Meta-Optimization ............................... 15
2.5 Experiments .................................................................................. 17
  2.5.1 Experimental Setup .................................................................. 17
    2.5.1.1 Datasets ................................................................. 17
    2.5.1.2 Attack Methods ..................................................... 19
    2.5.1.3 Baselines .............................................................. 19
    2.5.1.4 Settings and Parameters ......................................... 21
  2.5.2 Robustness Comparison .......................................................... 21
    2.5.2.1 Defense Against Non-Targeted Attack ....................... 21
    2.5.2.2 Defense Against Targeted Attack ............................. 22
    2.5.2.3 Defense Against Random Attack ............................. 22
  2.5.3 Ablation Study ......................................................................... 23
    2.5.3.1 Varying the Penalized Aggregation Mechanism ........... 23
    2.5.3.2 Varying the Meta-Optimization Algorithm ................ 24
  2.5.4 Parameter Sensitivity Analysis ............................................... 25

Chapter 3
Fairness of Graph Neural Networks .................................................. 26
  3.1 Introduction .............................................................................. 27
  3.2 Related Work .......................................................................... 29
    3.2.1 Graph Convolutional Neural Networks ......................... 29
    3.2.2 Self-Supervised Learning .............................................. 29
    3.2.3 Fairness on Graph ....................................................... 30
  3.3 Preliminaries ........................................................................... 30
  3.4 Data Analysis .......................................................................... 31
    3.4.1 Degree Distribution ...................................................... 32
    3.4.2 Accuracy Varying Node Degree ..................................... 32
    3.4.3 Labeled Neighbor Distribution .................................... 33
    3.4.4 Bridging Node Degree and Biases in GCNs ................... 33
  3.5 Approach ................................................................................... 36
    3.5.1 Degree-Specific GCN Layer ......................................... 37
    3.5.2 Self-Supervised-Training with Bayesian Teacher Network 38
      3.5.2.1 Pre-training with the Annotator ............................... 39
      3.5.2.2 Fine-tuning Student with Uncertainty Scores ............ 41
    3.5.3 Training Algorithm ........................................................ 42
  3.6 Experiments .............................................................................. 43
    3.6.1 Experimental Setup ....................................................... 43
      3.6.1.1 Datasets ............................................................. 43
      3.6.1.2 Baselines .......................................................... 44
      3.6.1.3 Settings and Hyperparameters ............................... 45
    3.6.2 Node Classification Performance .................................. 46
    3.6.3 Performance on Low Degree Nodes ............................... 46
    3.6.4 Ablation Study ............................................................... 47
Chapter 4
Explainability of Graph Neural Networks 50

4.1 Introduction .................................................. 50
4.2 Related Work .................................................. 52
  4.2.1 User Behaviour Modeling ................................. 52
  4.2.2 Explainable Machine Learning ......................... 53
4.3 Preliminaries .................................................. 53
4.4 Our Approach: FATE ......................................... 56
  4.4.1 Friendship Module .................................... 56
  4.4.2 Temporal Module ..................................... 59
  4.4.3 User Engagement Score Generation .................. 59
  4.4.4 Explainable User Engagement ......................... 61
  4.4.5 Complexity Analysis .................................. 62
4.5 Evaluation .................................................... 63
  4.5.1 Datasets and Experiment Setup ....................... 63
  4.5.2 Compared Methods .................................... 63
  4.5.3 User Engagement Prediction Performance .............. 64
  4.5.4 Ablation Study ....................................... 65
  4.5.5 Explainability Evaluation .............................. 66
    4.5.5.1 User Action Importance ......................... 66
    4.5.5.2 Temporal Importance ............................ 68
    4.5.5.3 Friendship Importance ........................... 69
    4.5.5.4 Baseline comparisons on explainability .......... 70
  4.5.6 Practical Applications ................................. 71
4.6 Supplementary Material ..................................... 71
  4.6.1 Complexity of FATE ................................. 71
    4.6.1.1 Theoretical Analysis ............................ 71
    4.6.1.2 Experimental Results ............................ 72
  4.6.2 Implementation Details ............................... 73
    4.6.2.1 Experimental Environment ....................... 73
    4.6.2.2 Data Preprocessing ............................... 73
    4.6.2.3 Model Implementations ............................ 74
    4.6.2.4 Evaluation Metrics ............................... 75

Chapter 5
Discussion 76

Bibliography 78
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Overall framework of PA-GNN. Thicker arrows indicate higher attention coefficients. $\theta^*$ denotes the model initialization from meta-optimization.</td>
<td>12</td>
</tr>
<tr>
<td>2.2</td>
<td>Node classification accuracy under random attack.</td>
<td>23</td>
</tr>
<tr>
<td>2.3</td>
<td>Distributions of attention coefficients in PA-GNN.</td>
<td>24</td>
</tr>
<tr>
<td>2.4</td>
<td>Parameter sensitivity analysis.</td>
<td>25</td>
</tr>
<tr>
<td>3.1</td>
<td>Degree distribution.</td>
<td>32</td>
</tr>
<tr>
<td>3.2</td>
<td>Error distribution w.r.t node degree.</td>
<td>33</td>
</tr>
<tr>
<td>3.3</td>
<td>Ratio of being neighbor with a labeled node.</td>
<td>34</td>
</tr>
<tr>
<td>3.4</td>
<td>Distributions of the Influence Score.</td>
<td>36</td>
</tr>
<tr>
<td>3.5</td>
<td>GNN with degree-specific trainable parameters. Node features multiply with different parameters generated by the RNN according to their degree.</td>
<td>38</td>
</tr>
<tr>
<td>3.6</td>
<td>Overall framework of SL-DSGCN.</td>
<td>39</td>
</tr>
<tr>
<td>3.7</td>
<td>Uncertainty score and error distribution of the teacher network. Generally, nodes closer to labeled (training) ones tend to have lower uncertainty and error rate.</td>
<td>41</td>
</tr>
<tr>
<td>3.8</td>
<td>Node Classification Performance on Nodes with Different Degrees</td>
<td>47</td>
</tr>
<tr>
<td>4.1</td>
<td>User graphs are temporal, and capture friendship structure, user actions (node features), and user-user interactions (edge features) over various in-App functions.</td>
<td>54</td>
</tr>
<tr>
<td>4.2</td>
<td>Overall framework of FATE: tGCN-based friendship modules capture local network structure and user interactions at each timestep, and tLSTM captures temporal dynamics for distinct user actions. Finally, an attention mixture mechanism governs user engagement prediction.</td>
<td>55</td>
</tr>
<tr>
<td>4.3</td>
<td>Our proposed friendship module uses a tensor-based GCN with neighbor attention to generate user graph embeddings jointly from ego-networks and interactions.</td>
<td>56</td>
</tr>
<tr>
<td>4.4</td>
<td>Most users communicate frequently only with a subset ($\leq 20%$) of their friends, making careful aggregation important when considering influence from neighbors.</td>
<td>58</td>
</tr>
<tr>
<td>4.5</td>
<td>FATE’s global user action importances derived on Task 1 and Task 2 correctly infer past session time and snap-related actions as the most important for prediction.</td>
<td>66</td>
</tr>
</tbody>
</table>
4.6 Two sample local user action importances: Users with different dominating engagement behaviors exhibit different user action importances.

4.7 FATE’s global temporal importances show long and short-term action importances over time.

4.8 FATE can capture diverse local level temporal importance for users of different persona.

4.9 FATE’s local friendship importance captures asymmetric influence of friends: the user has similar session time behaviors (right) as their highest-importance friends (blue and orange lines are close); session time spikes coincide with high temporal importances (left) of those friends (dark hues).

4.10 Comparisons of explainability.
## List of Tables

2.1 Statistics of datasets ................................................................. 18
2.2 Node classification performance (Accuracy±Std) under non-targeted *metattack* [143] ............................................................. 20
2.3 Node classification accuracy under targeted attack. ...................... 22
2.4 Node classification accuracy of ablations. .................................. 22
2.5 Mean values of attention coefficients. ........................................ 23

3.1 Statistics of the Datasets .............................................................. 44
3.2 Node Classification Performance Comparison on Cora, Citeseer and PubMed .......................................................... 44
3.3 Ablation study on Cora dataset. .................................................. 48
3.4 Ablation study on Citeseer dataset. .............................................. 48
3.5 Influence of the softly-labeled node set. ...................................... 48

4.1 FATE consistently outperforms alternative models in prediction error metrics on both Task 1 and Task 2, and both datasets Region 1 and Region 2. .......... 65
4.2 All components help FATE: Removing (a) tGCN/tLSTM, (b) friendship module, (c) temporal module or (d) user interactions hurts performance. .............. 66
4.3 Comparisons of Runtime (min). FATE reduced 20% of runtime on average comparing with non-tensor-based FATE*ts*. ........................................ 73
4.4 Statistics of Datasets. ................................................................. 73
4.5 Selected features for user actions on Snapchat. ............................. 74
Acknowledgments

Firstly, I want to express the deepest gratitude to my advisers, Dr. Prasenjit Mitra, and Dr. Suhang Wang, for their continuous support of my research and my career. It was a wonderful time when we were working together on challenging problems. Their patience, motivation, and knowledge have significantly influenced me over the past years. There wouldn’t be any research outcomes without the supervision received from Prasenjit and Suhang. In addition, they guided my career development. The enthusiasm, bravery, and curiosity that I learned from them have already become the most precious treasures of my entire life.

Meanwhile, I sincerely appreciate my committee members Dr. Xiang Zhang and Dr. Wang-Chien Lee for their insightful suggestions on my research from different perspectives.

I would like to thank all labmates and collaborators for the discussions and their input to my research.

I also want to appreciate researchers and scientists from the industry for hosting my internships and providing me the opportunities to solve large-scale real-world problems.

Last but not least, I would like to thank my father Zhenghai Tang and and mother Shiyun Sun for their love and support. I also want to thank all families and friends for making it all possible. I would like to thank Zhuoran Li for her encouragement and accompanying.

This dissertation is based upon work supported by, or in part by, the National Science Foundation (NSF) under grant IIS-1909702, IIS-1955851, and the Global Research Outreach program of Samsung Advanced Institute of Technology under grant #225003. The findings and conclusions in this thesis do not necessarily reflect the view of the funding agency.
Introduction

Graphs are ubiquitous and Graph Neural Networks (GNNs) achieve remarkable performance in various applications, such as traffic prediction [109], social network analysis [102], drug discovery [85], circuit design [132], and so on. The success of GNN models is credited to the message-passing mechanism and the non-linear feature transformation. Firstly, the message-passing mechanism aggregates neighborhood information to update node representation on each GNN layer. As a result, node embedding vectors from GNN models preserve knowledge of the graph topological structure. In addition, comparing with conventional graph embedding algorithms such as LINE [100] and node2vec [36], GNN models benefit from their deep neural network designs in automatic feature extraction and representation learning. Therefore, GNN models lead to significant performance improvement in many tasks, and become popular in both academia and industry. Here I first give a general definition of graph neural networks.

1.1 A General Definition of Graph Neural Networks

Let us use $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{X})$ to denote a graph, where $\mathcal{V} = \{v_1, \ldots, v_N\}$ is the set of $N$ nodes, $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ represents the set of edges, and $\mathbf{X} = \{x_1, \ldots, x_N\}$ indicates node features. In a semi-supervised setting, partial nodes come with labels and are defined as $\mathcal{V}^L$, where the corresponding label for node $v$ is denoted by $y_v$. Note that the topology structure of $\mathcal{G}$ is damaged, and the original clean version is unknown. In addition to the poisoned graph $\mathcal{G}$, Let us assume there exists $M$ clean graphs sharing similar domains with $\mathcal{G}$. For example, when $\mathcal{G}$ is the citation network of publications in data mining field, a similar graph can be another citation network from physics. Let us use $\{\mathcal{G}_1, \ldots, \mathcal{G}_M\}$ to represent clean graphs. Similarly, each clean graph consists of nodes and edges. Let us use $\mathcal{V}_i^L$ to denote the labeled nodes in graph $\mathcal{G}_i$. 
1.1.1 Basic GNN Design

Here I introduce the general architecture of a graph neural network. A graph neural network contains multiple layers. Each layer transforms its input node features to another Euclidean space as output. Different from fully-connected layers, a GNN layer takes first-order neighbors’ information into consideration when transforming the feature vector of a node. This “message-passing” mechanism ensures the initial features of any two nodes can affect each other even if they are faraway neighbors, along with the network going deeper. The input node features to the $l$-th layer in an $L$-layer GNN can be represented by a set of vectors $H^l = \{h^l_1, \ldots, h^l_N\}$, $h^l_i \in \mathbb{R}^{d_l}$, where $h^l_i$ corresponds to $v_i$. Obviously, $H^1 = X$. The output node features of the $l$-th layer, which also formulate the input to the next layer, are generated as follows:

$$h^{l+1}_i = \text{Update}[h^l_i, \text{Agg}(h^l_j | j \in \mathcal{N}_i)]$$

(1.1)

where $\mathcal{N}_i$ is the set of first-order neighbors of node $i$, $\text{Agg}(\cdot)$ indicates a generic aggregation function on neighbor nodes, and $\text{Update}(\cdot)$ is an update function that generates a new node representation vector from the previous one and messages from neighbors. Most graph neural networks follow the above definition. For example, Hamilton et al. [38] introduce mean, pooling and LSTM as the aggregation function, Veličković et al. [106] leverage self-attention mechanism to update node representations. A GNN can be represented by a parameterized function $f_\theta$ where $\theta$ represents parameters, the loss function can be represented as $\mathcal{L}_c(\theta)$.

1.1.2 GNN for Classification Tasks

In semi-supervised classification tasks such as predicting gender for unlabeled social network users, the cross-entropy loss function for classification takes the form:

$$\mathcal{L}_c(\theta) = - \sum_{v \in \mathcal{V}^U} y_v \log \hat{y}_v,$$

(1.2)

where $\hat{y}_v$ is the predicted label generated by passing the output from the final GNN layer to a softmax function.

1.1.3 GNN for Regression Tasks

In regression tasks such as analyzing traffic flow/volume, real-valued results are generated by GNNs. Here I use mean square error (MSE) as an example of the loss function, which is defined
as follows:

\[ \mathcal{L}_r(\theta) = \sum_{v \in \mathcal{V}} \| y_v - \hat{y}_v \|^2, \]  

(1.3)

where \( \hat{y}_v \) is the predicted value acquired from a fully-connected neural network layer after the GNN model.

1.2 Limitations

As discussed above, the accuracy of GNN models are usually prioritized. However, purely focus on accuracy will not lead to optimal solutions in many cases. Typically, there are many other requirements and constraints apart from the accuracy of GNN models. Here I give three examples of robustness, fairness, and explainability, accordingly.

Example 1.2.1. (Robustness) Traffic is the pulse of modern cities. Traffic prediction and analysis contribute to city planning, decision making of government and transportation control. By formulating the road network as a graph, GNNs have achieved state-of-the-art performance in traffic prediction tasks [109]. For example, DeepMind leverages GNNs to boost the accuracy of estimated time of arrival (ETA) for Google Maps by up to 50% \(^1\). However, the predicted traffic volume values are highly sensitive to the historical observations. Namely, some small perturbations in the training data can easily “fool” the system. For example, an artist Simon Weckert from Berlin has successfully attacked Google Maps with 99 phones in early 2020 \(^2\). Such a less robust system would suffer from adversarial attackers, leading to unsatisfying user experiences. Therefore, it requires us to develop new techniques to build robust graph neural networks on even perturbed training data.

Example 1.2.2. (Fairness) Machine learning systems have been an inseparable part of our daily lives. GNNs, like other deep neural networks, are widely deployed in interdisciplinary fields and products. For example, self-driving vehicles leverage objection detection algorithms with GNNs [93]; employers accelerate the candidate screening with machine learning models to identify potential talents [127, 55]; biologists discover new drug designs with GNN on 3D chemical structures [65]. The data-driven machine learning algorithms are not absolutely objective. Training machine learning algorithms on noisy and highly-biased will lead to unfair models. For example, it is reported in 2016 that a recidivism prediction model in COMPAS \(^3\) produces much higher false positive rate for black people than white people \(^4\); some job platforms show prefer-

\(^1\)https://deepmind.com/blog/article/traffic-prediction-with-advanced-graph-neural-networks
\(^2\)http://simonweckert.com/googlemapshacks.html
\(^3\)https://en.wikipedia.org/wiki/COMPAS_(software)
\(^4\)https://www.propublica.org/article/machine-bias-risk-assessments-in-criminal-sentencing
ence to less qualified male candidates over more qualified female candidates [55]; several face recognition services are found to suffer from achieving much lower accuracy on females with darker skin color [12]. The ubiquitous of unfairness and discrimination in machine learning models that related to people urges the community to develop fair machine learning and GNN models.

Example 1.2.3. (Explainability) Thanks to the popularity of mobile devices and the increasing in network speed, social network applications (Apps) becomes more and more convenient. Understanding the users is an important task that provides useful insights to a number of downstreaming applications, such as personalized advertisement, designing new features, and App development. Prior research have shown several promising GNN approaches for user analysis [29, 102]. However, the black-box designs of deep neural networks limit their explainability. It is difficult to understand how the outcomes are generated, and why the framework makes mistakes in certain scenarios. Developing advanced techniques to understand and explain the behaviours of graph neural networks can benefit a wide range of real-world applications.

However, it is non-trivial to tackle the above limitations for GNNs. Not only because of the unique message-passing mechanism of graph neural networks, but also owing to the characteristic of semi-supervised learning tasks. In the following section, I identify the challenges of building robust, unbiased and explainable GNN models and introduce my solutions correspondingly.

1.3 Challenges and Solutions

1.3.1 Robustness

Similar to other neural network models, GNNs also suffer from adversarial attacks. Namely, a small, unnoticeable perturbation on node features or on graph topological structure could significantly reduce the accuracy of GNNs significantly. The reason why GNNs are vulnerable are two folds: (1) the neural-network-based feature transformation in GNN layers is sensitive to adversarial attacks; (2) the message-passing mechanism could amplify perturbations and errors on graphs with modified topological structure. Apart from other neural networks, there are several unique challenges of building robust GNNs. I summarize them as follows:

- Different from adversarial attack and defense in computer vision field, unnoticeable modifications could only happen in graph structure. How to identify adversarial edges and elevate their negative influences is under-explored.
As most applications on graph-structured data are semi-supervised learning tasks, the training and inference of GNNs are processed on the same graph. The lack of supervised information about real perturbations in a poisoned graph obstructs training robust GNNs.

To tackle the above challenges, I explore clean graphs for improving the robustness of GNNs against poisoning attacks. In particular, I propose a novel message-passing mechanism that penalize perturbations to reduce the negative effects of adversarial attacks (first challenge); I then train such message-passing mechanism on clean graphs from similar domains with synthetic perturbations, and transfer the alleviation ability to the target poisoned graph with meta-optimization (second challenge). Our solution demonstrate dramatic improvement of robustness against the strongest adversarial attack method, comparing with state-of-the-art defending approaches for GNNs. I will introduce the details of the robust GNN design in Chapter 2.

1.3.2 Fairness

Fairness has become a critical issue as more and more parts of the modern society are being automated by machine learning frameworks. It has been widely shown that machine learning algorithms would show and even amplify the biases contained in the training data. Undoubtedly, GNNs and GNN-based machine learning frameworks also need extra constraints and regularizations to guarantee their fairness. However, different from other neural networks, GNNs take graph as the input. Since graph is not defined in Euclidean space, there are some potential challenges when constructing fair GNN models:

- Since some important properties (e.g., degree distribution, centrality) of a graph are defined on the topological structure, they should also be covered in a fair GNN design like node features such as gender and race. Can we build a GNN-based loan prediction framework that judge normal people and super-stars equally?

- The biases of GNNs on graph properties lack both empirical and theoretical analysis. How to quantify the bias of GNNs w.r.t graph properties is an open question.

- How to design/train GNNs to reduce the biases on graph property is not studied yet.

In attempting to solve the above challenges, I first study a novel problem of addressing the degree-related biases in GNNs. To the best of our knowledge, it is the first try to empirically and theoretically analyze the unfairness of GNNs w.r.t degree distribution. I then design a self-supervised learning framework to reduce the degree-related biases and train fair GNNs with degree-specific layers. The entire pipeline is validated on benchmark graph datasets and not
only outperform baseline approaches but also reduce the degree-related biases significantly. The
detailed approach is described in Chapter 3.

1.3.3 Explainability

While the performance of GNNs is satisfying in various graph-related applications, the transparency of GNNs is rather limited. Similar to other black-box deep learning frameworks, it is difficult for humans to explain how GNN models compute the outputs. Recently, Ying et al.[129] introduce a post-hoc GNN explanation framework that learns a compact sub-graph indicating the importance of nodes and edges. However, when the explanation meets real-world applications, more challenges arise:

- The post-analyzing interpretation approaches are inefficient and hard to scale. In real-world industrial applications, the explanation method must be fast and can be deployed to large systems.
- Many attention-based interpretation approaches [82, 37] are able to generate explanations on-the-fly. However, they are typically designed for a specific neural network architecture such as LSTMs [41] and convolutional neural networks (CNNs). How to provide meaningful attentive explanations for GNN structures in hierarchical deep learning frameworks is not well-studied.

In Chapter 4, I take explainable user engagement prediction for social network applications as an example, to tackle the above challenges. In particular, I first design a flexible definition of the explanations based on attention mechanism that can be learned efficiently. I also propose an end-to-end self-explainable deep learning framework based on GNNs and other neural architectures to jointly predict user engagement scores and derive meaningful explanations. The proposed framework is validated on two real-world datasets from Snapchat, with a significant improvement of accuracy and a remarkable runtime reduction against state-of-the-art methods.

1.4 Overview

This dissertation is divided into five parts. First, I discuss the robustness of GNNs in Chapter 2. Then, I introduce the degree-related biases of GNNs and the corresponding solution in Chapter 3. The explainability of GNNs is studied in Chapter 4. Finally, I conclude the dissertation and discuss future directions in Chapter 5.
Robustness of Graph Neural Networks

Graph neural networks (GNNs) are widely used in many applications. However, their robustness against adversarial attacks is criticized. Prior studies show that using unnoticeable modifications on graph topology or nodal features can significantly reduce the performances of GNNs. It is very challenging to design robust graph neural networks against poisoning attack and several efforts have been taken. Existing work aims at reducing the negative impact from adversarial edges only with the poisoned graph, which is sub-optimal since they fail to discriminate adversarial edges from normal ones. On the other hand, clean graphs from similar domains as the target poisoned graph are usually available in the real world. By perturbing these clean graphs, we create supervised knowledge to train the ability to detect adversarial edges so that the robustness of GNNs is elevated. However, such potential for clean graphs is neglected by existing work. To this end, we investigate a novel problem of improving the robustness of GNNs against poisoning attacks by exploring clean graphs. Specifically, we propose PA-GNN, which relies on a penalized aggregation mechanism that directly restrict the negative impact of adversarial edges by assigning them lower attention coefficients. To optimize PA-GNN for a poisoned graph, we design a meta-optimization algorithm that trains PA-GNN to penalize perturbations using clean graphs and their adversarial counterparts, and transfers such ability to improve the robustness of PA-GNN on the poisoned graph. Experimental results on four real-world datasets demonstrate the robustness of PA-GNN against poisoning attacks on graphs.
2.1 Introduction

Graph neural networks (GNNs) [52, 38], which explore the power of neural networks for graph data, have achieved remarkable results in various applications [106, 29, 43]. The key to the success of GNNs is its signal-passing process [110], where information from neighbors is aggregated for every node in each layer. The collected information enriches node representations, preserving both nodal feature characteristics and topological structure.

Though GNNs are effective for modeling graph data, the way that GNNs aggregate neighbor nodes’ information for representation learning makes them vulnerable to adversarial attacks [142, 143, 21, 111, 117]. Poisoning attack on a graph [142], which adds/deletes carefully chosen edges to the graph topology or injects carefully designed perturbations to nodal features, can contaminate the neighborhoods of nodes, bring noises/errors to node representations, and degrade the performances of GNNs significantly. The lack of robustness become a critical issue of GNNs in many applications such as financial system and risk management [1]. For example, fake accounts created by a hacker can add friends with normal users on social networks to promote their scores predicted by a GNN model. A model that’s not robust enough to resist such “cheap” attacks could lead to serious consequences. Hence, it is important to develop robust GNNs against adversarial attacks. Recent studies of adversarial attacks on GNNs suggest that adding perturbed edges is more effective than deleting edges or adding noises to node features [111]. This is because node features are usually high-dimensional, requiring larger budgets to attack. Deleting edges only result in the loss of some information while adding edges is cheap to contaminate information passing dramatically. For example, adding a few bridge edges connecting two communities can affect the latent representations of many nodes. Thus, we focus on defense against the more effective poisoning attacks that a training graph is poisoned with injected adversarial edges.

To defend against the injected adversarial edges, a natural idea is to delete these adversarial edges or reduce their negative impacts. Several efforts have been made in this direction [137, 111, 44]. For example, Wu et al. [111] utilize Jaccard similarity of features to prune perturbed graphs with the assumption that connected nodes have high feature similarity. RGCN in [137] introduce Gaussian constrains on model parameters to absorb the effects of adversarial changes. The aforementioned models only rely on the poisoned graph for training, leading to sub-optimal solutions. The lack of supervised information about real perturbations in a poisoned graph obstructs models from modeling the distribution of adversarial edges. Therefore, exploring alternative supervision for learning the ability to reduce the negative effects of adversarial edges is promising.

There usually exist clean graphs with similar topological distributions and attribute features
to the poisoned graph. For example, Yelp and Foursquare have similar co-review networks where the nodes are restaurants and two restaurants are linked if the number of co-reviewers exceeds a threshold. Facebook and Twitter can be treated as social networks that share similar domains. It is not difficult to acquire similar graphs for the targeted perturbed one. As shown in existing work [94, 57], because of the similarity of topological and attribute features, we can transfer knowledge from source graphs to target ones so that the performance on target graphs is elevated. Similarly, we can inject adversarial edges to clean graphs as supervisions for training robust GNNs, which are able to penalize adversarial edges. Such ability can be further transferred to improve the robustness of GNNs on the poisoned graph. Leveraging clean graphs to build robust GNNs is a promising direction. However, prior studies in this direction are rather limited.

Therefore, in this chapter, we investigate a novel problem of exploring clean graphs for improving the robustness of GNNs against poisoning attacks. The basic idea is first learning to discriminate adversarial edges, thereby reducing their negative effects, then transferring such ability to a GNN on the poisoned graph. In essence, we are faced with two challenges: (i) how to mathematically utilize clean graphs to equip GNNs with the ability of reducing negative impacts of adversarial edges; and (ii) how to effectively transfer such ability learned on clean graphs to a poisoned graph. In an attempt to solve these challenges, we propose a novel framework Penalized Aggregation GNN (PA-GNN). Firstly, clean graphs are attacked by adding adversarial edges, which serve as supervisions of known perturbations. With these known adversarial edges, a penalized aggregation mechanism is then designed to learn the ability of alleviating negative influences from perturbations. We further transfer this ability to the target poisoned graph with a special meta-optimization approach, so that the robustness of GNNs is elevated. To the best of our knowledge, we are the first one to propose a GNN that can directly penalize perturbations and to leverage transfer learning for enhancing the robustness of GNN models. The main contributions of this chapter are:

- We study a new problem and propose a principle approach of exploring clean graphs for learning a robust GNN against poisoning attacks on a target graph;
- We provide a novel framework PA-GNN, which is able to alleviate the negative effects of adversarial edges with carefully designed penalized aggregation mechanism, and transfer the alleviation ability to the target poisoned graph with meta-optimization;
- We conduct extensive experiments on real-world datasets to demonstrate the effectiveness of PA-GNN against various poisoning attacks and to understand its behaviors.

The rest of the chapter is organized as follows. We review related work in Section 2. We
define our problems in Section 3. We introduce the details of PA-GNN in Section 4. Extensive experiments and their results are illustrated and analyzed in Section 5. We conclude the chapter in Section 6.

2.2 Related Work

In this section, we briefly review related works, including graph neural networks, adversarial attack and defense on graphs.

2.2.1 Graph Neural Networks

In general, graph neural networks refer to all deep learning methods for graph data [114, 71, 73, 74, 25, 24]. It can be generally categorized into two categories, i.e., spectral-based and spatial-based. Spectral-based GNNs define “convolution” following spectral graph theory [11]. The first generation of GCNs are developed by Bruna et al. [11] using spectral graph theory. Various spectral-based GCNs are developed later on [22, 52, 40, 61]. To improve efficiency, spatial-based GNNs are proposed to overcome this issue [38, 76, 77, 32]. Because spatial-based GNNs directly aggregate neighbor nodes as the convolution, and are trained on mini-batches, they are more scalable than spectral-based ones. Recently, Veličković et al. [106] propose graph attention network (GAT) that leverages self-attention of neighbor nodes for the aggregation process. The major idea of GATs [133] is focusing on most important neighbors and assign higher weights to them during the information passing. However, existing GNNs aggregates neighbors' information for representation learning, making them vulnerable to adversarial attacks, especially perturbed edges added to the graph topology. Next, we review adversarial attack and defense methods on graphs.

2.2.2 Adversarial Attack and Defense on Graphs

Neural networks are widely criticized due to the lack of robustness [35, 62, 63, 16, 64, 59], and the same to GNNs. Various adversarial attack methods have been designed, showing the vulnerability of GNNs [21, 7, 14, 140, 116]. There are two major categories of adversarial attack methods, namely evasion attack and poisoning attack. Evasion attack focuses on generating fake samples for a trained model. Dai et al. [21] introduce an evasion attack algorithm based on reinforcement learning. On the contrary, poisoning attack changes training data, which can decrease the performance of GNNs significantly. For example, Zügner et al. [142] propose nettack which make GNNs fail on any selected node by modifying its neighbor connections. They further develop metattack [143] that reduces the overall performance of GNNs. Comparing
with evasion attack, poisoning attack methods are usually stronger and can lead to an extremely low performance [142, 137, 98], because of its destruction of training data. Besides, it is almost impossible to clean up a graph which is already poisoned. Therefore, we focus on defending the poisoning attack of graph data in this chapter.

How to improve the robustness of GNNs against adversarial poisoning attacks is attracting increasing attention and initial efforts have been taken [117, 111, 137, 44, 140, 8, 141, 136, 134, 69, 33]. For example, Wu et al. [111] utilize the Jaccard similarity of features to prune perturbed graphs with the assumption that connected nodes should have high feature similarity. RGCN in [137] adopts Gaussian distributions as the node representations in each convolutional layer to absorb the effects of adversarial changes in the variances of the Gaussian distributions. More recent approaches formulate the defending tasks as optimization problems, where the clean graph structure and node features are learned from pre-defined constraints, such as low-rank and sparsity [46, 27]. The basic idea of aforementioned robust GNNs against poisoning attack is to alleviate the negative effects of the perturbed edges. However, perturbed edges are treated equally as normal edges during aggregation in existing robust GNNs.

The proposed PA-GNN is inherently different from existing works: (i) instead of purely trained on the poisoned target graph, adopting clean graphs with similar domains to learn the ability of penalizing perturbations; and (ii) investigating meta-learning to transfer such ability to the target poisoned graph for improving the robustness.

2.3 Problem Definition

The problem of exploring clean graphs for learning a robust GNN against poisoning attacks on a target graph is formally defined as:

**Problem 1.** Given the target graph $G$ that is poisoned with adversarial edges, a set of clean graphs $\{G_1, \ldots, G_M\}$ from similar domain as $G$, and the partially labeled nodes of each graph (i.e., $\{V^L_1, \ldots, V^L_M; V^U\}$), we aim at learning a robust GNN to predict the unlabeled nodes of $G$.

It is worth noting that, in this chapter, we learn a robust GNN for semi-supervised node classification. The proposed PA-GNN is a general framework for learning robust GNN of various graph mining tasks such as link prediction.
2.4 Proposed Framework

In this section, we give the details of PA-GNN. An illustration of the framework is shown in Figure 2.1. Firstly, clean graphs \{G_1, \ldots, G_M\} are introduced to generate perturbed edges. The generated perturbations then serve as supervised knowledge to train a model initialization for PA-GNN using meta-optimization. Finally, we fine-tune the initialization on the target poisoned graph for the best performance. Thanks to the meta-optimization, the ability to reduce negative effects of adversarial attack is retained after adapting to \( G \). In the following sections, we introduce technical details of PA-GNN.

2.4.1 Penalized Aggregation Mechanism

We begin by analyzing the reason why GNNs are vulnerable to adversarial attacks with the general definition of GNNs in Equation 1.1. Suppose the graph data fed into a GNN is perturbed, the aggregation function \( \text{Agg}(\cdot) \) treats “fake” neighbors equally as normal ones, and propagates their information to update other nodes. As a result, GNNs fail to generate desired outputs under influence of adversarial attacks. Consequently, if messages passing through perturbed edges are filtered, the aggregation function will focus on “true” neighbors. In an ideal condition, GNNs can work well if all perturbed edges produced by attackers are ignored.

Motivated by above analysis, we design a novel GNN with penalized aggregation mechanism (PA-GNN) which automatically restrict the message-passing through perturbed edge. Firstly, we adopt similar implementation from [105] and define the self-attention coefficient \( a_{i,j}^t \) for node...
features of \(v_i\) and \(v_j\) on the \(l\)-the layer using a non-linear function:

\[
a_{ij}^l = \text{LeakyReLU}
\left((a^l)^\top [W^l h_i^l \oplus W^l h_j^l]\right),
\]

(2.1)

where \(a^l\) and \(W^l\) are parameters, \(\top\) represents the transposition, and \(\oplus\) indicates the concatenation of vectors. Note that coefficients are only defined for first-order neighbors. Take \(v_i\) as an example, we only compute \(a_{ij}^l\) for \(j \in \mathcal{N}_i\), which is the set of direct neighbors of \(v_i\). The attention coefficients related to \(v_i\) are further normalized among all nodes in \(\mathcal{N}_i\) for comparable scores:

\[
\alpha_{ij}^l = \frac{\exp(a_{ij}^l)}{\sum_{k \in \mathcal{N}_i} \exp(a_{ik}^l)}.
\]

(2.2)

We use normalized attention coefficient scores to generate a linear combination of their corresponding node features. The linear combination process serves as the aggregating process, and its results are utilized to update node features. More concretely, a graph neural network layer is constructed as follows:

\[
h_i^{l+1} = \sigma \left( \sum_{j \in \mathcal{N}_i} \alpha_{ij}^l W^l h_j^l \right).
\]

(2.3)

A similar definition can be found in [106]. Clearly, the above design of GNN layer cannot discriminate perturbed edges, let alone alleviate their negative effects on the “message-passing” mechanism, because there is no supervision to teach it how to honor normal edges and punish perturbed ones. A natural solution to this problem is reducing the attention coefficients for all perturbed edges in a poisoned graph. Noticing the exponential rectifier in Equation 2.2, a lower attention coefficient only allows little information passing through its corresponding edge, which mitigate negative effects if the edge is an adversarial one. Moreover, since normalized attention coefficient scores of one node always sum up to 1, reducing the attention coefficient for perturbed edges will also introduce more attention to clean neighbors. To measure the attention coefficients received by perturbed edges, we propose the following metric:

\[
\mathcal{S}_p = \sum_{l=1}^{L} \sum_{e_{ij} \in \mathcal{P}} a_{ij}^l,
\]

(2.4)

where \(L\) is the total number of layers in the network, and \(\mathcal{P}\) denotes the perturbed edges. Generally, a smaller \(\mathcal{S}_p\) indicates less attention coefficients received by adversarial edges. To further train GNNs such that a lower \(\mathcal{S}_p\) is guaranteed, we design the following loss function to penalize
perturbed edges:

\[ L_{\text{dist}} = -\min \left( \eta, \frac{\mathbb{E}}{e_{ij} \in \mathcal{E} \setminus \mathcal{P}} \sum_{1 \leq l \leq L} a^l_{ij} - \frac{\mathbb{E}}{e_{ij} \in \mathcal{P}} \sum_{1 \leq l \leq L} a^l_{ij} \right), \tag{2.5} \]

where \( \eta \) is a hyper parameter controlling the margin between mean values of two distributions, \( \mathcal{E} \setminus \mathcal{P} \) represents normal edges in the graph, and \( \mathbb{E} \) computes the expectation. Using the expectation of attention coefficients for all normal edges as an anchor, \( L_{\text{dist}} \) aims at reducing the averaged attention coefficient of perturbed edges, until a certain discrepancy of \( \eta \) between these two mean values is satisfied. Note that minimizing \( S_p \) directly instead of \( L_{\text{dist}} \) will lead to unstable attention coefficients, making PA-GNN hard to converge. The expectations of attention coefficients are estimated by their empirical means:

\[ \mathbb{E}_{e_{ij} \in \mathcal{E} \setminus \mathcal{P}}^{a^l_{ij}} = \frac{1}{|\mathcal{E} \setminus \mathcal{P}|} \sum_{l=1}^{L} \sum_{e_{ij} \in \mathcal{E} \setminus \mathcal{P}} a^l_{ij}, \tag{2.6} \]

\[ \mathbb{E}_{e_{ij} \in \mathcal{P}}^{a^l_{ij}} = \frac{1}{|\mathcal{P}|} \sum_{l=1}^{L} \sum_{e_{ij} \in \mathcal{P}} a^l_{ij}, \tag{2.7} \]

where \( |\cdot| \) denotes the cardinality of a set. We combine \( L_{\text{dist}} \) with the original cross-entropy loss \( L_c \) and create the following learning objective for PA-GNN:

\[ \min_{\theta} \mathcal{L} = \min_{\theta} (L_c + \lambda L_{\text{dist}}), \tag{2.8} \]

where \( \lambda \) balances the semi-supervised classification loss and the attention coefficient scores on perturbed edges.

Training PA-GNN with the above objective directly is non-trivial, because it is unlikely to distinguish exact perturbed edges \( \mathcal{P} \) from normal edges in a poisoned graph. However, it is practical to discover vulnerable edges from clean graphs with adversarial attack methods on graphs. For example, \textit{metattack} poisons a clean graph to reduces the performance of GNNs by adding adversarial edges, which can be treated as the set \( \mathcal{P} \). Therefore, we explore clean graphs from domains similar to the poisoned graph. Specifically, as shown in Figure 2.1, we first inject perturbation edges to clean graphs using adversarial attack methods, then leverage those adversarial counterparts to train the ability to penalize perturbed edges. Such ability is further transferred to GNNs on the target graph, so that the robustness is improved. In the following section, we discuss how we transfer the ability to penalize perturbed edges from clean graphs to the target poisoned graph in detail.
2.4.2 Transfer with Meta-Optimization

As discussed above, it is very challenging to train PA-GNN for a poisoned graph because the adversarial edge distribution remains unknown. We turn to exploit clean graphs from similar domains to create adversarial counterparts that serve as supervised knowledge. One simple solution to utilize them is pre-training PA-GNN on clean graphs with perturbations, which formulate the set of adversarial edges $\mathcal{P}$. Then the pre-trained model is fine-tuned on target graph $G$ purely with the node classification objective. However, the performance of pre-training with clean graphs and adversarial edges is rather limited, because graphs have different data distributions, making it difficult to equip GNNs with a generalized ability to discriminate perturbations. Our experimental results in Section 2.5.3 also confirm the above analysis.

In recent years, meta-learning has shown promising results in various applications [87, 107, 123, 126]. The goal of meta-learning is to train a model on a variety of learning tasks, such that it can solve new tasks with a small amount or even no supervision knowledge [42, 30, 125]. Finn et al. [30] propose model-agnostic meta-learning algorithm where the model is trained explicitly such that a small number of gradient steps and few training data from a new task can also produce good generalization performance on that task. This motivates us to train a meta model with a generalized ability to penalize perturbed edges (i.e., assign lower attention coefficients). The meta model serve as the initialization of PA-GNN, and its fast-adaptation capability helps retain such penalizing ability as much as possible on the target poisoned graph. To achieve the goal, we propose a meta-optimization algorithm that trains the initialization of PA-GNN. With manually generated perturbations on clean graphs, PA-GNN receive full supervision and its initialization preserve the penalizing ability. Further fine-tuned model on the poisoned graph $G$ is able to defend adversarial attacks and maintain an excellent performance.

We begin with generating perturbations on clean graphs. State-of-the-art adversarial attack method for graph – metattack [143] is chosen. Let $\mathcal{P}_i$ represent the set of adversarial edges created for clean graph $G_i$. Next, we define learning tasks for the meta-optimization. The learning objective of any task is defined in Equation 2.8, which aims at classifying nodes accurately while assigning low attention coefficient scores to perturbed edges on its corresponding graph. Let $T_i$ denote the specific task for $G_i$. Namely, there are $M$ tasks in accordance with clean graphs. Because clean graphs are specified for every task, we use $\mathcal{L}_{T_i}(\theta)$ to denote the loss function of task $T_i$. We then compile support sets and query sets for learning tasks. Labeled nodes from each clean graph is split into two groups – one for the support set and the other as the query set. Let $S_i$ and $Q_i$ denote the support set and the query set for $G_i$, respectively.

Given $M$ learning tasks, the optimization algorithm first adapts the initial model parameters to every learning task separately. Formally, $\theta$ becomes $\theta_i'$ when adapting to $T_i$. We use
gradient descent to compute the updated model parameter $\theta'_i$. The gradient w.r.t $\theta'_i$ is evaluated using $L_T(\theta)$ on corresponding support set $S_i$, and the initial model parameters $\theta$ are updated as follows:

$$
\theta'_i = \theta - \alpha \nabla_{\theta} L_T(\theta),
$$

(2.9)

where $\alpha$ controls the learning rate. Note that only one gradient step is shown in Equation 2.9, but using multiple gradient updates is a straightforward extension, as suggested by [30]. There are $M$ different versions of the initial model (i.e., $f_{\theta'_1}, \cdots, f_{\theta'_M}$) constructed in accordance with learning tasks.

The model parameters are trained by optimizing for the performance of $f_{\theta'_i}$ with respect to $\theta$ across all tasks. More concretely, we define the following objective function for the meta-optimization:

$$
\min_\theta \sum_{i=1}^{M} L_T(\theta'_i) = \min_\theta \sum_{i=1}^{M} L_T(\theta - \alpha \nabla_{\theta} L_T(\theta)).
$$

(2.10)

Because both classifying nodes and penalizing adversarial edges are considered by the objective of PA-GNN, model parameters will preserve the ability to reduce the negative effects from adversarial attacks while maintaining a high accuracy for the classification. Note that we perform meta-optimization over $\theta$ with the objective computed using the updated model parameters $\theta'_i$ for all tasks. Consequently, model parameters are optimized such that few numbers of gradient steps on a new task will produce maximally effective behavior on that task. The characteristic of fast-adaptation on new tasks would help the model retain the ability to penalize perturbed edges on $G$, which is proved by the experiential results in Section 2.5.3.1. Formally, stochastic gradient descent (SGD) is used to update model parameters $\theta$ cross tasks:

$$
\theta \leftarrow \theta - \beta \nabla_{\theta} \sum_{i=1}^{M} L_T(\theta'_i).
$$

(2.11)

In practice, the above gradients are estimated using labeled nodes from query sets $S_i$ of all tasks. Our empirical results suggest that splitting support sets and query sets on-the-fly through iterations of the meta-optimization improves overall performance. We adopt this strategy for the training procedure of PA-GNN.

**Training Algorithm** An overview of the training procedure of PA-GNN is illustrated in Algorithm 1.
Algorithm 1: The training framework of PA-GNN

**Input:** $G$ and $\{G_1, \ldots, G_M\}$

**Output:** Model parameters $\theta$

1. Randomly initialize $\theta$;
2. for $G_i = G_1, \ldots, G_M$ do
   3. Select perturbed edge set $P_i$ with $\text{metattack}$;
   4. end
   5. while not early-stop do
      6. for $G_i = G_1, \ldots, G_M$ do
         7. Split labeled nodes of $G_i$ into support set $S_i$ and $Q_i$;
         8. Evaluating $\nabla_\theta \mathcal{L}_{T_i}(\theta)$ with $S_i$ and $\mathcal{L}_{T_i}$;
         9. Compute adapted parameters $\theta_i'$ with gradient descent: $\theta_i' \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}_{T_i}(\theta)$;
      10. end
      11. Update $\theta$ on $\{Q_1, \ldots, Q_M\}$ with: $\theta \leftarrow \theta - \beta \nabla_\theta \sum_{i=1}^M \mathcal{L}_{T_i}(\theta_i')$;
   12. end
   13. Fine-tune $\theta$ on $G$ use $\mathcal{L}_c$;

2.5 Experiments

In this section, we conduct experiments to evaluate the effectiveness of PA-GNN. We aim to answer the following questions:

- Can PA-GNN outperform existing robust GNNs under representative and state-of-the-art adversarial attacks on graphs?

- How the penalized aggregation mechanism and the meta-optimization algorithm contribute to PA-GNN?

- How sensitive of PA-GNN on the hyper-parameters?

Next, we start by introducing the experimental settings followed by experiments on node classification to answer these questions.

2.5.1 Experimental Setup

2.5.1.1 Datasets.

To conduct comprehensive studies of PA-GNN, we conduct experiments under two different settings:

- **Same-domain setting:** We sample the poisoned graph and clean graphs from the same data distribution. Two popular benchmark networks (i.e., *Pubmed* [89] and *Reddit* [38]) are selected
as large graphs. *Pubmed* is a citation network where nodes are documents and edges represent citations; *Reddit* is compiled from reddit.com where nodes are threads and edges denote two threads are commented by a same user. Both graphs build nodal features using averaged word embedding vectors [81] of documents/threads. We create desired graphs using sub-graphs of the large graph. Each of them is randomly split into 5 similar-size non-overlapping sub-graphs. One graph is perturbed as the poisoned graph, while the remained ones are used as clean graphs.

- **Similar-domain setting**: We put PA-GNN in real-world settings where graphs come from different scenarios. More concretely, we compile two datasets from Yelp Review\(^1\), which contains point-of-interests (POIs) and user reviews from various cities in Northern American. Firstly, each city in Yelp Review is transferred into a graph, where nodes are POIs, nodal features are averaged word-embedding vector [81] of all reviews that a POI received, and binary labels are created to tell whether corresponding POIs are restaurants. We further define edges using co-reviews (i.e., reviews from the same author). Graphs from different cities have different data distribution because of the differences in tastes, culture, lifestyle, etc. The first dataset (Yelp-Small) contains four middle-scale cities including Cleveland, Madison, Mississauga, and Glendale where Cleveland is perturbed as \(G\). The second dataset (Yelp-Large) contains top-3 largest cities including Charlotte, Phoenix, and Toronto. Specifically, we inject adversarial edges to the graph from Toronto to validate the transferability of PA-GNN because Toronto is a foreign city compared with others.

We itemize statistics of datasets in Table 2.1. We randomly select 10% of nodes for training, 20% for validation and remained for testing on all datasets (i.e., on \(G\)). 40% nodes from each clean graph are selected to build support and query sets, while remained ones are treated as unlabeled. Support sets and query sets are equally split on-the-fly randomly for each iteration of the meta-optimization (i.e., after \(\theta\) is updated) to ensure the maximum performance.

\(^1\)https://www.yelp.com/dataset

<table>
<thead>
<tr>
<th></th>
<th>Pubmed</th>
<th>Reddit</th>
<th>Yelp-Small</th>
<th>Yelp-Large</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Avg. # of nodes</strong></td>
<td>1061</td>
<td>3180</td>
<td>3426</td>
<td>15757</td>
</tr>
<tr>
<td><strong>Avg. # of edges</strong></td>
<td>2614</td>
<td>14950</td>
<td>90431</td>
<td>160893</td>
</tr>
<tr>
<td><strong># of features</strong></td>
<td>500</td>
<td>503</td>
<td>200</td>
<td>25</td>
</tr>
<tr>
<td><strong># of classes</strong></td>
<td>3</td>
<td>7</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>
2.5.1.2 Attack Methods.

To evaluate how robust PA-GNN is under different attack methods and settings, three representative and state-of-the-art adversarial attack methods on graphs are chosen:

- **Non-Targeted Attack:** Non-targeted attack aims at reducing the overall performance of GNNs. We adopt metattack [143] for non-targeted attack, which is also state-of-the-art adversarial attack method on graph data. We increase the perturbation rate (i.e., number of perturbed edges over all normal edges) from 0 to 30%, by a step size of 5% (10% for Yelp-Large dataset due to the high computational cost of metattack). We use the setting with best attack performance according to [143].

- **Targeted Attack:** Targeted attack focuses on misclassifying specific target nodes. nettack [142] is adopted as the targeted attack method. Specifically, we first randomly perturb 500 nodes with nettack on target graph, then randomly assign them to training, validating, and testing sets according to their proportions (i.e., 1:2:7). This creates a realistic setting since not all nodes will be attacked (hacked) in a real-world scenario, and perturbations can happen in training, validating and testing sets. We adopt the original setting for nettack from [142].

- **Random Attack:** Random attack randomly select some node pairs, and flip their connectivity (i.e., remove existing edges and connect non-adjacent nodes). It can be treated as an injecting random noises to a clean graph. The ratio of the number of flipped edges to the number of clean edges varies from 0 to 100% with a step size of 20%.

We evaluate compared methods against state-of-the-art non-targeted attack method metattack on all datasets. We analyze the performances against targeted attack on Reddit and Yelp-Large datasets. For random attack, we compare each method on Pubmed and Yelp-Small datasets as a complementary. Consistent results are observed on remained datasets.

2.5.1.3 Baselines.

We compare PA-GNN with representative and state-of-the-art GNNs and robust GNNs. The details are:

- **GCN** [52]: GCN is a widely used graph neural network. It defines graph convolution via spectral analysis. We adopt the most popular version from [52].

- **GAT** [38]: As introduced in Section 2.2.1, GAT leverages multi-head self-attention to assign different weights to neighborhoods.
Note that PreProcess, RGCN and VPN are state-of-the-art robust GNNs developed to defend against adversarial attacks.
2.5.1.4 Settings and Parameters.

We report the averaged results of 10 runs for all experiments. We deploy a multi-head mechanism [105] to enhance the performance of self-attention. We adopt metattack to generate perturbations on clean graphs. All hyper-parameters are tuned on the validation set to achieve the best performance. For a fair comparison, following a common way [137], we fix the number of layers to 2 and the total number of hidden units per layer to 64 for all compared models. We set $\lambda$ to 1.0 and $\eta$ to 100 for all settings. Parameter sensitivity on $\lambda$ and $\eta$ will be analyzed in Section 2.5.4. We perform 5 gradient steps to estimate $\theta'$ as suggested by [30].

2.5.2 Robustness Comparison

To answer the first question, we evaluate the robustness of PA-GNN under various adversarial attack scenarios with comparison to baseline methods. We adopt semi-supervised node classification as our evaluation task as described in Section 2.5.1.4.

2.5.2.1 Defense Against Non-Targeted Attack.

We first conduct experiments under non-targeted attack on four datasets. Each experiment is conducted 10 times. The average accuracy with standard deviation is reported in Table 2.2. From the table, we make the following observations: (i) As illustrated, the accuracy of vanilla GCN and GAT decays rapidly when the perturbation rate goes higher, while other robust GNN models achieve relatively higher performance in most cases. This suggests the necessity of improving the robustness of GNN models; (ii) The preprocessing-based method shows consistent results on the Pubmed dataset with sparse features. However, it fails for other datasets. Because the feature similarity and neighbor relationship are often complementary, purely relying on feature similarity to determining perturbation edges is not a promising solution. On the contrary, PA-GNN aims at learning the ability to detect and penalizing perturbations from data, which is more dynamic and reliable; (iii) Comparing with RGCN, PA-GNN achieves higher performance under different scenarios. This is because PA-GNN successfully leverages clean graphs for improving the robustness. Moreover, instead of constraining model parameters with Gaussian distributions, PA-GNN directly restricts the attention coefficients of perturbed edges, which is more straightforward. The above observations articulate the efficacy of PA-GNN, which successfully learns to penalize perturbations thanks to the meta-optimization on clean graphs. Lastly, we point out that PA-GNN achieves slightly higher or comparable performance even if $\mathcal{G}$ is clean (i.e., no
Table 2.3: Node classification accuracy under targeted attack.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>GCN</th>
<th>GAT</th>
<th>PreProcess</th>
<th>RGCN</th>
<th>VPN</th>
<th>PA-GNN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reddit</td>
<td>74.25±0.20</td>
<td>73.83±0.12</td>
<td>73.02±0.18</td>
<td>74.75±0.15</td>
<td>74.00±0.07</td>
<td><strong>79.57±0.13</strong></td>
</tr>
<tr>
<td>Yelp-Large</td>
<td>71.97±0.12</td>
<td>71.12±0.73</td>
<td>74.83±0.12</td>
<td>77.01±0.24</td>
<td>72.09±0.73</td>
<td><strong>82.28±0.49</strong></td>
</tr>
</tbody>
</table>

Table 2.4: Node classification accuracy of ablations.

<table>
<thead>
<tr>
<th>Ptb Rate (%)</th>
<th>0</th>
<th>5</th>
<th>10</th>
<th>15</th>
<th>20</th>
<th>25</th>
<th>30</th>
</tr>
</thead>
<tbody>
<tr>
<td>PA-GNN&lt;sub&gt;np&lt;/sub&gt;</td>
<td>95.25±0.81</td>
<td>92.17±0.23</td>
<td>90.45±0.72</td>
<td>88.72±0.61</td>
<td>86.66±0.18</td>
<td>84.68±0.52</td>
<td>81.53±0.34</td>
</tr>
<tr>
<td>PA-GNN&lt;sub&gt;2nd&lt;/sub&gt;</td>
<td>77.11±0.67</td>
<td>75.43±1.11</td>
<td>71.18±1.24</td>
<td>68.51±1.95</td>
<td>64.86±1.59</td>
<td>63.16±1.29</td>
<td>61.08±1.07</td>
</tr>
<tr>
<td>PA-GNN&lt;sub&gt;ft&lt;/sub&gt;</td>
<td><strong>96.72±0.09</strong></td>
<td>91.89±0.14</td>
<td>89.79±0.24</td>
<td>87.56±0.25</td>
<td>85.41±0.17</td>
<td>83.88±0.35</td>
<td>82.14±0.38</td>
</tr>
<tr>
<td>PA-GNN&lt;sub&gt;jt&lt;/sub&gt;</td>
<td>96.63±0.18</td>
<td>92.13±0.19</td>
<td>88.62±0.35</td>
<td>87.00±0.27</td>
<td>84.65±0.25</td>
<td>82.75±0.27</td>
<td>81.20±0.30</td>
</tr>
<tr>
<td>PA-GNN</td>
<td>95.80±0.11</td>
<td><strong>94.35±0.33</strong></td>
<td><strong>92.16±0.49</strong></td>
<td><strong>90.74±0.56</strong></td>
<td><strong>88.44±0.20</strong></td>
<td><strong>86.60±0.17</strong></td>
<td><strong>84.45±0.34</strong></td>
</tr>
</tbody>
</table>

adversarial edges), showing the advantage of the meta-optimization process.

2.5.2.2 Defense Against Targeted Attack

We further study how robust PA-GNN is under targeted attack. As shown in Table 2.3, PA-GNN outperforms all the compared methods under targeted attack, with approximate 5% performance improvements on both datasets compared with second accurate methods. This confirms the reliability of PA-GNN against targeted attack. Moreover, note that the perturbations of clean graphs are generated by metattack, which is a non-target adversarial attack algorithm. We conclude that PA-GNN does not rely on specific adversarial attack algorithm to train model initialization. The ability to penalize perturbation can be generalized to defend other adversarial attacks. A similar conclusion can be drawn from following experiments against random attack.

2.5.2.3 Defense Against Random Attack.

Finally, we evaluate all compared methods against random attack. As shown in Figure 2.2, PA-GNN consistently out-performs all compared methods. Thanks to the meta-optimization process, PA-GNN successfully learns to penalize perturbations, and transfers such ability to target graph with a different kind of perturbation. Besides, the low performance of GAT indicates the vulnerability of the self-attention, which confirms the effectiveness of the proposed penalizing aggregation mechanism.
2.5.3 Ablation Study

To answer the second question, we conduct ablation studies to understand the penalized aggregation and meta-optimization algorithm.

2.5.3.1 Varying the Penalized Aggregation Mechanism.

We analyze the effect of proposed penalized aggregation mechanism from two aspects. Firstly, we propose PA-GNN_{np}, a variant of PA-GNN that removes the penalized aggregation mechanism by setting $\lambda = 0$. We validate PA-GNN_{np} on Reddit dataset, and its performance against different perturbation rates is reported in Table 2.4. As we can see, PA-GNN consistently outperforms PA-GNN_{np} by 2% of accuracy. The penalized aggregation mechanism limits negative effects from perturbed edges, in turns improves the performance on the target graph. Secondly, we explore distributions of attention coefficient on the poisoned graph of PA-GNN with/without the penalized aggregation mechanism. Specifically, the normalized distributions of attention coefficients for normal and perturbed edges are plotted in Figure 2.3. We further report their mean values in Table 2.5. Without the penalized aggregation, perturbed edges obtain relatively higher attention coefficients. This explains how adversarial attacks hurt the aggregation process of a GNN. As shown in Figure 2.3b, normal edges receive relative higher attention coefficients.
through PA-GNN, confirming the ability to penalize perturbations is transferable since PA-GNN is fine-tuned merely with the node classification objective. These observations reaffirm the effectiveness of the penalized aggregation mechanism and the meta-optimization algorithm, which successfully transfers the ability to penalize perturbations in the poisoned graph.

2.5.3.2 Varying the Meta-Optimization Algorithm.

Next, we study the contribution of the meta-optimization algorithm. As discussed in Section 2.4.2, three ablations are created accordingly: PA-GNN$_{2nd}$, PA-GNN$_{ft}$, and PA-GNN$_{jt}$. PA-GNN$_{2nd}$ ignores clean graphs and rely on a second-time attack to generate perturbed edges. PA-GNN$_{ft}$ omit the meta-optimization process, training the model initialization on clean graphs and their adversarial counterparts jointly. We then fine-tune the initialization for $\mathcal{G}$ using the classification loss $L_c$. PA-GNN$_{jt}$ further simplifies PA-GNN$_{ft}$ by adding $\mathcal{G}$ to the joint training step. Note that we remove $L_{dist}$ for $\mathcal{G}$ because detailed perturbation information is unknown for a poisoned graph. All three variants are evaluated on Reddit dataset, and their performance is reported in Table 2.4.

PA-GNN$_{2nd}$ performs the worst among all variations. Because perturbed edges from the adversarial attack can significantly hurt the accuracy, treating them as clean edges is not a feasible solution. PA-GNN$_{ft}$, and PA-GNN$_{jt}$ slightly out-perform PA-GNN when $\mathcal{G}$ is clean. This is not amazing since more training data can contribute to the model. However, their performance decreases rapidly as the perturbation rate raises up. Because the data distribution of a perturbed graph is changed, barely aggregate all available data is not an optimal solution for defending adversarial attack. It is vital to design PA-GNN which leverages clean graphs from similar domains for improving the robustness of GNNs. At last, PA-GNN$_{np}$ consistently out-performs PA-GNN$_{ft}$, and PA-GNN$_{jt}$ in perturbed cases. shown advantages of the meta-optimization algorithm which utilizes clean graphs to train the model regardless of the penalized aggregation
mechanism.

2.5.4 Parameter Sensitivity Analysis

We investigate the sensitivity of $\eta$ and $\lambda$ for PA-GNN. $\eta$ controls the penalty of perturbed edges, while $\lambda$ balances the classification objective and the penalized aggregation mechanism. Generally, a larger $\eta$ pull the distribution of perturbed edges farther away from that of normal edges. We explore the sensitivity on Pubmed and Reddit datasets, both with a 10% perturbation rate. We alter $\eta$ and $\lambda$ among $\{0, 1, 10, 100, 1000\}$ and $\{0, 50, 100, 200, 400, 800\}$, respectively. The performance of PA-GNN is illustrated in Figure 2.4. As we can see, the accuracy of PA-GNN is relatively smooth when parameters are within certain ranges. However, extremely large values of $\eta$ and $\lambda$ result in low performances on both datasets, which should be avoided in practice. Moreover, increasing $\lambda$ from 0 to 1 improves the accuracy on both datasets, demonstrating the proposed penalized aggregation mechanism can improve the robustness of PA-GNN.

![Figure 2.4](image-url)
Chapter 3

Fairness of Graph Neural Networks

Graph Neural Networks show promising results for semi-supervised learning tasks on graphs, thus become favorable comparing with other approaches. Despite the remarkable success of GNNs, it is difficult to train GNNs with insufficient supervision. When labeled data are limited, the performance of GNNs becomes unsatisfying for low-degree nodes, showing biases towards the certain group of individuals in the graph/network. While some prior work analyze successes and failures of GNNs on the entire model level, profiling GNNs on individual node level is still underexplored.

In this chapter, we analyze GNNs in regard to the node degree distribution. Specifically, we choose the graph convolutional networks (GCNs) as the implementation of GNNs. From empirical observation to theoretical proof, we confirm that GCNs are biased towards nodes with larger degrees with higher accuracy on them, even if high-degree nodes are underrepresented in most graphs. We further develop a novel Self-Supervised-Learning Degree-Specific GCN (SL-DSGCN) that mitigates the degree-related biases of GCNs from model and data aspects. Firstly, we propose a degree-specific GCN layer that captures both discrepancies and similarities of nodes with different degrees, which reduces the inner model-aspect biases of GCNs caused by sharing the same parameters with all nodes. Secondly, we design a self-supervised-learning algorithm that creates pseudo labels with uncertainty scores on unlabeled nodes with a Bayesian neural network. Pseudo labels increase the chance of connecting to labeled neighbors for low-degree nodes, thus reducing the biases of GCNs from the data perspective. Uncertainty scores are further exploited to weight pseudo labels dynamically in the stochastic gradient descent for SL-DSGCN. Experiments on three benchmark datasets show SL-DSGCN not only outperforms state-of-the-art self-training/self-supervised-learning GCN methods, but also improves GCN accuracy dramatically for low-degree nodes.
3.1 Introduction

Over last few years, Graph Convolutional Networks (GCNs) have benefited many real world applications across different domains, such as molecule design [130], financial fraud detection [108], traffic prediction [109, 131], and user behavior analysis [102, 60, 43]. One of the most important and challenging applications for GCNs is to classify nodes in a semi-supervised manner. In semi-supervised learning, GCNs recursively update the feature representation of each node by applying node-agnostic transformation parameters. The whole training process is supervised by a few labeled nodes.

However, degree distributions of most real-world graphs (e.g., citation graphs, review graphs, etc.) are power-law [2, 28, 19]. While the degree of major nodes are relatively small, few nodes on the long-tail side can dominate the training/learning of GCNs (we refer to Figure 3.1 in the analysis section as examples). We argue the power-law distributed node degree could hurt the performance of GCNs. On the one hand, nodes on such a graph are not independent and identically distributed (i.i.d), thus the parameters of a GCN should not be shared by all nodes. As suggested by [75], nodes with various degrees play different roles in the graph. Taking social networks as an example, high-degree nodes are usually leaders with higher influence; while most low-degree ones are at the fringes of the network. Current GCNs with node-agnostic parameters overlook the complex relations and roles of nodes with different degrees. On the other hand, the non-i.i.d node degrees can hurt the message-passing mechanism of GCNs. In fact, the superior performance of GCNs relies on the information propagating from labeled nodes to unlabeled nodes [38]. Obviously, nodes with lower degrees are less likely to be connected to labeled neighbors, compared with high-degree ones. As a result, less information are passed to these low-degree nodes, resulting in unsatisfying or even poor prediction performance. Few literature have explored the effects of non-i.i.d node degrees on real-world graphs. Recently, Wu et al. [112] propose a multi-task learning framework for GCNs, where the degree information is encoded into learned node representations. However, simply incorporating the value of degree as an extra feature does not solve the potential biases of GCNs, and low-degree nodes still suffer from the insufficient supervisions.

Therefore, in this chapter, we analyze the degree-related biases in GCNs thoroughly. First, we design a series of observational tests to validate our assumption: the performance of GCNs are not evenly distributed regarding node degrees, and GCNs are biased on low-degree nodes. We further prove that the training of GCNs are more sensitive to nodes with higher degrees using sensitivity analysis and influence functions in statistics [53, 119]. Inspired by the analytic results, we realize two challenges of addressing the degree-related biases in GCNs as follows:

(C1) How to capture the complex relation among nodes with different degrees? We recog-
nize three types of node relations including global shared relation, local intra-relation, and local inter-relation. The global shared relation captures the common property among all nodes in the whole graph (i.e., what GCNs already done); the local intra-relation describes the similarity of nodes with the same degree; and the local inter-relation further characterizes the interacted information from nodes with similar degrees, as they may behave likewise. Therefore, a sufficiently generalized and powerful degree-specific GCN is required, which not only balances the global generalization and local degree customization of different nodes, but also captures local relation among nodes with various degrees; and (C2) **How to provide effective and robust information to facilitate the learning of GCNs on low-degree nodes?** It is non-trivial to make accurate predictions with limited labeled neighbors, due to the biased information propagation. How to create sufficient supervisions for low-degree nodes is extremely challenging.

To address these challenges, in this chapter, we propose a novel Self-Supervised-Learning Degree-Specific GCN (SL-DSGCN), which reduces the biases from non-\textit{i.i.d} node degrees in conventional GCNs. In particular, we first design a degree-specific GNN layer, which considers both globally shared information and local relation among nodes with same degree value. A recurrent neural network (RNN) based parameter generator is designed for modeling the inter-degree relation, which is ignored in the prior work DEMO-Net [112]. We then leverage the massive unlabeled nodes to construct artificial supervisions for low-degree nodes. We propose a self-supervised-learning paradigm where a Bayesian neural network serves as the teacher and assigns pseudo/soft labels jointly with uncertainty scores on unlabeled nodes. We further utilize the uncertainty scores as a guidance in stochastic gradient descent to prevent overfitting inaccurate pseudo labels when training SL-DSGCN. SL-DSGCN is evaluated on three benchmark datasets and show superior performance over state-of-the-art methods. Besides, it reduces label prediction error on low-degree nodes dramatically.

In summary, our contributions are three-fold:

- We study a novel problem of addressing the degree-related biases in GCNs. To the best of our knowledge, we are the first to analyze this problem empirically and theoretically.

- We design SL-DSGCN that tackles the degree-related biases in GCNs from both model and data distribution aspects using the proposed degree-specific GCN layer and self-learning algorithm, correspondingly.

- We validate SL-DSGCN on three benchmark graph datasets and confirm that SL-DSGCN not only out-performs state-of-the-art baselines, but also improves the prediction accuracy on low-degree nodes significantly.
3.2 Related Work

In this section, we review related works for graph neural networks and self-supervised learning.

3.2.1 Graph Convolutional Neural Networks

Graph data are ubiquitous in real-world. Recently, graph convolutional neural networks (GCNNs) have achieved state-of-the-art performance for many graph mining tasks [118, 52, 38], and many efforts have been taken [115, 128, 119, 46, 101, 99]. In general, these GCNNs could be divided into two categorizes: spectral based GCNNs and spatial-based GCNNs. Bruna et al. firstly propose the spectral based GCNNs [11] by applying the spectral filter on the local spectral space according to the spectral graph theory. Following this work, various spectral-based GCNNs [10, 22, 38, 52] are developed to improve the performances. GCN [52] aggregates the neighborhood information from the perspective of spectral theory. With the similar intuition, GraphSAGE [38] extends prior works in the inductive setting. The spectral based GCNNs usually require to compute the Laplacian eigenvectors or the approximated eigenvalues as suggested by spectral theory, and these methods are inefficient on large scale graph. Different from the spectral based ones, to improve the efficiency, the spatial-based GCNNs [4, 106, 133] attempt to directly capture the spatial topological information and use the mini-batch training schema. For example, DCNN [4] combines graph convolutional operator with the diffusion process and Velićković et al. proposes the graph attention network [106] with the self-attention mechanism on the neighbors of the node and assign different weights accordingly during the aggregation process. Of all these GCNNs, GNNs [52] are highly favorable by the computer science community [97, 58] due to the reliable performance. Thus, we select GCNs for this work.

Though GCNs have show promising results, recent advancements [142, 21, 118] also reveal various issues of GCNs including the over-smoothing and the vulnerability. In this chapter, we empirically validate a new issue of GCNNs, i.e., GCNNs are biased towards high-degree nodes and have low accuracy on low-degree ones. A potential reason is the imbalanced labeled node distribution. The issue is amplified when the total amount of labeled node for training is small.

3.2.2 Self-Supervised Learning

Recently, self-supervised learning, which generally refers to explicitly training models with automatically generated labels, has become a successful approach in computer vision and natural language processing for unsupervised pretraining and for addressing the issue of lacking labeled data [47]. For example, pretext tasks such as Image Inpainting [80] and Image Jigsaw Puzzle [78] are widely adopted in computer vision domains.
The success of self-supervision has motivated its study in graph mining domains. Though still in its early stage, there are a few seminal work trying to exploit self-supervised training to improve the performance of GCNs [45]. For example, Li et al. [58] propose the co-training and self-training based GCN models by expanding the training node set with pseudo labels from its nearest neighborhoods; Sun et al. [97] combine DeepCluster [13] with a multi-stage training framework so that the generalization performance of GCNs with few labeled nodes are improved.

Despite their initial success, existing studies mainly utilize self-supervised training as a trick for GCNs, without digging deep into why self-supervised training can improve the performance and what kind of nodes are benefited most from the self-supervised training. Our work is inherently different from existing ones on self-supervised GCNs. The lack of labeled neighborhoods among low-degree nodes motivate us to explore self-supervised training to balance the label distribution. The proposed self-supervision based one teacher-student network is also different from existing work. In addition, we also address the issue from the perspective of degree-specific layers.

To the best of our knowledge, only few work address the degree non-i.i.d sampled problem. DEMO-Net [112] learn the degree-specific representation for each node via the explicitly designed hash table. This work is significantly different from ours. Besides, it fails to capture the similarity of nodes with close degree values, where the RNN-based parameter generator in SL-DSGCN is able to do so.

3.2.3 Fairness on Graph

As fairness of machine learning is gaining more and more attention, some pioneering approaches have been proposed to address the discrimination and biases in graph-structured data [31, 48]. For example, Bose and Hamilton [9] propose compositional fairness constraints to learn unbiased node embedding vectors for graph with protected node attributes, such as gender and race; Rahman et al. [84] extend the popular graph embedding method node2vec [36] to protect the sensitive attributes by diversifying neighborhood representations; Dai and Wang [20] introduce adversarial training and group level fairness constraints to train GNNs with less discrimination over underrepresented groups. However, none of the above methods consider and tackle the biases of GNNs from the degree distribution prospective.

3.3 Preliminaries

We use $G = (V, E, X)$ to denote a graph, where $V = \{v_1, \ldots, v_N\}$ is the set of $N$ nodes, $E \subseteq V \times V$ represents the set of edges, and $X = \{x_1, \ldots, x_N\}$ indicates node features. We
use $d_i \in \mathbb{R}^+$ to denote the degree of node $v_i$. In semi-supervised setting, partial nodes come with labels and are denoted as $\mathcal{V}^L$, where the corresponding label of node $v_i$ is $y_i$. Similarly, the unlabeled part is defined as $\mathcal{V}^U$.

We introduce the architecture of a GCN. A GCN contains multiple layers. Each layer transforms its input node features to another Euclidean space as output. Different from fully-connected layers, a GCN layer takes first-order neighbors’ information into consideration when transforming the feature vector of a node. This “message-passing” mechanism ensures the initial features of any two nodes can affect each other even if they are faraway neighbors, along with the network going deeper. We use $x^l_v$ to denote the learned representation of node $v$ from the $l$-th layer in a GNN ($l = 1, \cdots, L$). Specifically, $x^0_v = x_v$. The output node features of the $l$-th layer, which also formulate the input to the next layer, are generated as follows:

$$x^{l+1}_i = \sigma\left(\sum_{v_j \in \mathcal{N}(i)} \frac{1}{\sqrt{d_i \cdot d_j}} W^l x^l_j\right), \quad (3.1)$$

where $\mathcal{N}(i)$ denotes the immediate neighbor nodes of $v_i$ and $\sigma$ is the activation function (e.g., ReLU).

We take node classification as an example task for the rest of the chapter, without loss of generality. The objective of training GNNs is to minimize the following cross-entropy loss function:

$$\mathcal{L} = \sum_{v_i \in \mathcal{V}^L} \mathcal{L}(v_i) = -\sum_{v_i \in \mathcal{V}^L} y_i \log \hat{y}_i, \quad (3.2)$$

where $y_i$ and $\hat{y}_i$ are true and predicted labels, respectively. Typically, $\hat{y}_i = \text{Softmax}(x^{(L)}_v)$ is acquired by applying Softmax to the representations from the last layer.

### 3.4 Data Analysis

In this section, we conduct preliminary analysis on real-world graphs to show the properties of real-world graphs for semi-supervised node classification and the issue of GCNs on these datasets. The preliminary analysis lays a solid foundation and paves us a way to design better GCNs. Since we aim to discover the issue of GCNs on real-world datasets, we choose four widely used datasets from GCNs literature to perform the analysis, which includes Cora, Citeseer, Pubmed [52], and Reddit [38]. Note that the split of training, validation and testing on all datasets are the same as described in the cited papers.
3.4.1 Degree Distribution

The degree distribution of most real-world graphs follows the power-law [28, 2]. To verify this, we plot the degree distribution of the four datasets in Figure 3.1. As we can see from the figure, degrees of the majority of nodes are relatively low, and decrease as the value of degree raise up. The shape of the degree distributions verify our assumption. The power-law distribution indicates nodes on graph are non-i.i.d distributed. Applying the same network parameters on all nodes may result in sub-optimal prediction/classification.

3.4.2 Accuracy Varying Node Degree

GCNs rely on message-passing mechanism, and aggregates the information from neighbors to learn representative embedding vectors. Because the degree of nodes follows a nonuniform (power-law) distribution, low-degree nodes, which are the majority, will receive less information during the aggregation. As a results, the error rate on low-degree nodes could be higher. To validate the assumption, we train GCNs following the same setting in [52], and report its error rate on node classification tasks w.r.t degree of nodes. From Figure 3.2, we find that, when degree is small, the error rate decreases significantly as the degree of nodes becomes larger. This verify our assumption that low-degree nodes receive less information during the aggregation and GCNs is biased against low-degree nodes.
3.4.3 Labeled Neighbor Distribution

To further understand how the non-uniform degree distribution hurts GCNs, we analyze the probability of being connected to any labeled neighbor w.r.t node degree, as illustrated in Figure 3.3. We can conclude that nodes with higher degrees are much more likely to own labeled neighbors comparing with lower degree ones. In training process, GCNs use back-propagation to update its neural parameters such that the classification error on labeled nodes is reduced. Thanks to the message-passing mechanism, nodes with labeled neighbors participate more frequently in the optimization process. As a result, GCNs performs better on high-degree nodes.

3.4.4 Bridging Node Degree and Biases in GCNs

Inspired by Koh and Liang [53] and Xu et al. [119], we borrow ideas of sensitivity analysis and influence functions in statistics field to measure the influence of a specific node to the accuracy of GCNs. We first define node influence from node $v_i$ to $v_k$ as follows:

$$I(i, k) = \| \mathbb{E}(\partial x_i^L / \partial x_k) \|,$$

which measures how the feature of $v_i$ affects the training of GCN on node $v_k$. Because the loss function is defined purely on labeled nodes, the influence of any unlabeled node (say $v_i$) to the
whole GCN can be approximated by the overall influence of every labeled node:

$$S(i) = \sum_{v_k \in V^L} I(i, k).$$  \hspace{1cm} (3.4)$$

We can summarize the relation of node degree and the performance of GCNs in the following theorem:

**Theorem 1.** Assume ReLU is the activation function. Let $v_i$ and $v_j$ denote two nodes in a graph. If we have $d_i > d_j$, then the influence score follows: $S(i) > S(j)$ of an untrained GCN.

**Proof.** The partial differential between $x_i^l$ and $x_k$ is derived as:

$$\frac{\partial x_i^l}{\partial x_k} = \frac{1}{\sqrt{d_i}} \cdot \text{diag}(1_{\sigma_l}) \cdot W^k \cdot \sum_{v_n \in N(i)} \frac{1}{\sqrt{d_n}} \frac{\partial x_n^{l-1}}{\partial x_k},$$

where $\sigma_l$ denote the output from the activation function (i.e. ReLU) at the $l$-th GCN layer, and $\text{diag}(1_{\sigma_l})$ is a diagonal mask matrix representing the activation result. Using chain rule, we further derive:

$$\frac{\partial x_i^L}{\partial x_k} = \sqrt{d_i d_k} \cdot \prod_{p=1}^{0} \frac{1}{d_p} \text{diag}(1_{\sigma_l}) \cdot W^l,$$

(3.5)
where $\Psi$ is the set of all $(L+1)$-length random-walk paths on the graph from node $v_i$ to $v_k$, and $p^l$ represents the $l$-th node on a specific path $p$ ($p^L$ and $p^0$ denote node $i$ and $k$, accordingly). Note that every path is fully-connected where $v_p^l \in \mathcal{N}(p^{l+1})$ for any $p$ and any $l$. Similar to Xu et al. [119], the expectation of $\frac{\partial x^L_i}{\partial x_k}$ can be estimated as follows:

$$
\mathbb{E} \left( \frac{\partial x^L_i}{\partial x_k} \right) = \sqrt{d_id_k} \cdot \sum_{p=1}^{\Psi_n} \mathbb{E} \left( \prod_{l=L}^{0} \frac{1}{d_p^l} \text{diag}(1_{\sigma_l}) \cdot W^l \right) = \rho \sum_{v_n \in \mathcal{N}(i)} \sum_{p=1}^{\Psi_n} \mathbb{E} \left( \prod_{l=L-1}^{0} \frac{1}{d_p^l} \text{diag}(1_{\sigma_l}) \cdot W^l \right),
$$

where $\rho = \frac{(\sqrt{d_k}/\sqrt{d_i}) \cdot \text{diag}(1_{\sigma_L}) \cdot W^L}$ only correlated to $v_i$ and $v_k$, and $\Psi_n$ denote the set of all $L$-length walks from a neighborhood of $v_i$ to $v_k$. Assume the neighborhoods are randomly distributed (i.e., $v_n$ is (near) randomly sampled), the expectation on walks starting from neighborhoods can be replaced by a constant value $\nu$:

$$
\sum_{p=1}^{\Psi_n} \mathbb{E} \left( \prod_{l=L-1}^{0} \frac{1}{d_p^l} \text{diag}(1_{\sigma_l}) \cdot W^l \right) = \nu,
$$

and we further have:

$$
\mathbb{E} \left( \frac{\partial x^L_i}{\partial x_k} \right) = \rho d_i \nu = \nu \sqrt{d_kd_i} \cdot \text{diag}(1_{\sigma_L}) \cdot W^L \propto \sqrt{d_i},
$$

therefore, if $d_i > d_j$, then we have $\mathbb{E} \left( \frac{\partial x^L_i}{\partial x_k} \right) > \mathbb{E} \left( \frac{\partial x^L_j}{\partial x_k} \right)$. By summing up over all labeled nodes in $\mathcal{V}^L$, we have $S(i) > S(j)$.

We validate our conclusion in Figure 3.4.

We first visualize the influence score distribution on a subgraph of the Cora dataset in Figure 3.4a. Clearly, the hub node at the center of the graph gains a much higher influence score than others. We further analyze the distribution of the influence score on four datasets, and report the results in Figure 3.4b. Clearly, the influence score increases as the node degree becomes larger. This indicates that nodes with larger degrees have higher impact on the training process of GCN, resulting in imbalanced error rate distribution over different degrees.
(a) Topology of influence score on a subgraph of Cora. Darker colors denote higher influences.

(b) Distribution of influence score varying node degree.

Figure 3.4: Distributions of the Influence Score.

3.5 Approach

With the above analysis, we summarize the limitations of GCNs as follows: (1) GCNs use the same set of parameters for all nodes and fails to model the local intra- and inter- relations of nodes, resulting in model-aspect biases; (2) low degree nodes are less likely to have labeled neighbors and participate inactively when training GCNs, such biases come from the data distribution aspect. To address these issues, we propose SL-DSGCN that improves GCNs from two folds: *Firstly*, we propose a degree-specific GCN (DSGCN) layer whose parameters are generated by a recurrent neural network (RNN). Nodes with different degrees have their own specific parameters so that the local intra-relation is captured. Besides, as parameters are iteratively gen-
erated from the same RNN, their inner correlations help model the inter-relation of nodes with similar degrees. The DSGCN layer balances the global generalization and local discrepancies for nodes with various degrees. Secondly, we design a self-supervised-learning algorithm to construct pseudo labels with uncertainty within unlabeled nodes. This is achieved by training a Bayesian neural network (BNN). The DSGCN is fine-tuned on both true and pseudo labels, where the artificial ones are weighted according to their uncertainties. This prevents SL-DSGCN from overfitting to inaccurate pseudo labels.

3.5.1 Degree-Specific GCN Layer

As the training of a GCN is dominated by high-degree nodes, using one set of parameters could lead to sub-optimal results. To increase the diversity of learned parameters for nodes with different degrees, following aggregation can be used to distinguish the degree-specific information from the graph:

$$x_{l+1}^i = \sigma \left( \sum_{j \in \mathcal{N}(i)} a_{ij} (W^l + W^l_{d(j)}) x^l_j \right), \quad (3.8)$$

where $W^l_{d(j)}$ captures degree-specific information. $W^l$ is the original GNN parameters at layer $l$ in Eqn 3.1.

The design of $W^l_{d(j)}$ is a non-trivial task. One straightforward way is making degree-specific parameters unique for all degrees. However, the maximum value of node degree on a graph can be extremely large due to the long-tail power-law distribution, constructing unique parameters for every degree is impractical. Besides, some higher degrees are underrepresented, with only few nodes available. How to prevent underfitting issue for them is also a challenging problem. To overcome this issue, Wu et al. [112] propose a hashing-based solution where some degrees are mapped to the same entry of a hash table containing multiple sets of GCN parameters. By manually tuning the size of the hash table, the total number of degree-specific parameters is under control.

However, the hashing-based approach randomly maps node degree to parameters, and ignores the local inter-relations of nodes with similar degrees. If two nodes have close degree values, their may have a tight correlation. The necessity of capturing local inter-relation of nodes motivates us to adopt an RNN to generate the degree-specific parameters, which is shown in Figure 3.5. Specifically, let $W^l_0$ denote the initialization input to an RNN cell $\text{RNN}(\cdot)$, degree-specific parameters are generated as follows:

$$W^l_{k+1} = \text{RNN}(W^l_k), \quad k = 0, 1, \ldots, d_{\text{max}}, \quad (3.9)$$
where $W^l_{k+1}$ is the updated hidden state of the RNN after feeding $W^l_k$ as the input, and $d_{\text{max}}$ is a threshold to prevent long-tail issue of the degrees. Nodes with degree higher than $d_{\text{max}}$ are processed using $W^l_{\text{max}+1}$. The generated parameters can cover every degree. The advantages of using an RNN are (1) as RNN is iterating over all degrees, generated degree-specified parameters are correlated with each other corresponding to the degree so that modeling local inter-relations of nodes is guaranteed; (2) the total number of actual trainable parameters is fixed (i.e., the initialization input and parameters in the RNN cell), which is more efficient comparing with setting up every $W^l_{d(i)}$ separately or use a hashing table. Note that the generated parameters from RNN naturally capture the local intra-relation because every degree has its unique parameters. Besides, the shared parameters $W^l$ handles the globally shared node relations.

While the DSGCN layer reduces degree-related biases in GCNs from the model aspect, low-degree nodes still participate less frequently when training the DSGCN. To provide sufficient supervisions for low-degree nodes, we introduce a self-supervised-learning algorithm that creates high-quality pseudo-labels on unlabeled nodes.

### 3.5.2 Self-Supervised-Training with Bayesian Teacher Network

In most semi-supervised settings on graph data, the number of unlabeled nodes is much larger than that of labeled ones (i.e., $|V^L| \ll |V^U|$). We assume the existence of a graph annotator that can heuristically generate pseudo-labels for nodes in $V^U$, such as propagation algorithm [138], label spreading [135], and PairWalks [113]. The pseudo-labels are noisy and less accurate compared with the true labels from $V^L$ because of the limitations of the annotator. The intuition of proposed self-learning algorithm is to leverage the large amount of pseudo-labels in the training of GCNs so that even for low-degree nodes can have labeled neighbors. However, different from existing literature [97, 58] that use pseudo labels in the same way of true labeled nodes, we also
judge the quality of pseudo labels to avoid overfitting on inaccurate pseudo labels. Specifically, we design a Bayesian neural network as a teacher to justify the quality of pseudo-labels from the annotator, so that the GCNs as a student can fully exploit the pseudo-labels. There are two steps of the self-learning process as illustrated in Figure 3.6.

3.5.2.1 Pre-training with the Annotator

Firstly, we build the student network using the proposed degree-specific GNN layer. As shown in Figure 3.6a, the student first applies multiple DSGCN layers over the input graph ($\psi(\cdot)$ part) to capture the dependencies of graph structure and to model the correlation among nodes with different degrees. Taking the graph $G$ as an input, $\psi(\cdot)$ transform each node into its representation vector. To further classify each node, we then apply fully-connected layers followed by a softmax layer ($\phi(\cdot)$ part) on representation vectors from $\psi(\cdot)$. Different from conventional GNNs, the student network leverage $\psi(\cdot)$ to learn data representation from the graph, and assign
the classification task to the second part $\phi(\cdot)$. Using the pseudo labels from the annotator, we pre-train the student network so that $\psi(\cdot)$ is fitted to the data and $\phi(\cdot)$ becomes a noisy classifier. The whole student network is represented by $\phi(\psi(\cdot))$.

However, simply treating all pseudo labels as ground truth will hurt the performance. We then design a teacher network to estimate the uncertainty of pseudo labels from the annotator. The teacher network is constructed based on a Bayesian neural network (BNN) [23]. We use the node representation from the data representation learner $\psi(\cdot)$ as the input, to train a fully-connected BNN using real-world truely labeled nodes $\mathcal{V}^L$, as illustrated in Figure 3.6a. In particular, the BNN aims at learning the posterior distribution of its parameters, defined as follows:

$$p(\zeta|\psi(x)) \propto p(\psi(x)|\zeta) \cdot p(\zeta), \quad (3.10)$$

where $\zeta$ denotes the parameters of the BNN, $p(\zeta)$ is the prior of $\zeta$ that contains our assumption of the network parameters, and $p(\psi(x)|\zeta)$ is the likelihood which describe the input data (i.e., node representation from $\psi(x)$). The probability distributions of model parameters $\zeta$ are updated with the Bayes theorem taking into account both the prior and the likelihood. Without loss of generalities, we use normal distribution as the prior for the BNN. We fix the representation learner when updating the BNN part, so that the teacher can leverage the knowledge from the annotated results. Besides, training on top of $\psi(\cdot)$ ensures the teacher is learning in the same representation space of the student, so that the judgement of unlabeled nodes in further steps is unbiased and has no domain shifting for the student network. We use a two-layer fully-connected network as the approximation for the likelihood. The posterior mean $\mu$ and posterior covariance $\kappa$ of the BNN is acquired after training the BNN model, and are further used to create soft labels on unlabeled nodes with uncertainties. In particular, for every unlabeled node $v_i \in \mathcal{V}^U$, we acquire its prediction and uncertainty score as follows:

$$y^s_i = f(\mu(x_i)), \quad c_i = g(\kappa(x_i)),$$

where $f(\cdot)$ and $g(\cdot)$ are two functions (e.g., neural networks) that map the posterior mean and covariance vectors to desired soft label and uncertainty score.

We visualize the prediction and uncertainty of the teacher BNN trained on a small subset from the reddit network dataset in Figure 3.7. As we can see in Figure 3.7a, the uncertainty for labeled nodes are almost zero, indicating the teacher fit the training data very well. Meanwhile, we also observe that the uncertainty scores on low degree nodes tend to be larger, which is consistent with our previous analysis. As low degree nodes have less impact on the training loss function and receive less supervision from labeled neighbors, it is harder to generate a
(a) Uncertainty scores from the teacher network. Darker color means higher uncertainty and “⊤” denotes training nodes.

(b) Classification error of the teacher network. Red and green denote wrong and correct prediction respectively, and black represents training nodes.

Figure 3.7: Uncertainty score and error distribution of the teacher network. Generally, nodes closer to labeled (training) ones tend to have lower uncertainty and error rate.

confident prediction for them. Similarly in Figure 3.7b, it is more likely for low degree nodes to be misclassified than high degree ones.

3.5.2.2 Fine-tuning Student with Uncertainty Scores.

After the pre-training of student and teacher network, the second step of the self-learning process is fine-tuning the student network using generated labels and uncertainty scores from the teacher. We define a softly-labeled node set $V^S \subset V^U$ where nodes in $V^S$ are labeled identically by both the annotator and the teacher. The intuition is similar to majority vote. Given large amount of unlabeled nodes, it is worthwhile to compile a cleaner labeled node set as a compensation to the existing true labeled nodes.

Existing works exploring self-learning for GNNs treat selected pseudo labels in the same way of using labeled nodes. For example, Li et al. [58] and Sun et al. [97] progressively add selected nodes with pseudo labels into the training set. However, such solutions are sub-optimal. One bottleneck is that all selected pseudo labels are equally treated, and are utilized in the same way of true labeled nodes. However, even for pseudo labels with high confidence, they still contain more noise than the real labeled part.

Fortunately, the proposed BNN-based teacher network naturally solves the above challenge. The generated uncertainty scores can be utilized when training with pseudo labels. Specifically, we fine-tune the student network on $V^{LS} = V^L \cup V^S$ using stochastic gradient descent (SGD) algorithm, where the uncertainty score controls the step size for each nodes in $V^{LS}$. We use $\theta$ to
denote parameters in the student network, the optimization (learning) goal is as follows:

$$\theta^* = \arg\max_{\theta} L(\theta) = \sum_{v_i \in V^{LS}} L(v_i; \theta).$$  \hspace{1cm} (3.11)$$

The updating rule for parameters $\theta$ is:

$$\theta' = \theta - \sum_{v_i \in V^{LS}} \eta_i L(v_i; \theta),$$  \hspace{1cm} (3.12)$$

where $\eta_i$ is a dynamic step size defined as follows:

$$\eta_i = \eta \cdot \eta_i^c \cdot \eta_i^d = \eta \cdot \exp(-\alpha c_i) \cdot \exp(\beta d_i),$$  \hspace{1cm} (3.13)$$

which contains three parts. The first part $\eta$ is the global step size used in classic SGD. The second part $\eta_i^c$ penalize each sample (node) by its quality, using the uncertainty score acquired from the teacher network. We choose a negative exponential function over the uncertainty score so that nodes with larger uncertainty participate less in the updating process. The third term empirically assigns larger weights to nodes with higher degrees according to the observations in Figure 3.4a and Figure 3.7. Here $\alpha$ and $\beta$ are hyperparameters that balance three parts in the dynamic step size. Generally, larger values of $\alpha$ and/or $\beta$ pay more attention to the uncertainty scores and the degree distribution, correspondingly. After fine-tining on $V^{LS}$ using SGD with dynamic step size, we use the student network to predict node labels.

### 3.5.3 Training Algorithm

We summarize the self-learning process in Algorithm 2. Line 1-3 are the pre-training of student and teacher network. After acquiring predictions and uncertainty scores from the pre-trained teacher in Line 4, we compile $V^{LS}$ using true labels and the softly-labeled nodes (Line 5-6). Finally, as introduced in Line 7-9, the student network is fine-tuned on $V^{LS}$ with dynamic step size. Note that although we select GCN as the basis of SL-DMSGCN, the idea of capturing globally shared, local intra- and inter- relations of nodes with an RNN-based parameter generator, and using self-supervised-learning with dynamic step size are model agnostic. Namely, they can also be applied on other GNN models, such as graph attention networks [106], GraphSAGE [38], etc. We leave this part for future work.
Algorithm 2: Self-learning for SL-DSGCN

\begin{algorithm}
\textbf{Input:} $G = (V, E, X)$
\textbf{Output:} Parameters $\theta$ of student network $\phi(\psi(\cdot))$

// Pre-training
1. Acquire pseudo-labels for $V^U$ using a graph annotator;
2. Pre-train $\phi(\psi(\cdot))$ on pseudo labels;
3. Fix $\psi(\cdot)$ and pre-train BNN part of the teacher network;
4. Acquire prediction $y^a_i$ and uncertainty score $c_i$ for every node in $V^U$ from the teacher;

// Fine-tuning
5. Compile a soft-labeled node set $V^S \subset V^U$ where the teacher network agrees with the annotator;
6. Build $V^{LS} = V^L \cup V^S$ to fine-tune the student network;
7. \textbf{while} not converge \textbf{do}
\phantom{1.}8. Compute dynamic step size $\eta_i$ for $v_i \in V^{LS}$ as $\eta_i = \eta \cdot \eta_i^c \cdot \eta_i^d$,
\phantom{1.}9. Update parameters of the student network as $\theta' = \theta - \sum_{v_i \in V^{LS}} \eta_i \mathcal{L}(v_i; \theta)$;
8. \textbf{end}
\end{algorithm}

3.6 Experiments

In this section, we conduct experiments on real-world datasets to evaluate the effectiveness of SL-DSGCN. In particular, we aim to answer the following questions:

- Can SL-DSGCN outperform existing self-training algorithms for GNNs on various benchmark datasets?

- How do the degree-specific design (DSGCN), the machine teaching approach, and the dynamic step size contribute to SL-DSGCN?

- How sensitive of SL-DSGCN is on the selection of softly-labeled node set?

Next, we start by introducing the experimental settings followed by experiments on node classification to answer these questions.

3.6.1 Experimental Setup

3.6.1.1 Datasets

For a fair comparison, we adopt same benchmark datasets used by Sun et al. [97] and Li et al. [58], including Cora, Citeseer, Pubmed [89]. Each dataset contains a citation graph, where nodes represent articles/papers and edges denote citation correlation. Node features are constructed using bag-of-words features. The detailed statistics of the datasets are summarized in Table 3.1.
### Table 3.1: Statistics of the Datasets

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Nodes</th>
<th>Edges</th>
<th>Classes</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cora</td>
<td>2708</td>
<td>5429</td>
<td>7</td>
<td>1433</td>
</tr>
<tr>
<td>CiteSeer</td>
<td>3327</td>
<td>4732</td>
<td>6</td>
<td>3703</td>
</tr>
<tr>
<td>PubMed</td>
<td>19717</td>
<td>44338</td>
<td>3</td>
<td>500</td>
</tr>
</tbody>
</table>

### Table 3.2: Node Classification Performance Comparison on Cora, CiteSeer and PubMed

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Cora</th>
<th>CiteSeer</th>
<th>PubMed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Label Rate</td>
<td>0.5%</td>
<td>1%</td>
<td>2%</td>
</tr>
<tr>
<td>LP</td>
<td>29.05</td>
<td>38.63</td>
<td>53.26</td>
</tr>
<tr>
<td>ParWalks</td>
<td>37.01</td>
<td>41.40</td>
<td>50.84</td>
</tr>
<tr>
<td>GCN</td>
<td>35.89</td>
<td>46.00</td>
<td>60.00</td>
</tr>
<tr>
<td>DEMO-Net</td>
<td>33.56</td>
<td>40.05</td>
<td>61.18</td>
</tr>
<tr>
<td>Self-Train</td>
<td>43.83</td>
<td>52.45</td>
<td>63.36</td>
</tr>
<tr>
<td>Co-Train</td>
<td>40.99</td>
<td>52.08</td>
<td>64.27</td>
</tr>
<tr>
<td>Union</td>
<td>45.86</td>
<td>53.59</td>
<td>64.86</td>
</tr>
<tr>
<td>Intersection</td>
<td>33.38</td>
<td>49.26</td>
<td>62.58</td>
</tr>
<tr>
<td>M3S</td>
<td>50.28</td>
<td>58.74</td>
<td>68.04</td>
</tr>
<tr>
<td>SL-DSGCN</td>
<td>53.58</td>
<td>61.36</td>
<td>70.31</td>
</tr>
</tbody>
</table>

#### 3.6.1.2 Baselines

We compare SL-DSGCN with representative and state-of-the-art node classification algorithms, which includes:

- **LP [138]:** Label Propagation is a classical self-supervised learning algorithm which where we iteratively assign labels to unlabelled points by propagating labels through the graph. It serves as the weak annotator in our framework.

- **ParWalks [113]:** ParWalks extends label propagation by using partially absorbing random walk.

- **GCN [52]:** GCN is a widely used graph neural network. It defines graph convolution via spectral analysis.

- **DEMO-Net [112]:** It proposes multi-task graph convolution where each task represents node representation learning for nodes with a specific degree value, thus leading to preserving the degree specific graph structure. DEMO-net also contains other constraints to improve the representation learning, including order-free and seed-oriented. These constraints are removed
for a fair comparison because they do not tackle the degree-related biases of GCNs, and can be applied on all above methods. We choose the weight version of DEMO-net due to better performances.

• Co-Training [58]: This method uses the ParWalk to find the most confident vertices – the nearest neighbors to the labeled vertices of each class, and then add them to the label set to train a GCN.

• Self-Training, Union and Intersection [58]: Self-training picks the most confident soft-labels of GCN and puts it into the labeled node set to improve the performance of GCN. Union takes the union of the most confident soft-labels by both GCN and ParWalk as self-supervision while Intersection takes the intersection of the two as the self-supervision.

• M3S [97]: Multi-Stage Self-Supervised Training leverages DeepCluster technique to provide self-supervision and utilizes the cluster information to iterative train GNN.

3.6.1.3 Settings and Hyperparameters

The training and testing set are generated as follows: we randomly sample $x\%$ of nodes for training, 35\% nodes for testing, and treat the remained nodes as unlabeled ones for each dataset. Furthermore, to understand how SL-DSGCN performs under various label sparsity scenarios in real-world, for CORA and Citeseer, we vary $x$ as $\{0.5, 1, 2, 3, 4\}$. Since PubMed is relative larger than Cora and CiteSeer, we vary $x$ as $\{0.03, 0.06, 0.09\}$ for it. Note that we set $x$ as small values because in typical setting of real-world semi-supervised node classification tasks, only a small amount of nodes are labeled for training [97, 58]. We adopt the same hyper-parameters for GCN as introduced by Kipf and Welling [52], which is a two-layer GCN with 16 hidden units on each layer. For DEMO-Net, Self-Train, Co-train, Union, and Intersection, we adopt their public code and tune hyperparameters for the best performance. We implement M3S following the descriptions in the paper [97]. For the student network part, both $\phi(\cdot)$ and $\psi(\cdot)$ are implemented by one DSGCN layer. We set $d_{\text{max}}$ to 10. The Bayesian neural network part of the teacher contains two fully-connected layers, each contains 16 hidden units. We fix $\alpha$ and $\beta$ to 1. Note that for fair comparison, we set all self-supervised-learning GCNs to two-layers with 16 hidden units, which is aligned with both GCN and SL-DSGCN. We report the averaged results over 10 times of running.
3.6.2 Node Classification Performance

To answer the first research question, we conduct node classification with comparison to existing self-training algorithms for GNNs on the datasets introduced above. The experimental results in terms of accuracy for the three datasets are reported in Table 3.2. From the table, we make the following observations:

• Generally, self-supervision based approaches such as M3S, Intersection and Union outperform algorithms without self-supervision such as LP and GCN, which implies that self-supervision could help provide more labeled nodes to training so that the percentage of labeled neighborhood of low-degree increases.

• As label rate $x$ increases, the performance improvement of self-supervision based approaches over non-self-supervision approaches decreases. For example, on Cora dataset, as $x$ increase from 0.5% to 4%, the performance improvement of M3S and SL-DGNN over GCN are $\{14.39, 12.74, 8.04, 3.94, 3.12\}$ and $\{17.69, 15.36, 10.31, 9.00, 5.37\}$, respectively. This is because as the amount of labeled data increases, the percentage of labeled neighborhood of low-degree also increases, which makes the introduction of self-supervision less useful.

• For all the three datasets and label rate, SL-DSGCN consistently outperforms all the baselines significantly, which shows the effectiveness of the proposed framework. In particular, both M3S and SL-DSGCN adopt self-supervision. SL-DSGCN significantly outperforms M3S because SL-DSGCN explicitly model degree-specific GNN layer through LSTM, which could benefit the low-degree nodes more.

3.6.3 Performance on Low Degree Nodes

SL-DSGCN is motivated by the observation that the number of labeled nodes for low-degree nodes is very much smaller than that of high-degree nodes, which makes GNN biased towards high-degree nodes. Thus, degree specific GNN layer and self-training with Bayesian teacher networks are leveraged to alleviate the issue. To validate the effectiveness of the proposed framework SL-DSGCN on low-degree nodes, we further visualize the node classification performance of low-degree nodes on Cora and Citeseer in Figure 3.8. Note that for Cora and Citeseer, 96.45% and 97.53% nodes have a degree less than 11. From the figure, we observe that:

• Both DSGCN and SL-DSGCN outperform GNN significantly, especially on node with small degrees, which shows the effectiveness of degree specific layer and self-supervision for improving performance of low-degree nodes. In addition, SL-DSGCN has better performance than DSGCN, which implies that the degree specific layer and self-supervision improves the
performance from two different perspectives. Degree-specific layer tries to learn node-specific parameters to reduce the bias towards high-degree nodes while self-supervision tries to improve the number labeled nodes in each node’s neighborhood.

- When degree the node degree is very small, say \{1, 2, 3, 4, 5\}, the improvement of DSGCN and SL-DSGCN is very significant. As the degree become larger, the improvement becomes smaller. This is because when degree is very small, most of these nodes have very few labeled nodes in their neighborhood. A small amount of soft-label and the degree-specific parameters could improve the performance a lot. However, when the degree become larger, there are already enough supervision to train a good GNN, which makes the improvement insignificant. However, as the majority nodes in graphs are low degree nodes, SL-DSGCN can still improve the overall performance significantly.

### 3.6.4 Ablation Study

In this subsection, we conduct ablation study to understand the impact of degree-specific GNN, the dynamic step size for SGD, and the self-teaching algorithm, which answers the second research question. Specifically, several variations of SL-DSGCN are compared including (1): DSGCN which applies the degree-specific parameters on GCN; (2) MT-GNN which replace the dynamic step size with original one and remove the softly-labeled node set from \(\gamma^{LS}\) (i.e., only use the labeled nodes for fine-tuning the student network). MT-GNN can be treated as a GNN enhanced by the vanilla machine teaching algorithm; (3) SL-DSGCN\(_f\) which removes the dynamic step size; and (4) SL-GNN which removes the degree-specific design in the student network. The performance of SL-DSGCN and the variants on Cora and Citeseer are reported in Table 3.3 and 3.4, respectively. From these two tables, we observe that: (i) In terms of the com-
Table 3.3: Ablation study on Cora dataset.

<table>
<thead>
<tr>
<th>Label Rate</th>
<th>0.5%</th>
<th>1%</th>
<th>2%</th>
<th>3%</th>
<th>4%</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSGCN</td>
<td>36.11</td>
<td>47.67</td>
<td>61.91</td>
<td>73.87</td>
<td>77.03</td>
</tr>
<tr>
<td>MT-GNN</td>
<td>50.51</td>
<td>57.47</td>
<td>67.26</td>
<td>78.52</td>
<td>78.84</td>
</tr>
<tr>
<td>SL-DSGCN&lt;sub&gt;fs&lt;/sub&gt;</td>
<td>51.36</td>
<td>59.85</td>
<td>68.81</td>
<td>79.14</td>
<td>79.90</td>
</tr>
<tr>
<td>SL-GNN</td>
<td>52.05</td>
<td>60.41</td>
<td>69.51</td>
<td>79.75</td>
<td>80.21</td>
</tr>
<tr>
<td>SL-DSGCN</td>
<td>53.58</td>
<td>61.36</td>
<td>70.31</td>
<td>80.15</td>
<td>81.05</td>
</tr>
</tbody>
</table>

Table 3.4: Ablation study on Citeseer dataset.

<table>
<thead>
<tr>
<th>Label Rate</th>
<th>0.5%</th>
<th>1%</th>
<th>2%</th>
<th>3%</th>
<th>4%</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSGCN</td>
<td>37.51</td>
<td>44.75</td>
<td>55.41</td>
<td>56.9</td>
<td>60.24</td>
</tr>
<tr>
<td>MT-GNN</td>
<td>49.78</td>
<td>50.75</td>
<td>55.14</td>
<td>59.01</td>
<td>61.23</td>
</tr>
<tr>
<td>SL-DSGCN&lt;sub&gt;fs&lt;/sub&gt;</td>
<td>51.89</td>
<td>53.26</td>
<td>58.38</td>
<td>60.63</td>
<td>62.15</td>
</tr>
<tr>
<td>SL-GNN</td>
<td>52.77</td>
<td>54.79</td>
<td>57.27</td>
<td>61.98</td>
<td>63.99</td>
</tr>
<tr>
<td>SL-DSGCN</td>
<td>54.07</td>
<td>56.68</td>
<td>59.93</td>
<td>62.20</td>
<td>64.45</td>
</tr>
</tbody>
</table>

Table 3.5: Influence of the softly-labeled node set.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Node set</th>
<th>0.5%</th>
<th>1%</th>
<th>2%</th>
<th>3%</th>
<th>4%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cora</td>
<td>DSGCN</td>
<td>36.11</td>
<td>47.67</td>
<td>61.91</td>
<td>73.87</td>
<td>77.03</td>
</tr>
<tr>
<td></td>
<td>V&lt;sub&gt;S&lt;/sub&gt;</td>
<td>47.21</td>
<td>55.10</td>
<td>67.15</td>
<td>76.39</td>
<td>75.07</td>
</tr>
<tr>
<td></td>
<td>V&lt;sub&gt;A&lt;/sub&gt;</td>
<td>50.73</td>
<td>58.29</td>
<td>68.85</td>
<td>77.24</td>
<td>76.93</td>
</tr>
<tr>
<td></td>
<td>SL-DGSCN</td>
<td>53.58</td>
<td>61.36</td>
<td>70.31</td>
<td>80.15</td>
<td>81.05</td>
</tr>
<tr>
<td>Citeseer</td>
<td>DSGCN</td>
<td>37.51</td>
<td>44.75</td>
<td>55.41</td>
<td>56.9</td>
<td>60.24</td>
</tr>
<tr>
<td></td>
<td>V&lt;sub&gt;S&lt;/sub&gt;</td>
<td>50.68</td>
<td>53.42</td>
<td>57.10</td>
<td>60.52</td>
<td>60.63</td>
</tr>
<tr>
<td></td>
<td>V&lt;sub&gt;A&lt;/sub&gt;</td>
<td>52.25</td>
<td>52.80</td>
<td>55.13</td>
<td>61.82</td>
<td>61.01</td>
</tr>
<tr>
<td></td>
<td>SL-DGSCN</td>
<td>54.07</td>
<td>56.68</td>
<td>59.93</td>
<td>62.20</td>
<td>64.45</td>
</tr>
</tbody>
</table>

Comparison between SL-GNN and SL-DSGCN, SL-DSGCN performs slightly better than SL-GNN, which shows that degree specific layer can slightly improve the performance; (ii) In terms of the comparison between SL-DSGCN<sub>fs</sub> and SL-DSGCN, SL-DSGCN has better performance than SL-DSGCN<sub>fs</sub>, which is because SL-DSGCN<sub>fs</sub> doesn’t adopt the dynamic step size; and (iii) SL-DSGCN significantly outperforms DSGCN, which shows the effectiveness of the proposed self-supervised training.
3.6.5 Sensitivity on Softly-labeled Node Set

In this subsection, we further analyze how the construction of softly-labeled node set can impact the performance of SL-DSGCN. We compare the intersection approach in SL-DSGCN with the following alternations: (1) using pseudo labels from the annotator and build $V^S_A$ for all unlabeled nodes; (2) using predictions from the teacher network and compile $V^S_T$ for all unlabeled nodes; and (3) without adding any soft labels, which is actually DSGCN. The node classification performance of SL-DSGCN with comparison to the three alternatives is reported in Table 3.5. From the table, we make the following observations: (i) Compared with training without soft-labels, i.e., trained on $V^L$ only, using soft-labels, i.e., $V^S_A$, $V^S_T$ and $V^S$, can significantly improve the performance, which shows the importance of soft-labels in providing supervision to GNN for classification; and (ii) Though $V^S_A$, $V^S_T$ and $V^S$ all utilize soft-labels, the performance of $V^S$ is much better than $V^S_A$ and $V^S_T$, which indicates that the teacher network and the annotator may infer some wrongly labeled nodes that could negatively affect the performance. Taking the intersection of these two can help pick nodes with correct soft labels and improve the performance.
Chapter 4

Explainability of Graph Neural Networks

With the rapid growth and prevalence of social network applications (Apps) in recent years, understanding user engagement has become increasingly important, to provide useful insights for future App design and development. While various promising GNN approaches were recently pioneered for accurate prediction, their black-box designs are unfortunately limited in model explainability. In this chapter, we explore explainable GNNs for user engagement prediction in social network Apps. First, we propose a flexible definition of user engagement for various business scenarios, based on future metric expectations. Next, we design an end-to-end neural framework, FATE, which incorporates three key factors that we identify to influence user engagement, namely friendships, user actions, and temporal dynamics to achieve explainable engagement predictions. FATE is based on a self-explainable tensor-based GNN, LSTM and a mixture attention mechanism, which allows for (a) predictive explanations based on learned weights across different feature categories, (b) reduced network complexity, and (c) improved performance in both prediction accuracy and training/inference time. We conduct extensive experiments on two large-scale datasets from Snapchat, where FATE outperforms state-of-the-art approaches by $\approx 10\%$ error and $\approx 20\%$ runtime reduction. We also evaluate explanations from FATE, showing strong quantitative and qualitative performance.

4.1 Introduction

With rapid recent developments in web and mobile infrastructure, social networks and applications (Apps) such as Snapchat and Facebook have risen to prominence. The first priority of
development of most social Apps is to attract and maintain a large userbase. Understanding user engagement plays an important role for retaining and activating users. Prior studies try to understand the return of existing users using different metrics, such as churn rate prediction [121] and lifespan analysis [122]. Others model user engagement with macroscopic features (e.g., demographic information) [3] and historical statistic features (e.g., user activities) [66]. Recently, Liu et al. [67] propose using dynamic action graphs, where nodes are in-App actions, and edges are transitions between actions, to predict future activity using a neural model.

Despite some success, existing methods generally suffer from the following: (1) They fail to model friendship dependencies or ignore user-user interactions when modeling user engagement. As users are connected in social Apps, their engagement affects each other [96]. For example, active users may keep posting new contents, which attract his/her friends and elevate their engagement. Thus, it is essential to capture friendship dependencies and user interactions when modeling user engagement. (2) Engagement objectives may differ across Apps and even across features. For example, an advertising team may target prediction of click-through-rate, while a growth-focused team may care about usage trends in different in-App functions. Therefore, the definition of user engagement must be flexible to satisfy different scenarios. (3) Existing methods focus on the predicting user engagement accurately, but fail to answer why a user engages (or not). Explaining user engagement is especially desirable, since it provides valuable insights to practitioners on user priorities and informs mechanism and intervention design for managing different factors motivating different users’ engagement. However, to our knowledge, there are no explainable models for understanding user engagement.

To tackle the aforementioned limitations, we aim to use three key factors: friendship, in-App user actions, and temporal dynamics, to derive explanations for user engagement. Firstly, since users do not engage in a vacuum, but rather with each other, we consider friendships to be key in engagement. For example, many users may be drawn to use an App because of their family and friends’ continued use. Secondly, user actions dictate how a user uses different in-App features, and hints at their reasons for using the App. Thirdly, user behavior changes over time, and often obey temporal periodicity [79]. Incorporating periodicity and recency effects can improve predictive performance.

In this work, we first propose measurement of user engagement based on the expectation of metric(s) of interests in the future, which flexibly handles different business scenarios. Next, we formulate a prediction task to forecast engagement score, based on heterogeneous features identified from friendship structure, user actions, and temporal dynamics. Finally, to accurately predict future engagement while also obtaining meaningful explanations, we propose an end-to-end neural model called FATE (Friendship, Action and Temporal Explanations). In particular,
our model is powered by (a) a friendship module which uses a tensor-based graph convolutional network to capture the influence of network structure and user interactions, and (b) a tensor-based LSTM [37] to model temporal dynamics while also capturing exclusive information from different user actions. FATE’s tensor-based design not only improves explainability aspects by deriving both local (user-level) and global (App-level) importance vectors for each of the three factors using attention and Expectation-Maximization, but is also more efficient compared to classical versions. We show that FATE significantly outperforms existing methods in both accuracy and runtime on two large-scale real-world datasets collected from Snapchat, while also deriving high-quality explanations. To summarize, our contributions are:

- We study the novel problem of explainable user engagement prediction for social network applications;
- We design a flexible definition for user engagement satisfying different business scenarios;
- We propose an end-to-end self-explainable neural framework, FATE, to jointly predict user engagement scores and derive explanations for friendships, user actions, and temporal dynamics from both local and global perspectives; and
- We evaluate FATE on two real-world datasets from Snapchat, showing \( \approx 10\% \) error reduction and \( \approx 20\% \) runtime improvement against state-of-the-art approaches.

### 4.2 Related Work

#### 4.2.1 User Behaviour Modeling

Various prior studies model user behaviours for social network Apps. Typical objectives include churn rate prediction, return rate analysis, intent prediction, etc [5, 50, 49, 6, 68, 121, 54, 67] and anomaly detection [90, 56, 91]. Conventional approaches rely on feature-based models to predict user behaviours. They usually apply learning methods on handcrafted features. For example, Kapoor et al.[49] introduces a hazard based prediction model to predict user return time from the perspective of survival analysis; Lo et al.[68] extract long-term and short-term signals from user activities to predict purchase intent; Trouleau et al.[104] introduce a statistical mixture model for viewer consumption behavior prediction based on video playback data. Recently, neural models have shown promising results in many areas such as computer vision and natural language processing, and have been successfully applied for user modeling tasks [26, 121, 67]. Yang et al.[121] utilize LSTMs [41] to predict churn rate based on historical user activities. Liu et al.[67] introduce a GNN-LSTM model to analyze user engagement, where GNNs are applied
on user action graphs, and an LSTM is used to capture temporal dynamics. *Although these neural methods show superior performance, their black-box designs hinder interpretability, making them unable to summarize the reasons for their predictions, even when their inputs are meaningful user activities features.*

### 4.2.2 Explainable Machine Learning

Explainable machine learning has gain increasing attention in recent years [34]. We overview recent research on explainable GNN/RNN models, as they relate to our model design. We group existing solutions into two categories. The first category focuses on post-hoc interpretation for trained deep neural networks. One kind of model-agnostic approach learns approximations around the predictions, such as linear proxy model [86] and decision trees [88, 139]. Recently, Ying et al.[129] introduce a post-hoc explainable graph neural network to analyze correlations between graph topology, node attributes and predicted labels by optimizing a compact subgraph structure indicating important nodes and edges. *However, post-analyzing interpretations are computationally inefficient, making it difficult to deploy on large systems. Besides, these methods do not help predictive performance.* The second group leverages attention methods to generate explanations on-the-fly, and gained tremendous popularity due to their efficiency [120, 37, 18, 82, 95]. For example, Pope et al.[82] extend explainability methods for convolutional neural networks (CNNs) to cover GNNs; Guo et al.[37] propose an interpretable LSTM architecture that distinguishes the contribution of different input variables to the prediction. *Despite these attention methods successfully provides useful explanations, they are typically designed for one specific deep learning architecture (e.g., LSTMs or CNNs). How to provide attentive explanations for hierarchical deep learning frameworks with heterogeneous input is yet under-explored.*

### 4.3 Preliminaries

First, we define notations for a general social network App. We begin with the *user* as the base unit of an App. Each user represents a registered individual. We use $u$ to denote a user. We split the whole time period (e.g., two weeks) into equal-length continuous *time intervals*. The length of time intervals can vary from hours to days. The past $T$ time intervals in chronological order are denoted as $1, 2, \ldots, T$. Users are connected by *friendship*, which is an undirected relationship. Namely, if $u$ is a friend of $v$, $v$ is also a friend of $u$. Note that friendship is time aware, users can add new friends or remove existing friends at any given time. Users can also use multiple in-App features, like posting a video, chatting with a friend, or liking a post on Facebook; we
call these various user actions. We use a time-aware feature vector to represent the user action for each specific user. A typical feature of social network Apps is in-App communication. By sending and receiving messages, photos, and videos, users share information and influence each other. We call these user interactions.

**User graph**: To jointly model user activities and social network structures, we define a temporal user graph for every user at time $t$ as $G_t^u = (V_t^u, E_t^u, X_t^u, E_t^u)$. Here $V_t^u = \{u\} \cup \mathcal{N}_t(u)$ denotes the nodes in $G_t^u$, where $\mathcal{N}_t(u)$ is a group of users related to $u$, the set of edges $E_t^u$ represents friendships, nodal features $X_t^u$ characterize user actions, and features on edges $E_t^u$ describe user interactions. Note that we split nodal features into $K$ categories, so that each category of features is aligned with a specific user action, respectively. Thus, both the topological structure and the features of user graphs are temporal. In particular, for any given node $u$, its feature vector (i.e., a row of $X_t$) is represented by $x_t^u = [x_{t,1}^u, \ldots, x_{t,K}^u]$, where $x_{t,k}^u \in \mathbb{R}^{d_k}$ is the $k$-th category of features, and $[\cdot]$ denotes concatenation alongside the row. There are many ways to define the graph structure. One example of selecting $G$ is based on ego-networks, as shown in Figure 4.1; here, $\mathcal{N}_t(u)$ is the set of friends of $u$, which reduces the size of graph sharply compared to using the whole social network. Each individual can take different actions in every time interval to control and use in-App functions.

**Defining user engagement**: Because of the dynamism of user activities, social network structure, and the development of the App itself, the user engagement definition should be specified for every user and every time interval. Besides, the primary focus of user engagement varies widely depending on the specific business scenario. For example, Facebook may utilize login frequency to measure engagement, while Snapchat may use the number of messages sent. Thus, user engagement requires a flexible definition which can meet different needs. To tackle above challenges, we define user engagement score using the expectation of a metric of interest in the future, as: $e_t^u = \mathbb{E}(M(u, \tau) | \tau \in [t, t + \Delta t])$, where $M$ is the metric of interest, and $\Delta t$ denotes...
Figure 4.2: Overall framework of FATE: tGCN-based friendship modules capture local network structure and user interactions at each timestep, and tLSTM captures temporal dynamics for distinct user actions. Finally, an attention mixture mechanism governs user engagement prediction.

Explaining user engagement: We identify three key factors that highly impact the user engagement, including user action, temporal dynamics, and friendship. The interpretation is to derive importance/influence of these three factors for user engagement. In particular, we aim at interpreting user engagement from both local (i.e., for individual users) and global (i.e., for the whole group of people, or even the entire App) perspectives. The local interpretations for individual users are formulated as following vectors: (1) User action importance $A_u^u \in \mathbb{R}_{\geq 0}^K$, $\sum_{k=1}^K A_{uk}^u = 1$, which assigns each user action a score that reflects its contribution to user engagement. (2) Temporal importance $T^u \in \mathbb{R}_{\geq 0}^{T \times K}$, $\sum_{t=1}^T T_{tk}^u = 1$ for $k = 1, \cdots, K$, which identifies the importance of user actions over every time interval for the engagement; (3) Friendship importance $F^u \in \mathbb{R}_{\geq 0}^{(T \times N_t(u))}$, $\sum_{v \in N_t(u)} F^u_{tv} = 1$ for $t = 1, \cdots, T$, which characterizes the contributions of friends to user engagement of $u$ over time. For user action and temporal dynamics, we also derive explanations from a global view since they are shared by all users. Specifically, we formulate (1) global user action importance $A^* \in \mathbb{R}_{\geq 0}^K$, $\sum_{k=1}^K A^*_k = 1$ and (2) global temporal importance $T^* \in \mathbb{R}_{\geq 0}^{T \times K}$, $\sum_{t=1}^T T^*_{tk} = 1$ for $k = 1, \cdots, K$. Compared to local explanations which help understand individual user behaviors, global explanations inform overall App-level user behaviors.
We pose the following problem formalization:

**Problem (Explainable Engagement Prediction).** Build a framework that (a) for every user \( u \), predicts the engagement score \( e_u^T \) with explanations \( A_u^T \), \( T_u^T \) and \( F_u^T \) based on the historical user graphs \( G_u^1, \ldots, G_u^T \), and (b) generates global explanations \( A^* \) and \( T^* \).

### 4.4 Our Approach: FATE

We next introduce our proposed approach for explainable engagement prediction, FATE. Firstly, FATE leverages specific designed friendship modules (bottom of Figure 4.2) to model the non-linear social network correlations and user interactions from user graphs of a given user as input. The friendship modules aggregate user graphs and generate representations for user graphs accordingly. These graph representations preserve exclusive information for every time interval and every user action. Next, a temporal module based on tensor-based LSTM [37] (tLSTM, middle part of Figure 4.2) is utilized to capture temporal correlations from graph representations. Finally, a mixture of attention mechanisms (top of Figure 4.2) is deployed to govern the prediction of user engagement based on the output of tLSTM, while also jointly deriving importance vectors as explanations. An illustration of the framework is given in Figure 4.2. We discuss FATE in detail in the following text.

#### 4.4.1 Friendship Module

As shown in Figure 4.3, the goal of the friendship module is to model the non-linear correlation of social network structure and user interactions in every user graph \( G_u^T \). Naturally, graph neural networks (GNNs) [72, 70, 101, 46] can be applied to capture the dependencies of users. We choose the popular graph convolutional networks (GCNs) [52] as our base GNN model. A GCN takes a graph as input, and encodes each node into an embedding vector. The embedding for
each node is updated using its neighbor information on each layer of a GCN as:

\[ \tilde{x}_u = \sigma \left( \sum_{v \in \mathcal{N}(u)} x_v W \right), \quad (4.1) \]

where \( x \) and \( \tilde{x} \) denote input feature and output embedding of the layer, respectively, \( W \) is a feature transformation matrix, and \( \sigma(\cdot) \) denotes a non-linear activation.

However, adopting vanilla GCN in our case is not ideal, because matrix multiplication in GCN mixes all features together. It is difficult to distinguish the importance of input features by looking at the output of a GCN layer. To tackle this limitation, we propose a tensor-based GCN (tGCN), which uses a tensor of learnable parameters. The updating rule of one tGCN layer is:

\[ \tilde{x}_u = \sigma \left( \sum_{v \in \mathcal{N}(u)} x_v \otimes W \right), \quad (4.2) \]

where \( \mathcal{W} = \{W_1, \ldots, W_K\}, W_k \in \mathbb{R}^{d_k \times d'} \), is a set of \( K \) parameter matrices corresponding to each group of features, and \( x_v \otimes W = [x_v^1 W_1, \ldots, x_v^K W_K] \in \mathbb{R}^{K \times d'} \), maps each category of features from the input to the output space separately (as illustrated by different matrices in the middle part of Figure 4.3). Note that each element (e.g. row) of the hidden matrix in a tGCN layer encapsulates information exclusively from a certain category of the input, so that the following mixture attention can distinguish the importance of different user actions and mix exclusive information to improve prediction accuracy. A tGCN layer can be treated as multiple parallel vanilla GCN layers, where each layer is corresponding to one category of features that characterizes one user action. Given a user graph input, We adopt a two-layer tGCN to encode the friendship dependencies into node embedding:

\[ \tilde{X} = \sigma \left( \hat{A} \sigma \left( \hat{X} \otimes \mathcal{W}_0 \right) \otimes \mathcal{W}_1 \right), \quad (4.3) \]

where \( \hat{A} \) is the symmetric normalized adjacency matrix derived from the input user graph, \( X \) are nodal features, and \( \mathcal{W}_s \) are parameters. As input features describe user actions, their exclusive information is preserved in the output of tGCN as \( \tilde{X} = [\tilde{X}_1, \ldots, \tilde{X}_K] \in \mathbb{R}^{K \times d' \times (|\mathcal{N}(u)|+1)} \), which will be used later for generating engagement predictions and explanations.

The learned node embedding vectors from the tGCN can be aggregated as a representation for the graph, such as using mean-pooling to average embedding vectors on all nodes. However, there is a significant disadvantage to such simple solution: namely, the closeness of friends is ignored. In reality, most users only have a few close friends; users with many friends may only
frequently engage with one or few of them. To validate, we compute the friend communication rate of all Snapchat users from a selected city (obscured for privacy reasons). Specifically, we compute the percentage of friends that a user has directly communicated (Chat/Snap) with at least once in a two-week span. As Figure 4.4 shows, most users mainly communicate with a small percentage (10-20\%) of their friends, and don’t frequently contact the remaining ones. Therefore, friendship activeness is key in precisely modeling the closeness of users. To this end, we propose a friendship attention mechanism \cite{105} to quantify the importance of each friend. Formally, a normalized attention score is assigned for each friend $v \in \mathcal{N}(u)$:

$$
\alpha_v = \frac{\exp(\phi(\tilde{x}^u \oplus e^u))}{\sum_{\nu \in \mathcal{N}(u)} \exp(\phi(\tilde{x}^\nu \oplus e^\nu))},
$$

(4.4)

where $\tilde{x}^u$ is the embedding vector of node $v$ from the tensor-based GCN, $e^u$ is the edge feature on edge between $u$ and $v$, $\oplus$ denotes concatenation, and $\phi(\cdot)$ is a mapping function (e.g., a feed-forward neural network). Both user actions (preserved by node embedding vectors) and user interactions are considered by the friendship attention mechanism. To obtain graph representations, we first get the averaged embedding from all friend users weighted by the friendship attention score:

$$
\hat{x} = \sum_{v \in \mathcal{N}(u)} \alpha_v \tilde{x}^v.
$$

(4.5)

Then we concatenate it with the embedding vectors on node $u$ alongside each feature category to get the graph embedding:

$$
g^u = \tilde{x}^u \oplus \hat{x} = [\tilde{x}_1^u \oplus \hat{x}_1, \cdots, \tilde{x}_K^u \oplus \hat{x}_K],
$$

(4.6)

as shown in the right part of Figure 4.3. Note that $\tilde{x}_k^u \oplus \hat{x}_k$ is specifically learned from user action.
4.4.2 Temporal Module

As user activities and interactions evolve over time, modeling its temporal dynamics is a key factor of an accurate prediction for user engagement. Inspired by the success of prior studies for modeling sequential behavior data [67, 121, 124, 103] with recurrent neural networks, we utilize LSTM [41] to capture the evolution of dynamic user graphs. Specifically, we adopt tLSTM following Guo et al.[37]. Mathematically, the transformation at each layer of the tLSTM is as follows:

\[
\begin{align*}
    f_t &= \sigma \left( g_{t,u}^u \otimes U_f + h_{t-1} \otimes U_f^b + b_f \right), \\
    i_t &= \sigma \left( g_{t,u}^u \otimes U_i + h_{t-1} \otimes U_i^b + b_i \right), \\
    o_t &= \sigma \left( g_{t,u}^u \otimes U_o + h_{t-1} \otimes U_o^b + b_o \right), \\
    c_t &= f_t \odot c_{t-1} + i_t \odot \tanh \left( g_{t,u}^u \otimes U_c + h_{t-1} \otimes U_c^b + b_c \right), \\
    h_t &= o_t \odot \tanh (c_t),
\end{align*}
\]

where \( \odot \) denotes element-wise multiplication, \( U_f, U_f^b \) and \( b_f \) are parameters. Similar to tGCN, tLSTM can also be considered as a set of parallelized LSTMs, where each LSTM is responsible for a specific feature group corresponding to its user action. Because the input graph embedding vectors \( g_{1,u}^u, \cdots, g_{T,u}^u \) to tLSTM are specific to each feature category (user action), tLSTM can capture the exclusive temporal dependencies of each user action separately. Similar to \( \mathbf{x} \), we define the hidden states of tLSTM as \( h_t = [h_{t,1}, \cdots, h_{t,K}] \) where \( h_{t,k} \) is exclusively learned for user action \( k \). We further use the hidden states to generate the engagement scores.

4.4.3 User Engagement Score Generation

As aforementioned, user action, temporal dynamics, and friendship are key factors to characterize and predict user engagement. We introduce three latent variables as \( z_A, z_J, z_I \) to represent different user actions (feature category), time intervals, and friends, respectively so that we can distinguish the influence of specific actions, time intervals, and friends. For example, different friends may contribute unequally to user engagement; and certain in-App functions could have higher contributions. Introducing latent variables also bridges the gap between learning
explanations and predicting engagement. The desired explanations are importance vectors that constrain the posteriors of latent variables, and further govern the generating of user engagement scores (introduced in Section 4.4.4). Specifically, FATE generates user engagement predictions as follows:

\[
p(e_T|\{G_s\}) = \sum_{k=1}^{K} \sum_{t=1}^{T} \sum_{v=1}^{|N(u)|} p(e_T, z_A = k, z_J = t, z_I = v|\{G_s\}) \\
= \sum_{k=1}^{K} \sum_{t=1}^{T} \sum_{v=1}^{|N(u)|} \left( \prod_{v} p(z_I = v|z_J = t, z_A = k, \{h_{s,k}\}) \cdot p(z_J = t|z_A = k, \{h_{s,k}\}) \cdot \prod_{h_{s,k}} p(z_A = k|\{h_s\}) \right) \\
\] (4.8)

where \(\{h_s\}\) denotes \(\{h_1 \ldots h_T\}\), and \(\{h_{s,k}\}\) denotes \(\{h_{1,k} \ldots h_{T,k}\}\). The joint probability distribution is further estimated using the conditional probability of latent variables \(z_I, z_J, z_A\), which characterize how user engagement scores are affected by the friendship, temporal dynamics, and user actions accordingly. We keep designing FATE in accordance with the generation process in Eqn. 4.8. In particular, node embeddings are first computed exclusively for every friend, time interval, and user action with proposed tGCN. Next, friendship attention \(p(z_I = v|z_J = t, z_A = k, G_t)\) is estimated using Eqn. 4.4. The summation over \(v\) in Eqn. 4.8 is derived by graph representations from friendship modules. Then tLSTM encapsulates temporal dynamics of graph representation. The conditional probability of \(z_J\) is given as a temporal attention over \(\{h_{s,k}\}\):

\[
\beta_{t,k} = p(z_J = t|z_A = k, \{h_{s,k}\}) = \frac{\exp(\phi_k(h_{t,k}))}{\sum_{\tau=1}^{T} \exp(\phi_k(h_{\tau,k}))}, \\
\] (4.9)

where \(\phi_k(\cdot)\) is a neural network function specified for user action type \(k\). Using temporal attention, each user action is represented by its exclusive summarization over all past time intervals as

\[
a_k = \sum_{t=1}^{T} \beta_{t,k} h_{t,k} \\
\] (4.10)

Finally, we approximate \(p(z_A = k|\{h_s\})\) as the user action attention with another softmax function:

\[
p(z_A = k|\{h_s\}) = \frac{\exp(\phi(a_k \oplus h_{T,k}))}{\sum_{\kappa=1}^{K} \exp(\phi(a_{\kappa} \oplus h_{T,k}))}, \\
\] (4.11)
where $\phi(\cdot)$ is parameterized by a neural network.

To approximate the summation over all time intervals $t = 1, \cdots, T$ in Eqn. 4.8, we use Gaussian distributions to estimate the contribution of every user action to user engagement. Specifically, we use $N(\mu_k, sd_k) = \psi_k(a_k \oplus h_{T,k})$ to parameterize the Gaussian distribution for user action $k$. Here $\psi_k(\cdot)$ is also a neural network. By integrating over all user actions, the user engagement score is derived as:

$$p(e_T) = \sum_{k=1}^{K} N(\mu_k, sd_k) \cdot p(z_A = k | \{h_u\}). \quad (4.12)$$

4.4.4 Explainable User Engagement

To interpret the predicted user engagement, FATE learns the importance vectors as explanations. Similar to many previous studies (e.g., [83, 18, 120, 37]), the local explanations for individual users are directly derived from proposed mixture attentions. Specifically, the friendship attention, temporal attention and user action attention are acquired as importance vectors for friendship, temporal and user action, respectively. Because the computation of these attention scores are included by FATE, it takes no extra cost to derive local explanations. Local explanations reflect specific characteristics and preferences for individual users, which can change dynamically for certain users.

However, local explanations could only help us understand user engagement from individual level. Taking user action as an example, the distribution of its importance vector could vary a lot among different users (see experiments in Section 4.5.5.1 as an example). Because some functions of the App cannot be personalized for every user, it is necessary to interpret their contributions from a global view. For example, when distributing a new feature in an A/B test, it is more reasonable to understand the impact of the feature globally. Under such circumstances, we formulate the global interpretation of user engagement as a learning problem, where the global importance vectors are jointly learned with the model. Taking the global importance vector for user action $A^*$ as an example, we adopt the Expectation–Maximization (EM) method to learn $A^*$ jointly with the optimization of model parameters $\theta$:

$$\mathcal{L}(\theta, A^*) = -\sum_{u \in S} \mathbb{E}_{q_A^u} [\log p(e_T | z_A^u; \{G_u^v\})]$$

$$- \mathbb{E}_{q_A^u} [\log p(z_A^u | \{h_u^v\})] - \mathbb{E}_{q_A^u} [\log p(z_A^u | A^*)], \quad (4.13)$$

where the summation $\sum$ is applied over all training samples $S$, and $q_A^u$ denotes the posterior
distribution for $z_A^u$:

$$q_A^u = p(z_A^u | \{G_s^u\}, e_T^u, \theta) \propto p(e_T^u | z_A^u, \{G_s^u\}) \cdot p(z_A^u | \{G_s^u\})$$

$$\approx p(e_T^u | z_A^u, q_{u,k} \oplus h_{T,k}) \cdot p(z_A^u | \{h_s^u\}). \quad (4.14)$$

The last term in Eqn. 4.13 serves as a regularization term over the posterior of $z_A^u$. Note that the posterior of $z_A^u$ governs the user action attention. Consequently, the regularization term encourages the action importance vectors of individual users to follow the global pattern parameterized by $A^*$. Moreover, we can derive the following closed-form solution of $A^*$ as:

$$A^* = \frac{1}{|S|} \sum_{u \in S} q_A^u, \quad (4.15)$$

which takes both user action attention and the prediction of user engagement into consideration. The learning of user action importance relies on the estimation of posterior $q_A^u$. During training stage, network parameters $\theta$ and the posterior $q_A^u$ are estimated alternatively. Namely, we first freeze all parameters $\theta$ to evaluate $q_A^u$ over the batch of samples, then use the updated $q_A^u$ with gradient descent to update $\theta$ by minimizing 4.13. Similarly for the global temporal importance, we derive the following closed-form solution:

$$T_{t,k}^* = \frac{1}{|S|} \sum_{u \in S} \beta_{t,k}. \quad (4.16)$$

### 4.4.5 Complexity Analysis

The proposed tGCN and adopted tLSTM [37] are more efficient than their vanilla versions. Specifically, we have:

**Theorem 2.** Let $d_{in}$ and $d_{out}$ denote input and output dimensions of a layer. The tensor-based designs for GCN and LSTM reduce network complexity by $(1 - 1/K)d_{in} \cdot d_{out}$ and $4(1 - 1/K)(d_{in} + d_{out})d_{out}$ trainable parameters, and reduce the computational complexity by $O(d_{in} \cdot d_{out})$ and $O((d_{in} + d_{out})d_{out})$, respectively.

**Proof.** We provide the proof in Supplementary 4.6.1.1. \[\square\]

As a result, the proposed designs accelerate the training and inference of FATE, and produce a more compact model. Supplementary 4.6.1.2 shows that FATE’s tensor-based design reduces training and inference time by $\approx 20\%$ compared to using the vanilla version (GCN/LSTM).
4.5 Evaluation

In this section, we aim to answer the following research questions:

- **RQ1**: Can FATE outperform state-of-the-art alternatives in the user engagement prediction task?
- **RQ2**: How does each part/module in FATE affect performance?
- **RQ3**: Can FATE derive meaningful explanations for friendships, user actions, and temporal dynamics?
- **RQ4**: Can FATE flexibly model different engagement metrics?

4.5.1 Datasets and Experiment Setup

We obtain two large-scale datasets from Snapchat. Each dataset is constructed from all users that live in a different city (on two different continents), we filter out inactive/already churned users. We follow previous studies on Snapchat [67] and collect 13 representative features for user actions on Snapchat, normalizing to zero mean and unit variance independently before training. Table 4.5 in Supplementary provides explains each feature. We consider 1-day time intervals over 6 weeks. We use the 3 weeks for training, and the rest for testing. We use 2 weeks of user graphs as input to predict engagement in the following week (i.e., $\Delta t = 7d$).

To show that FATE is general for multiple prediction scenarios, we evaluate on two notions of user engagement. The first metric considers user session time in hours (winsorized to remove extreme outliers). The second metric considers snap related activities, which are core functions of Snapchat. We aggregate and average four normalized snap related features, including send, view, create and save, as the measurement for user engagement. The prediction of user engagement scores based on two different metrics is denoted by Task 1 and Task 2, respectively. We choose root mean square error (RMSE), mean absolute percentage error (MAPE), and mean absolute error (MAE) as our evaluation metrics. We run all experiments 10 times and report the averaged results. Other technical details are discussed in Supplementary 4.6.2. Our code is publicly available on Github\(^1\).

4.5.2 Compared Methods

To validate the accuracy of user engagement prediction, we compare FATE with the following state-of-the-art methods:

---

\(^1\)https://github.com/tangxianfeng/FATE
• **Linear Regression** (LR): we utilize the averaged feature vectors of each node in \( G_t \) as a representation for time interval \( t \), and concatenate the vectors over all past time intervals as the input.

• **XGBoost** (XGB) [15]: We adopt the same prepossessing steps of LR as input for XGBoost.

• **MLP** [39]: We experiment on a two-layer MLP with the same input features to LR and XGBoost.

• **LSTM** [41]: LSTM is a popular RNN model for various sequential prediction tasks. We implement a two-layer LSTM which iterates over historical user action features. The final output is fed into a fully-connected layer to generate prediction.

• **GCN** [52]: We combine all historical dynamic friendship graphs into a single graph. For each user, we concatenate action features over the observed time period into a new nodal feature vector.

• **Temporal GCN-LSTM** (TGLSTM) [67]: TGLSTM is designed to predict future engagement of users, and can be treated as current state-of-the-art baseline. TGLSTM first applies GCN on action graph at each time interval, then leverage LSTM to capture temporal dynamics. We adopt the same design following Liu et al.[67] and train TGLSTM on our data to predict the engagement score.

To measure the explainability of \textit{FATE}, we compare with the feature importance of XGB, and LSTM with temporal attention. After the boosted trees of XGB are constructed, the importance scores for input features are retrieved and reshaped as an explanation for temporal importance. For LSTM, we compute attention scores over all hidden states as an explanation for time intervals.

### 4.5.3 User Engagement Prediction Performance

To answer the first research question, we report user engagement prediction accuracy of above methods in Table 4.1. As we can see, \textit{FATE} achieves best performance in both tasks. As expected, \textit{FATE} significantly out-performs two feature-based methods LR and XGB since it captures friendship relation and temporal dynamics. Deep-learning based methods MLP, GCN, and LSTM achieves similar performance. However, \textit{FATE} surpasses them with tremendous error reduction. Moreover, \textit{FATE} outperforms state-of-the-art approach TGLSTM, by at most 10%.

There are two potential reasons. First, \textit{FATE} additionally captures friendship relation by explicitly modeling user-user interaction. Secondly, tGCN and tLSTM maintain independent pa-
Table 4.1: FATE consistently outperforms alternative models in prediction error metrics on both Task 1 and Task 2, and both datasets Region 1 and Region 2.

<table>
<thead>
<tr>
<th></th>
<th>Region 1</th>
<th></th>
<th>Region 2</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RMSE</td>
<td>MAPE</td>
<td>MAE</td>
<td>RMSE</td>
</tr>
<tr>
<td>Task 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LR</td>
<td>0.188±0.001</td>
<td>0.443±0.001</td>
<td>0.153±0.000</td>
<td>0.183±0.000</td>
</tr>
<tr>
<td>XGB</td>
<td>0.141±0.000</td>
<td>0.260±0.000</td>
<td>0.101±0.000</td>
<td>0.140±0.000</td>
</tr>
<tr>
<td>MLP</td>
<td>0.139±0.003</td>
<td>0.233±0.007</td>
<td>0.094±0.004</td>
<td>0.125±0.005</td>
</tr>
<tr>
<td>GCN</td>
<td>0.131±0.012</td>
<td>0.228±0.019</td>
<td>0.094±0.007</td>
<td>0.128±0.008</td>
</tr>
<tr>
<td>LSTM</td>
<td>0.121±0.005</td>
<td>0.221±0.003</td>
<td>0.093±0.003</td>
<td>0.122±0.002</td>
</tr>
<tr>
<td>TGLSTM</td>
<td>0.114±0.002</td>
<td>0.215±0.005</td>
<td>0.088±0.000</td>
<td>0.122±0.005</td>
</tr>
<tr>
<td>FATE</td>
<td>0.109±0.003</td>
<td>0.204±0.001</td>
<td>0.081±0.001</td>
<td>0.118±0.002</td>
</tr>
<tr>
<td>Task 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LR</td>
<td>0.201±0.000</td>
<td>0.674±0.001</td>
<td>0.160±0.000</td>
<td>0.190±0.000</td>
</tr>
<tr>
<td>XGB</td>
<td>0.100±0.000</td>
<td>0.347±0.000</td>
<td>0.078±0.001</td>
<td>0.134±0.000</td>
</tr>
<tr>
<td>MLP</td>
<td>0.088±0.003</td>
<td>0.288±0.006</td>
<td>0.066±0.003</td>
<td>0.101±0.002</td>
</tr>
<tr>
<td>GCN</td>
<td>0.094±0.006</td>
<td>0.294±0.008</td>
<td>0.069±0.004</td>
<td>0.100±0.002</td>
</tr>
<tr>
<td>LSTM</td>
<td>0.080±0.002</td>
<td>0.249±0.005</td>
<td>0.059±0.002</td>
<td>0.097±0.002</td>
</tr>
<tr>
<td>TGLSTM</td>
<td>0.079±0.001</td>
<td>0.241±0.006</td>
<td>0.058±0.000</td>
<td>0.095±0.001</td>
</tr>
<tr>
<td>FATE</td>
<td>0.072±0.001</td>
<td>0.213±0.003</td>
<td>0.053±0.000</td>
<td>0.093±0.000</td>
</tr>
</tbody>
</table>

rameters to capture exclusive information for every user actions, which enhances the predicting accuracy.

4.5.4 Ablation Study

To answer the second question, we design four variations of FATE as follow: (1) FATE<sub>ts</sub>: We first evaluate the contribution of tensor-based design. To this end, we employ the original GCN [52] and LSTM [41] to create the first ablation FATE<sub>ts</sub>. We use the last output from LSTM to predict user engagement score. (2) FATE<sub>fnd</sub>: We then study the effectiveness of the friendship module. We apply tLSTM on raw features to create FATE<sub>fnd</sub>. (3) FATE<sub>tmp</sub>: Next we study the contribution from the temporal module. FATE<sub>tmp</sub> first concatenate outputs from all friendship modules, then apply a fully-connected layer to generate user engagement score. (4) FATE<sub>int</sub>: To analyze the contribution of explicitly modeling user interactions, we remove this part to create the last ablation FATE<sub>int</sub>. The performance of all variations are reported in Table 4.2. FATE<sub>ts</sub> performs worse when compared to FATE because it fails to extract exclusive information from each user action. However, it still outperforms TGLSTM, since user interactions enhance the modeling of friendship relation. The comparisons among FATE<sub>fnd</sub>, FATE<sub>tmp</sub> and FATE indicate the effectiveness of modeling friendship and temporal dependency for predicting user engagement. The comparison between FATE<sub>int</sub> and FATE highlights the contribution of user
To answer the third research question, we first analyze the explanations derived from FATE. Then we compare the results with explanations from baseline methods.

### 4.5.5 Explainability Evaluation

To answer the third research question, we first analyze the explanations derived from FATE. Then we compare the results with explanations from baseline methods.

#### 4.5.5.1 User Action Importance

We first study the global user action importance $A^*$. Figure 4.5 illustrates the importance score of different user actions, where a larger value indicates higher importance for user engagement.
Since the objective of Task 1 is to predict a session time-based engagement score, the importance of historical app usage length is significantly higher. This indicates that historical session time is the key factor for user engagement (defined by the expectation of session time in the future), as user activities usually follow strong temporal periodicity. Remaining user actions play similar roles in extending session time, which is intuitive, because on the entire App level, all the represented in-App functions are heavily consumed. However, we see that Snap-related actions are relatively more important than others. A potential reason is that sending and receiving Snaps (images/videos) are core functions which distinguish Snapchat from other Apps and define product value.

For predicting user engagement defined on normalized Snap-related actions in Task 2, we see that SnapSend, SnapView, and SnapCreate play the most important role. SnapSend contributes more to user engagement comparing with SnapView, as sending is an active generation activity while viewing is passively receiving information. Similarly, SnapCreate is more important than SnapSave, for the reason that creating a Snap is the foundation of many content generation activities, whereas Snap-saving is infrequent. Besides Snap-related actions, ChatSend is the most important, which makes sense given that private Chat messaging is the next most common use case after Snaps on Snapchat, and users often respond to Snaps with Chats and vice-versa.

Next, we analyze user action importance for individual users. We take Task 2 as an example, and select two random users from Region 1. To help understand user preference and characteristics, we query an internal labeling service that categorizes users according to their preferences for different Snapchat features. The service, built on domain knowledge, is as independent from FATE. Generally, a “Snap-er” uses Snap-related functions more frequently, while a “Story/Discover Viewer” is more active on watching friend/publisher Story content on
Figure 4.7: FATE’s global temporal importances show long and short-term action importances over time.

Snapchat. As illustrated in Figure 4.6, the importance scores of Snap-related user actions of a Snap-er are significantly higher than that of remained user actions. However, for Story/Discover Viewers, other actions (StoryView, Public-DiscoverView) contribute more. This shows the diversity of action importance for individual users, as the distribution of importance scores changes according to user characteristics.

### 4.5.5.2 Temporal Importance

Figure 4.7 displays the overall temporal importance of user actions across time (i.e., past 14 days). Darker hue indicates higher importance to user engagement. For **Task 1**, SessionTime has strong short-term importance in both cities. Temporally close SessionTime (later days) data contributes to user engagement more. On the contrary, other user actions show long-term importance. For example, SnapView and ChatView show relatively higher importance on the first day. In addition to long/short-term characteristics, we see the importance of most user actions showing strong periodicity in a weekly manner. Similar conclusions can also be drawn from **Task 2**, where SnapView, SnapCreate, and SnapSave show longer-term correlation to user engagement. SnapSend on the other hand demonstrates a short-term correlation. The periodicity of temporal importance is also relatively weaker compared to **Task 1**.

We then study the temporal importance for individual users. Similar to action importance,
Figure 4.8: FATE can capture diverse local level temporal importance for users of different persona.

we randomly select two users from Region 1, and plot temporal importance scores when predicting user engagement score in Task 1. As shown in Figure 4.8, users with different dominant behaviors exhibit different temporal importance score distributions. The temporal importance scores of Publisher-DiscoverView and DiscoverViewTime are relatively higher for the Story/Discover Watcher, with clear periodicity effects (importance in day 1-2, and then in 7-8, and again in 13-14, which are likely weekends when the user has more time to watch content). The Chatter has higher score for Chat-related features, with more weight on recent ChatViews (days 12-14). Our results suggest that explanations learned by FATE coincide with past understanding of temporal influence in these behaviors.

4.5.5.3 Friendship Importance

We next validate the learned (local) friendship importance. Figure 4.9 demonstrates two example users selected from Region 1, for Task 1. The heatmaps illustrate the importance scores of their friends. Clearly, friendship importance scores are not uniformly distributed among all friends. Some friends hold higher importance scores to the selected user, while others have relatively lower scores. This is potentially due to low similarity in user activities, or two friends being independently active (but not jointly interactive). To verify this assumption and interpret friendship importance scores, we compare user activeness (session time) of the selected user with their most important friends and their least importance friends (measured by the sum of scores over 14 days). As Figure 4.9 shows, the both users follow a pattern similar to their most important friends and unlike the least important ones. Moreover, temporal importance (darker hue) of the highest-importance friend coincides in the temporal heatmaps (left) and the session time activity plots (right) for both users in (a) and (b).
Figure 4.9: FATE’s local friendship importance captures asymmetric influence of friends: the user has similar session time behaviors (right) as their highest-importance friends (blue and orange lines are close); session time spikes coincide with high temporal importances (left) of those friends (dark hues).

Figure 4.10: Comparisons of explainability.

4.5.5.4 Baseline comparisons on explainability

Feature importance from XGBoost can be used as a temporal importance explanation. As in Figure 4.10, results from XGBoost are very sparse, where most user actions receive an unnatural, near-0 importance score, likely because feature importance is only a byproduct of the training of XGBoost. Unlike FATE, the XGBoost objective is purely defined on prediction accuracy, failing
to learn explanations for user actions over time. Figure 4.10 shows the temporal attention from LSTM. There are two weaknesses of using LSTM for explanation: (1) it is unable to capture the importance of each user action; (2) compared to FATE, the temporal attention fails to capture periodicity of user actions, which na"ive LSTM mixes and cannot separate. Comparatively, FATE derives richer and more fine-grained explanations.

4.5.6 Practical Applications

Our framework is designed with practical applications in mind. State-of-the-art in engagement prediction improves temporally-aware estimation of overall demand and key metrics, which offers flexible use in many forecasting and expectation-setting applications. Explainability in the model helps quantify both global and local factors in user engagement, and how they motivate users to engage with the platform. Moreover, it paves roads for personalized interventions and nudges to users to realize in-App value, stay in touch with their best friends and retain. Finally, our choices around tensor-based modeling improve efficiency by reducing parameters and decreasing training time. Yet, GNN training/inference is still a challenge for multi-million/billion-scale workloads, especially considering dynamism of the underlying data, temporality of predictions, and frequent model updation needs in practice, though new work in GNN scalability offers some promising inroads [128, 17]. In the future, we plan to develop automated and recurrent training and inference workflows which can handle these issues to gracefully scale FATE to production workloads larger than those we experimented on.

4.6 Supplementary Material

4.6.1 Complexity of FATE

4.6.1.1 Theoretical Analysis

In this section, we analyze the complexity of FATE. In particular, we focus on the complexity reduction from the tensor-based designs of GCN and LSTM over the standard ones. Without loss of generality, we use $d_{in}$ and $d_{out}$ to denote the dimensions of input and output of a neural network layer (e.g., GCN, LSTM, etc.). We use the number of learnable parameters (neurons in the network) to measure the network complexity as follows:

**Theorem 3.** By replacing the standard GCN and LSTM layers with corresponding tensor-based versions, the network complexity is reduced by $(1 - \frac{1}{R})d_{in} \cdot d_{out}$ and $4(1 - \frac{1}{R})(d_{in} + d_{out})d_{out}$ number of trainable parameters, respectively.
Proof. The number of trainable parameters for the GCN layer is \( d_{in} \cdot d_{out} \) (see Eqn. 4.1), and that for the tensor-based GCN layer is \( K \cdot \left( \frac{d_{in}}{K} \cdot \frac{d_{out}}{K} \right) = \frac{d_{in} \cdot d_{out}}{K} \) (see Eqn. 4.2, assume they are equally divided into each category of user action features). Therefore, tensor-based GCN reduces network complexity by \((1 - \frac{1}{K}) \cdot \frac{d_{in} \cdot d_{out}}{K}\). Similarly, the standard LSTM layer has \(4(d_{in} \cdot d_{out} + d_{out}^2 + d_{out})\) trainable parameters (corresponding to the input transition, hidden state transition, and the bias); while the tensor-based LSTM layer only maintains \(4\left(\frac{d_{in} \cdot d_{out}}{K} + \frac{d_{out}^2}{K} + d_{out}\right)\) number of parameters (for \(U_*, U^h_*\) and \(b_*\) in Eqn. 4.7). As a result, the total number of parameters is reduced by \(4(1 - \frac{1}{K})(d_{in} + d_{out})d_{out}\) when adopting the tensor-based LSTM over the standard one.

The computational complexity comes from multiplications. The reduction of computational complexity is analyzed through Theorem 4:

**Theorem 4.** The tensor-based GCN and the tensor-based LSTM reduce the computational complexity by \(O(d_{in} \cdot d_{out}) \) and \(O((d_{in} + d_{out})d_{out}) \), respectively.

**Proof.** Let \( N \) denote the number of nodes in the ego-network. Using Eqn. 4.1 and 4.2, the computational complexity of a GCN layer and a tensor-based GCN layer are \(N^2 \cdot d_{in} + N \cdot \frac{d_{in} \cdot d_{out}}{K}\) and \(N^2 \cdot \frac{d_{in}}{K} \cdot K + N \cdot \frac{d_{in}}{K} \cdot \frac{d_{out}}{K} \cdot K = N^2 \cdot \frac{d_{in}}{K} \cdot \frac{d_{out}}{K} \), respectively. The reduction is then \(N(1 - \frac{1}{K})d_{in} \cdot d_{out} = O(d_{in} \cdot d_{out})\). For an LSTM layer (Eqn. 4.7), it takes \(4(d_{in} \cdot d_{out} + d_{out}^2) + 3d_{out}\) multiplications to update its hidden and gate, while the tensor-based LSTM layer takes only \(4\left(\frac{d_{in} \cdot d_{out}}{K} + \frac{d_{out}^2}{K} \cdot K\right) + 3d_{out} = 4\left(\frac{d_{in} \cdot d_{out}}{K} + \frac{d_{out}^2}{K}\right) + 3d_{out}\) multiplications. Thus, the reduction of computational complexity by the tensor-based LSTM is \(O((d_{in} + d_{out})d_{out})\). □

Note that for FATE, it adopts multiple friendship modules with the tensor-based LSTM. Therefore, FATE is significantly benefited from the tensor-based design, reducing both network size and computational complexity sharply. However, the overall improvement over complexity does not exactly aligned with these tensor-based designs, due to costs from extra components in FATE such as the computation of attention scores. Therefore, we also analyze the real-world running time of FATE quantitatively in the following experiment section.

### 4.6.1.2 Experimental Results

We study the runtime complexity of FATE. We compare the runtime of FATEts and FATE, to demonstrate the improvement by using tensor-based designs for FATE over a non tensor-based model FATEts. Both training and testing (inference) run times are reported in 4.3. We can see that training FATE takes significantly less time then FATEts by an average of 20%. In addition, inference speed of FATE is also faster. Therefore, it is beneficial to adopt tensor-based designs
when constructing the framework. Note that our implementation uses PyTorch Geometric\(^2\) as the underlying message passing framework.

Table 4.3: Comparisons of Runtime (\textit{min}). FATE reduced 20\% of runtime on average comparing with non-tensor-based FATE\(_{ts}^{\text{eq}}\).

<table>
<thead>
<tr>
<th>Task</th>
<th>Region 1</th>
<th>Region 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Train</td>
<td>Test</td>
</tr>
<tr>
<td>Task 1</td>
<td>FATE(_{ts}^{\text{eq}})</td>
<td>216.96</td>
</tr>
<tr>
<td></td>
<td>FATE</td>
<td>181.35</td>
</tr>
<tr>
<td>Task 2</td>
<td>FATE(_{ts}^{\text{eq}})</td>
<td>207.23</td>
</tr>
<tr>
<td></td>
<td>FATE</td>
<td>172.00</td>
</tr>
</tbody>
</table>

4.6.2 Implementation Details

4.6.2.1 Experimental Environment

Our experiments are conducted on a single machine on Google Cloud Platform\(^3\), with a 16-core CPU, 60GB memory and 2 Nvidia P100 GPUs.

4.6.2.2 Data Preprocessing

We select two geographic regions, one from North America and the other from Europe, to compile two datasets. We set the time period from 09/16/2019 to 10/27/2019, with a one-day time interval length. There are totally 42 days in the time period (6 weeks). For each dataset, we first query all users whose locations are within the corresponding region. Users who spend less than one minute (session time) on a daily average are treated as extremely inactive and filtered. We then obtain the friendship of these users as our social network and historical user action

\(^2\)https://github.com/rusty1s/pytorch\_geometric
\(^3\)https://cloud.google.com

Table 4.4: Statistics of Datasets.

<table>
<thead>
<tr>
<th></th>
<th>Region 1</th>
<th>Region 2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Time period</strong></td>
<td>09/16/2019 - 10/27/2019</td>
<td></td>
</tr>
<tr>
<td><strong>Avg. # users</strong></td>
<td>153006</td>
<td>108452</td>
</tr>
<tr>
<td><strong>Avg. node degree</strong></td>
<td>51.58</td>
<td>36.95</td>
</tr>
<tr>
<td><strong># node features</strong></td>
<td>13</td>
<td></td>
</tr>
<tr>
<td><strong># edge features</strong></td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>
Table 4.5: Selected features for user actions on Snapchat.

<table>
<thead>
<tr>
<th>In-App function</th>
<th>Feature name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Snap</td>
<td>SnapSend</td>
<td># of snaps sent to friends.</td>
</tr>
<tr>
<td></td>
<td>SnapView</td>
<td># of snaps viewed from friends.</td>
</tr>
<tr>
<td></td>
<td>SnapCreate</td>
<td># of snaps created by the user.</td>
</tr>
<tr>
<td></td>
<td>SnapSave</td>
<td># of snaps saved to the memory/smartphone.</td>
</tr>
<tr>
<td>Chat</td>
<td>ChatSend</td>
<td># of text messages sent to friends.</td>
</tr>
<tr>
<td></td>
<td>ChatView</td>
<td># of received text messages.</td>
</tr>
<tr>
<td>Story</td>
<td>StoryPost</td>
<td># of videos posted to the user’s page</td>
</tr>
<tr>
<td></td>
<td>StoryView</td>
<td># of watched story videos posted by others.</td>
</tr>
<tr>
<td></td>
<td>StoryViewTime</td>
<td>Total time spent for watching stories.</td>
</tr>
<tr>
<td>Discover</td>
<td>FriendDiscoverView</td>
<td># of watched videos posted by friends on Discover page</td>
</tr>
<tr>
<td></td>
<td>PublisherDiscoverView</td>
<td># of watched videos posted by publisher on Discover page</td>
</tr>
<tr>
<td></td>
<td>DiscoverViewTime</td>
<td>Total time spent for watching videos on Discover page.</td>
</tr>
<tr>
<td>Misc.</td>
<td>SessionTime</td>
<td>Total time spent on Snapchat.</td>
</tr>
</tbody>
</table>

Besides, we also acquire user-user commutation as features for user interaction, including chat, snap, and story. These features are constructed from the aggregation of each type of interaction. Table 4.4 details both datasets.

4.6.2.3 Model Implementations

We implement all compared baseline methods in Python 3.7. Linear Regression is adopted from scikit-learn\(^4\). We use XGBoost[15] from the official package\(^5\) with its recommended setting and parameters. We implement the GCN model with PyTorch Geometric. We set up a two layer GCN, with the hidden size of 128, using ELU as the activation function. Similarly, we build the LSTM model as a two-layer LSTM using PyTorch\(^6\). The hidden size is 128. We set the dropout rate to 0.5 for the second layer. ELU is used as the activation. We following the original settings for TGLSTM as introduced in the paper [67]. We implement FATE with PyTorch and PyTorch Geometric. Friendship modules contain two-layer tGCN. The dimension of output embedding for all feature categories is set to 32. The design of tLSTM is inspired by IMV-LSTM\(^7\). We use two layers of tLSTM for FATE. Our code is available on Github\(^8\).

\(^4\)https://scikit-learn.org
\(^5\)https://xgboost.readthedocs.io/
\(^6\)https://pytorch.org/
\(^7\)https://github.com/KurochkinAlexey/IMV_LSTM
\(^8\)https://github.com/tangxianfeng/FATE
For LR and Xgboost, we train until convergence. For neural network models, we set the batch size to 256 and the max number of epoch to 10. All models are optimized by Adam algorithm [51], with a learning rate of 0.001. They are trained until reaching the max epoch or early-stopped on the validation set. The validation set contains 10% samples randomly selected from the training set. All methods are trained and tested 10 times to get averaged results.

4.6.2.4 Evaluation Metrics

Three common metrics Root Mean Square Error (RMSE), Mean Absolute Percentage Error (MAPE) and Mean Absolute Error (MAE) are used to evaluate the performance of all methods. The detailed definitions of these metrics are stated as below:

\[
\text{RMSE} = \sqrt{\frac{1}{|S|} \sum_{u \in S} (e^u - \hat{e}^u)^2},
\]
\[
\text{MAPE} = \frac{1}{|S|} \sum_{u \in S} \frac{|e^u - \hat{e}^u|}{\hat{e}^u},
\]
\[
\text{MAE} = \frac{1}{|S|} \sum_{u \in S} |e^u - \hat{e}^u|, \tag{4.17}
\]

where \(\hat{e}^u\) denotes the ground truth of predicted user engagement score \(e^u\).

While RMSE and MAE receive higher penalties from larger values, MAPE focuses on the prediction error of samples with smaller engagement scores. Therefore, combining these metrics leads to more comprehensive conclusions.
Chapter 5

Discussion

In this dissertation, I go beyond the accurate graph neural networks to build robust, fair, and explainable GNN models. Firstly, I study a new problem of exploring extra clean graphs for learning a robust GNN against the poisoning attacks on a target graph. I propose a novel framework PA-GNN, that leverages penalized attention mechanism to learn the ability to reduce the negative impact from perturbations on clean graphs and meta-optimization to transfer the alleviation ability to the target poisoned graph. Experimental results of node classification tasks demonstrate the efficacy of PA-GNN against different poisoning attacks. To reduce the discrimination and biases in GNNs, I empirically analyze an issue of GNN for semi-supervised node classification, i.e., when labeled nodes are randomly distributed on the graph, nodes with low degrees tend to have very few labeled nodes, which results in sub-optimal performance on low-degree nodes. To solve this issue, I propose a novel framework SL-DSGCN, which leverages degree-specific GCN layers and the self-supervised-learning with Bayesian teacher network to introduce more labeled neighbors for low-degree nodes. Experimental results on real-world datasets demonstrate the effectiveness of the proposed framework for semi-supervised node classification. I also discuss the contributions of each component of SL-DSGCN. Finally, I explore the problem of explainable GNN designs for user engagement prediction in social network Apps. I propose an end-to-end neural framework, FATE, which contains a self-explainable GCN module and LSTM component to generate accurate predictions while jointly deriving local and global explanations for key factors of social network Apps. Extensive experiments on two datasets and two engagement prediction tasks from Snapchat demonstrate the efficiency, generality, and accuracy of my approach: FATE improves accuracy compared to state-of-the-art methods by $\approx 10\%$ while reducing runtime by $\approx 20\%$ owing to its use of proposed tensor-based self-explainable GCN and LSTM components.
There are several interesting directions that need further investigation, where I will keep improving the robustness, fairness, and explainability of graph neural networks. In particular, I would like to explore the potential of transfer learning for improving robustness on other models, such as community detection and graph classification. The fairness of GNNs can be further improved by expanding the proposed DSGCN layer and self-supervised-learning with the Bayesian teacher network to various GNNs. I will investigate the generic framework for other GNNs such as GAT [106]. In addition, I will study the degree issue of heterogeneous information networks [92], which are also pervasive in the real world. One potential approach is extending SL-DSGCN for heterogeneous networks by considering different types of links/edges. Lastly, I would like to explore the explainable GNNs in other domains, such as multi-label graphs, spatial-temporal graphs, and heterogeneous graphs.


[92] Chuan Shi, Yitong Li, Jiawei Zhang, Yizhou Sun, and S Yu Philip. A survey of heterogeneous information network analysis. *TKDE*, 2016.


[121] Carl Yang, Xiaolin Shi, Luo Jie, and Jiawei Han. I know you’ll be back: Interpretable new user clustering and churn prediction on a mobile social application. In *KDD*, pages 914–922. ACM, 2018.


Vita

Xianfeng Tang

I get my Ph.D. from the College of Information Sciences and Technology, the Pennsylvania State University. My advisers are Dr. Prasenjit Mitra and Dr. Suhang Wang. My research interests include machine learning and data mining. In particular, I focus on machine learning models and applications for graph-structured data, including adversarial learning, explainable machine learning, fairness, and computational social sciences. I have published 15+ papers in top conferences and journals, such as KDD, CIKM, WSDM, WWW, AAAI, etc. and have earned over 700 citations. Some selected publications are listed as follows:

