공학박사 학위논문

Improving Message-Passing and Representation Learning of Graph Neural Networks

그래프 신경망의 메시지 전달 및 표현 학습 개선

2023 년 8 월

서울대학교 대학원

컴퓨터공학부

최윤혁

Improving Message-Passing and Representation Learning of Graph Neural Networks

지도 교수 권 태 경

이 논문을 공학박사 학위논문으로 제출함
2023 년 6 월

서울대학교 대학원
컴퓨터공학부
최윤혁

최윤혁의 공학박사 학위논문을 인준함
2023 년 6 월

위 원 장 이 상 구 (인)

부위원장 권 태 경 (인)

위 원 강 유 (인)

위 원 김 종 권 (인)

위 원 양 은 호 (인)

Improving Message-Passing and Representation Learning of Graph Neural Networks

by

Yoonhyuk Choi

Department of Computer Science & Engineering

Doctor of Philosophy in Seoul National University

Abstract

Graph Neural Networks (GNNs) achieve substantial improvement in analyzing graph-structured datasets under semi-supervised setting, where few labels are available during the training. The discriminative power of GNNs stem from the message-passing scheme, where they utilize information from neighboring nodes. Generally, under the graphs with strong homophily, features from the adjacent nodes can be used to guide decision boundary (e.g., neural networks) more precisely. Nonetheless, they fail to achieve satisfying results under heterophilous graphs, where most edges connect two nodes with different labels.

In the first paper, we analyze the performance of GNNs based on the multiple propagation schemes theoretically. For example, flipping the sign of edges is rooted in a strong theoretical foundation, and attains significant performance enhancements. Nonetheless, they assume a binary class scenario and they may suffer from confined applicability. Here, we extend the prior understandings to multi-class scenarios and points out two drawbacks: In case two nodes belong to different classes but have a high similarity, signed propagation can

decrease the discrimination power of the GNNs, (2) signed message also increases the prediction uncertainty (e.g., conflict evidence) which can impede the stability of the algorithm.

In the second paper, we focus on finding the heterophilous edges, which can degrade the overall quality of GNNs significantly. To achieve this, we employ a confidence ratio as a hyper-parameter, assuming that some of the edges are disassortative (heterophilic). Here, we suggest the two-phased algorithm, (1) determining edge coefficients through subgraph matching using a supplementary module, and (2) the application of modified label propagation. Specifically, our supplementary module identifies a certain proportion of task-irrelevant edges based on a given confidence ratio. Further, the improved label propagation mechanism prevents two nodes with smaller weights from being closer effectively.

Lastly, we introduce the limitation of GNNs from another perspective, where they suffer from sparsity in initial node features. This can result in overfitting of the first projection matrix (or hyperplane), where the dimensions with zero inputs are not updated during training. To address this issue, we propose a novel data augmentation strategy, which flips the initial features and the hyperplane simultaneously. To the best of our knowledge, this is the first attempt to mitigate the overfitting problem caused by input features.

Keywords: Graph neural network, Semi-supervised learning, Message-passing, Signed propagation, Calibration, Heterophilic neighbor, Subgraph matching, Confidence ratio, Sparseness of node features

Student Number: 2019-25552

Acknowledgments

First of all, I would like to express my gratitude to my advisor Chong-Kwon Kim, who guided me for about five years. I would also like to extend my thanks to the thesis committee members: Professor Sang-goo Lee, Taekyoung Kwon, U Kang, and Eunho Yang, for their participation and valuable insights. Without them, I also could not have completed this, who generously provided their expertise and knowledge. Further, this endeavor would not have been possible without the generous support from the following foundation, who financed my research.

- National Research Foundation of Korea (NRF) (No. 2016R1A5A1012966, No. 2020R1A2C110168713)
- Technology Planning & Evaluation (IITP) (No. 2021-0-02068 Artificial Intelligence Innovation Hub, No. RS-2022-00156287 Innovative Human Resource Development for Local Intellectualization support program) grant funded by the Korea government (MSIT)

I am also thankful to my colleagues, especially my senior Jiho Choi and Hyungho Byun, office mates Taewook Ko, Ahyun Lee, and Hyungho Bae for their editing help and encouragement. Additionally, I'd like to appreciate to the staffs, research assistants, and study participants from the university, who impacted and inspired me.

Lastly, I would like to mention my family, especially my parents. Their belief has kept my spirits and motivation high during this process. I would also like to thank my friends for all the entertainment and emotional support.

List of Figures

List of Tables

List of Algorithms

Contents

Chapter 1

Introduction

The increase in graph-structured datasets has led to rapid advancements in graph mining techniques. Especially, GNNs provide satisfactory performances in various applications including node classification and link prediction, which also has been adopted in many fields; physics [28], protein-protein interactions [26], and social networks [23]. The main component of GNNs is message-passing [28], where the information is propagated between nodes and then aggregated. Also, the integration of a structural property with the node features enhances the representation and the discrimination powers of GNNs substantially [73; 16; 42; 32; 81]. Consequently, GNNs often have shown the best performance in various tasks including semi-supervised node classification and link prediction.

Early GNN schemes assume the network homophily where nodes of similar attributes make connections with each other based on the selection [59] or social influence theory [27]. Plain GNN algorithms [16; 42] simply perform Laplacian smoothing (a.k.a low-pass filtering) to receive low-frequency signals from neighbor nodes. Consequently, these methods fail to adequately deal with heterophilous graphs [63; 66; 99] such that even a simple MLP outperforms GNN in some cases. To relieve this problem, a plethora of clever algorithms have been proposed including the adjustment of edge coefficients [81; 37; 4], aggregation of remote nodes with high similarity [67; 53], and diversified message propagation [91]. However, the majority of prior schemes [57] stipulate certain conditions of advantageous heterophily and these constraints undermine their generality and applicability [66].

Many clever schemes have been introduced to solve the problem. Some of them specify different weights for each connection [81; 92; 2; 40], or remove disassortative edges [94; 22; 55]. Others employ distant nodes with similar features [67; 93; 37] or apply different aggregation boundary based on the central nodes [85]. Additionally, some bodies of work allow the edge coefficients to be negative [13; 2] to preserve high-frequency signal exchanges between neighbors. Further, from the perspective of gradient flow, [2; 20] shows that negative eigenvalue preserves the high-frequency signals to dominate during propagation. [3] introduces sheaf to enhance the linear separability of neural networks.

In the first paper, we aim to provide theoretical justification to answer this question "what kind of message-passing algorithm achieves best performance?", including signed and zero-weighted propagation. Firstly, we point out some limitation of previous analysis [57; 89] that provide theoretical boundaries under a binary class scenario, which may detriment their applicability to generic graphs. Here, we extend the theorem to a multi-class scenario positing that the blind application of signed messages to multi-class graphs may increase the uncertainty of predictions.

In addition to the theoretical understanding, we propose another method through the second paper, which aims to find heterophilous edges through subgraph matching. To achieve this, we focus on the GAM [78] that suggests a supplementary module with label propagation. Specifically, the supplementary module of GAM only utilizes a central node to debilitate noises, which is identical to a simple MLP. Though GAM might work well under high heterophily, they fail to generalize well under homophilous graphs. To solve this limitation, we measure the similarity of two nodes including their subgraphs by employing the widely used optimal transport [69; 87; 60; 44]. In addition, we further apply a confidence ratio to remove certain proportion of disassortative edges. Finally, considering

these predictions as supplementary edge coefficients, we apply label propagation [6] between a certain proportion of high confident edges.

Lastly, our focus is on the characteristics of graph datasets. We have observed that features from benchmark graph datasets have few non-zero elements (e.g., bag-of-words representation). Here, we contemplate that the shortage of training samples in semi-supervised settings can result in the overfitting of specific dimensions in the first layer parameters. This can negatively impact the quality of predictions for test nodes with untrained features in those dimensions. To optimize the first layer projection matrix better, we focused on perturbing the initial features. As a common data augmentation technique, dimensional shifting could be used which is commonly used in computer vision [75]. However, this was found to be unsuitable for GNNs with bag-of-words features, as it would disrupt the semantic information. Our proposed solution involves flipping the initial features and parameters simultaneously, which can ensure local invariance. This approach is inspired by shifting parameters [41] and rotating neural networks [51] that preserve the volume of gradients and initial features. This flipping mechanism can address the issue of zero gradients caused by sparse inputs and enhance the semantic learning of each dimension.

To summarize, chapter 3 analyzes the power of various message-passing schemes theoretically. In chapter 4, we provide our subgraph-based GNN, which can generalize well under heterophilous settings. Finally, chapter 5 provides new insights from gradient perspectives, which points out some limitations of graph benchmark datasets to solving semi-supervised classification scenario.

Chapter 2

Preliminary

In this section, we define some useful notations and explain the basics of the graph-related problems.

Let $G = (V, E, X)$ be a graph with $|V| = n$ nodes and $|E| = m$ edges. The node attribute matrix is $X \in \mathrm{R}^{n \times F}$, where F is the dimension of an input vector. Given X, the hidden representation of node features $H^{(l)}$ (l-th layer) is derived through message passing. Here, node $i's$ feature is the row of $h_i^{(l)}$. The structural property of G can be represented by its adjacency matrix $A \in \{0,1\}^{n \times n}$. Also, D is a diagonal matrix with node degrees $d_{ii} = \sum_{j=1}^{n} A_{ij}$. Each node has its label $Y \in \mathrm{R}^{n \times C}$, where C represents the number of classes.

The goal of semi-supervised node classification is to predict the class of unlabeled nodes $V_U = \{V - V_L\} \subset V$ given the partially labeled training set V_L. Generally, we assume 5% of entire nodes are available during training phase.

Chapter 3

Improving Signed Propagation for Graph Neural Networks

Message-passing Graph Neural Networks (GNNs), which collect information from adjacent nodes, achieve satisfying results on homophilic graphs. However, their performances are dismal in heterophilous graphs, and many researchers have proposed a plethora of schemes to solve this problem. Especially, flipping the sign of edges is rooted in a strong theoretical foundation, and attains significant performance enhancements. Nonetheless, previous analyses assume a binary class scenario and they may suffer from confined applicability. This paper extends the prior understandings to multi-class scenarios and points out two drawbacks: (1) In case two nodes belong to different classes but have a high similarity, signed propagation can decrease the discrimination power of the GNNs, (2) signed message also increases the prediction uncertainty (e.g., conflict evidence) which can impede the stability of the algorithm. Based on the theoretical understanding, we introduce two novel strategies for improving signed propagation under multi-class graphs. The proposed scheme combines calibration to secure robustness while reducing uncertainty. We show the efficacy of our theorem through extensive experiments on six benchmark graph datasets.

3.1 Introduction

The increase in graph-structured datasets has led to rapid advancements in graph mining techniques including random walk-based node embedding and graph neural networks (GNNs). Especially, GNNs provide satisfactory performances in various applications including node classification and link prediction. The main component of GNNs is message-passing [28], where the information is propagated between nodes and then aggregated. Also, the integration of a structural property with the node features enhances the representation and the discrimination powers of GNNs substantially [16; 42; 81].

Early GNN schemes assume the network homophily where nodes of similar attributes make connections with each other based on the selection [59] or social influence theory [27]. Plain GNN algorithms [16; 42] simply perform Laplacian smoothing (a.k.a low-pass filtering) to receive low-frequency signals from neighbor nodes. Consequently, these methods fail to adequately deal with heterophilous graphs [63; 66; 99] such that even a simple MLP outperforms GNN in some cases. To relieve this problem, a plethora of clever algorithms have been proposed including the adjustment of edge coefficients [81; 37; 4], aggregation of remote nodes with high similarity [67; 53], and diversified message propagation [91]. However, the majority of prior schemes [57] stipulate certain conditions of advantageous heterophily and these constraints undermine their generality and applicability.

Recently, some bodies of work allow the edge coefficients to be negative [13; 2] to preserve high-frequency signal exchanges between neighbors. Further, from the perspective of gradient flow, [2; 20] shows that negative eigenvalue preserves the high-frequency signals to dominate during propagation. [3] introduces sheaf to enhance the linear separability of neural networks. Instead of changing the signs of edges, others [55; 77] assign zero-weights to disassortative connections precluding

message diffusion on such edges. Here, there arises a question: *does signed messaging always yield better results than assigning zero-weights on heterophilic edges?*

To answer the above question, we conduct an empirical study and illustrate its results in Figure 15. Along with this, we aim to establish theoretical properties to compare their discrimination power. For this, recent studies [57; 89] scrutinize the changes in node features before and after message reception. Here, they provide some useful insights into using signed messages based on the node's relative degree and its homophily ratio. Nonetheless, prior analyses were confined to binary class graphs, which may detriment their applicability to generic graphs. In this paper, we extend the theorem to a multi-class scenario positing that the blind application of signed messages to multi-class graphs may increase the uncertainty of predictions. Throughout this analysis, we suggest employing confidence calibration [29; 84] which is simple yet effective to enhance the quality of predictions. To summarize, our contributions can be described as follows:

- Contrary to prior work confined to a binary class, we tackle the signed messaging mechanism in a multi-class scenario. Our work provides fundamental insight into using signed messages and establishing the theoretical background for the development of powerful GNNs.
- We conjecture and prove that signed messages escalate the inconsistency between neighbors and increase the uncertainty in predictions. Based on this understanding, we propose a novel uncertainty reduction method using confidence calibration.
- We conduct extensive experiments on six benchmark datasets to validate our theorems and show the effectiveness of confidence calibration.

3.2 Related Work

Graph Neural Networks (GNNs). Under semi-supervised settings, GNNs have shown great potential by utilizing the information of adjacent nodes. Early GNN studies [5; 16] focused on the spectral graph analysis (e.g., Laplacian decomposition) in a Fourier domain. However, they suffer from large computational costs as the scale of the graph increases. GCN [42] reduced the overhead by harnessing the localized spectral convolution through the first-order approximation of a Chebyshev polynomial. Another notable approach is spatial-based GNNs [81; 4] which aggregate information in a Euclidean domain. Early spatial techniques became a steppingstone to many useful schemes that encompass relevant remote nodes as neighbors.

GNNs on heterophilous graphs. Traditional message-passing GNNs fail to perform well in heterophilic graphs [67]. To redeem this problem, recent studies have paid attention to the processing of disassortative edges [17; 34]. They either capture the difference between nodes or incorporate distant but similar nodes as neighbors. For example, H2GCN [99] separates ego and neighbors during aggregation. SimP-GCN [37] suggests a long-range adjacency matrix and EvenNet [46] receives messages from even-hop away nodes only. Similarly, [48] selects neighbors from the nodes without direct connections. Configuring path-level pattern [79] or finding a compatibility matrix [100] has also been proposed. Another school of methodologies either changes the sign of disassortative edges from positive to negative [13; 2; 24; 31] or assigns zero-weights to disassortative edges [55]. Even though these schemes show their effectiveness [1] on binary classes, it may require further investigations before extending their applications to a multi-class scenario.

3.3 Preliminary

In this section, let us first define the notations and then explain the basics of the problem.

3.3.1 Definition of homophily.

The global edge homophily ratio ($\mathcal{H}_g$) is defined as:

$$\mathcal{H}_g \equiv \frac{\sum_{(i,j)\in\mathcal{E}} \mathbb{1}(Y_i = Y_j)}{|\mathcal{E}|} \tag{3.1}$$

Likewise, the local homophily (b_i) of node i is given as:

$$b_i \equiv \frac{\sum_{j=1}^{n} A_{ij} \cdot \mathbb{1}(Y_i = Y_j)}{d_{ii}} \tag{3.2}$$

3.4 Theoretical Analysis

We first discuss the mechanism of Message-Passing Neural Networks (MPNN) and the impact of using signed messages (§ 3.4.1). Then, we introduce the previous analysis of employing signed propagation on binary class graphs (§ 3.4.2). Through this, we extend them to a multi-class scenario and point out some drawbacks under this condition (§ 3.4.3). Finally, we suggest a simple yet effective solution to improve the quality of signed GNNs through the integration of calibration (§ 3.4.4).

3.4.1 Message-Passing Neural Networks.

Mechanism of Graph Neural Networks (GNNs). Generally, most of the GNNs employ the strategy of propagation and then aggregation, where the node features are updated iteratively. This can be represented as follows:

$$H^{(l+1)} = \phi(\bar{H}^{(l+1)}), \;\; \bar{H}^{(l+1)} = AH^{(l)}W^{(l)}. \tag{3.3}$$

$H^{(0)} = X$ is the initial vector and $H^{(l)}$ is nodes' hidden representations at the *l-th* layer. $H^{(l+1)}$ is retrieved through message-passing (A) and we obtain $H^{(l+1)}$ after an activation function ϕ (e.g. ReLU). $W^{(l)}$ is the trainable weight matrices that are shared across all nodes. The final prediction is produced by applying cross-entropy $\sigma(\cdot)$ (e.g., log-softmax) to $\bar{H}^{(L)}$ and the loss function is defined as:

$$\mathcal{L}_{GNN} = \mathcal{L}_{nll}(Y, \hat{Y}), \;\; \hat{Y} = \sigma(\bar{H}^{(L)}). \tag{3.4}$$

The parameters are updated by computing negative log-likelihood loss L_{nll} between the predictions (Y_b) and true labels (Y). Most GNN schemes assume that graphs are assortative and they construct the message-passing matrix (A) with positive values to preserve the low-frequency information (local smoothing) [51]. Consequently, they fail to capture the difference between node features and achieve lower performance on the heterophilous networks [65; 67].

Meaning of using signed messages. Recent studies [13; 2; 89; 12] emphasize the importance of high-frequency signals and suggest flipping the sign of disassortative edges from positive to negative to preserve such signals. We first show that they can also contribute to the separation of ego and neighbors.

3.4.2 Using Signed Messages on Binary Classes.

First, we assume a binary class and provide theoretical analysis by distinguishing two phases: message-passing and parameter update.

(Message-Passing) Signed GNN generally improves the overall performance.

In this section, we aim to analyze the movements of node features given three types of graphs (original, signed, and zero weights). We again employ GCN [42] as a baseline. Here, we assume a binary classification task ($y_i \in \{0, 1\}$) similar to previous work [1; 89] and inherit several useful notations for simplifications: (1) For all nodes $i = \{1, \ldots, n\}$, their degrees $\{d_i\}$ and features $\{\vec{h}_i\}$ are i.i.d. random variables. (2) We assume that every class has the same population. (3) With a slight abuse of notation, assume $h(0) = XW(0)$ is the first layer projection of initial node features. (4) Given the label y_i, the node feature follows the distribution (μ or $-\mu$) as:

$$\mathbb{E}(h_i^{(0)}|y_i) = \begin{cases} \mu, & \text{if } y_i = 0 \\ -\mu, & \text{if } y_i = 1 \end{cases} \tag{3.11}$$

Prior work [89] introduces Theorems 4.1, 4.2 using the local homophily (Eq. 3.2), message passing (Eq. 3.3), and expectation of node features (Eq. 3.11). Each theorem below utilizes the original and signed graph, respectively.

Theorem 4.1 (Binary class, vanilla GCN). *Let us assume $y_i = 0$. Then, the expectation after a single-hop propagation is defined as:*

$$\mathbb{E}(h_i^{(1)}|y_i, d_i) = \frac{(2b_i - 1)d_i' + 1}{d_i + 1}\mathbb{E}(h_i^{(0)}|y_i) \tag{3.12}$$

, where $d_i' = \sum_{j \in \mathcal{N}_i} \sqrt{\frac{d_i+1}{d_j+1}}$

Proof of Theorem 4.1.

Assume a binary class $y_i \in \{0,1\}$. Using the aggregation scheme of GCN [32], the hidden representation of node *i* after message-passing $h_i^{(1)}$ is defined as:

$$h_i^{(1)} = \frac{h_i^{(0)}}{d_i + 1} + \sum_{j \in \mathcal{N}_i} \frac{h_j^{(0)}}{\sqrt{(d_i + 1)(d_j + 1)}} \tag{3.13}$$

As illustrated in Figure 7a (binary class), we assume $h_i \sim N\left(\mu, 1/\sqrt{d_i}\right)$ *if* $y_i = 0$ and otherwise $h_i \sim N\left(-\mu, 1/\sqrt{d_i}\right)$. Based on the local homophily b_i, Eq. 28 becomes:

$$\begin{aligned}\mathbb{E}(h_i^{(1)}|v_i, d_i) &= \frac{\mu}{d_i+1} + \sum_{j\in\mathcal{N}_i}\left(\frac{b_i}{\sqrt{(d_i+1)(d_j+1)}}\mu - \frac{(1-b_i)}{\sqrt{(d_i+1)(d_j+1)}}\mu\right. \\ &= \left(\frac{1}{d_i+1} + \sum_{j\in\mathcal{N}_i}\frac{2b_i-1}{\sqrt{(d_i+1)(d_j+1)}}\right)\mu \\ &= \left(\frac{1}{d_i+1} + \frac{2b_i-1}{d_i+1}\sum_{j\in\mathcal{N}_i}\frac{\sqrt{d_i+1}}{\sqrt{d_j+1}}\right)\mu \\ &= \left(\frac{1+(2b_i-1)d_i'}{d_i+1}\right)\mu.\end{aligned} \tag{3.14}$$

End of proof.

The generalized version of the above theorem is described in [44], which takes two distributions μ_0,μ_1 as:

$$h_i \sim N(b_i\mu_0 + (1-b_i)\mu_1, \frac{1}{\sqrt{d_i+1}}) \tag{3.15}$$

Eq. 15 reduces to Eq. 12 when $\mu_1 = -\mu_0$.

Theorem 4.2 (Binary class, signed GCN). *If the sign of heterophilous edges is flipped correctly under the error ratio (e), the expectation is given by:*

$$\mathbb{E}(h_i^{(1)}|y_i, d_i) = \frac{(1-2e)d_i'+1}{(d_i+1)}\mathbb{E}(h_i^{(0)}|y_i) \tag{3.16}$$

Proof of Theorem 4.2. Similarly, signed GCN correctly configures the sign of heterophilous edges with the following error ratio $1-e$. For example, the sign of heterophilous nodes changes from $-\mu$ to μ with a probability $1-e$ and vice versa:

$$\begin{aligned}\mathbb{E}(h_i^{(1)}|v_i,d_i) &= \frac{\mu}{d_i+1}+\sum_{j\in\mathcal{N}_i}\left(\frac{\mu(1-e)-\mu e}{\sqrt{(d_i+1)(d_j+1)}}b_i+\frac{\mu(1-e)-\mu e}{\sqrt{(d_i+1)(d_j+1)}}(1-b_i)\right)\\ &= \frac{\mu}{d_i+1}+\sum_{j\in\mathcal{N}_i}\left(\frac{1-2e}{\sqrt{(d_i+1)(d_j+1)}}\mu\right)\\ &= \left(\frac{1}{d_i+1}+\frac{1-2e}{d_i+1}\sum_{j\in\mathcal{N}_i}\frac{\sqrt{d_i+1}}{\sqrt{d_j+1}}\right)\mu\\ &= \left(\frac{1+(1-2e)d_i'}{d_i+1}\right)\mu.\end{aligned} \tag{3.17}$$

End of proof.

(Parameter Update) Signed propagation can contributeto the separation of ego and neighbors. Let us assume an ego node i and its neighbor node j is connected with a signed edge. Let us ignore other neighbor nodes to concentrate on the mechanism of signed messaging. Applying GCN [42], we obtain the output of node i as:

$$\widehat{Y}_i=\sigma(\bar{H}_i^L)=\sigma(\frac{\bar{H}_i^{(L)}}{d_i+1}-\frac{\bar{H}_j^{(L)}}{\sqrt{(d_i+1)(d_j+1)}}) \tag{3.5}$$

Assuming that the label of the ego (Y_i) is k, we can calculate the loss (L_{nll}) between a true label $Y_i \in R^C$ and a prediction $\hat{Y}_i \in R^C$ as below:

$$\mathcal{L}_{nll}(Y_i,\widehat{Y}_i)=-\log(\widehat{y}_{i,k}) \tag{3.6}$$

Since the column-wise components of the last weight matrix $W^{(L)}$ act as an independent classifier, we prove that the probability of node j being a class k ($\hat{y}_{j,k}$), transitions in the opposite to the node i's probability ($y_{i,k}$) as the training epoch (t) proceeds:

$$\widehat{y}_{j,k}^{(t+1)} < \widehat{y}_{j,k}^{t},\ \widehat{y}_{i,k}^{(t+1)} > \widehat{y}_{i,k}^{t} \tag{3.7}$$

, where $\widehat{y}_{i,k}^{(t+1)} = \widehat{y}_{i,k}^{t} - \eta \bigtriangledown_i \mathcal{L}_{nll}(Y_i, \widehat{Y}_i)$ and $\widehat{y}_{j,k}^{(t+1)} = \widehat{y}_{j,k}^{t} - \eta \bigtriangledown_j \mathcal{L}_{nll}(Y_i, \widehat{Y}_i)$

Notation η is the learning ratio and a symbol $\bigtriangledown$ represents a partial derivative of the loss function.

Proof of Equation 7.

We first show that signed messages can contribute to separating the ego from its neighbors. Let us assume the label of the ego node i is k. A neighbor node j is connected to i with a signed edge. Since the column-wise components of the weight matrix act as an independent classifier, the probabilities that the two nodes belong to the same class, at a training epoch t are derived as,

$$\begin{aligned} \widehat{y}_{i,k}^{(t+1)} &= \widehat{y}_{i,k}^{t} - \eta \bigtriangledown_i \mathcal{L}_{nll}(Y_i, \widehat{Y}_i)_k \\ \widehat{y}_{j,k}^{(t+1)} &= \widehat{y}_{j,k}^{t} - \eta \bigtriangledown_j \mathcal{L}_{nll}(Y_i, \widehat{Y}_i)_k \end{aligned} \tag{3.8}$$

The loss function is defined as $\mathcal{L}_{nll}(Y_i, \widehat{Y}_i)_k = -\log(\widehat{y}_{i,k})$

The gradient of node i is well-known to be,

$$\begin{aligned} \bigtriangledown_j \mathcal{L}_{nll}(Y_i, \widehat{Y}_i)_k &= \frac{\partial \mathcal{L}_{nll}(Y_i, \widehat{Y}_i)_k}{\partial \widehat{y}_{i,k}} = \frac{\partial \mathcal{L}_{nll}(Y_i, \widehat{Y}_i)_k}{\partial \widehat{y}_{i,k}} \cdot \frac{\partial \widehat{y}_{i,k}}{\partial h_{\mathbf{j},k}^{(L)}} \\ &= -\frac{1}{\widehat{y}_{i,k}} \cdot (\widehat{y}_{i,k}(1 - \widehat{y}_{i,k})(-1)) = 1 - \widehat{y}_{i,k} > 0, \end{aligned} \tag{3.9}$$

Similarly, the gradient of node *j* is given by:

$$\begin{aligned}\bigtriangledown_i \mathcal{L}_{nll}(Y_i, \widehat{Y}_i)_k &= \frac{\partial \mathcal{L}_{nll}(Y_i, \widehat{Y}_i)_k}{\partial \widehat{y}_{i,k}} = \frac{\partial \mathcal{L}_{nll}(Y_i, \widehat{Y}_i)_k}{\partial \widehat{y}_{i,k}} \cdot \frac{\partial \widehat{y}_{i,k}}{\partial h_{i,k}^{(L)}} \\ &= -\frac{1}{\widehat{y}_{i,k}} \cdot (\widehat{y}_{i,k}(1 - \widehat{y}_{i,k})) = \widehat{y}_{i,k} - 1 < 0\end{aligned} \tag{3.10}$$

where we can retrieve Eq. 3.7.

End of proof.

Referring to this analysis, we can induce the expectation of zero-weight GCN as below.

Theorem 4.3 (Binary class, zero-weight GCN). *Similar to the Theorem 4.2, assigning zero weights to the heterophilous edges leads to the following feature distribution:*

$$\mathbb{E}(h_i^{(1)}|y_i, d_i) = \frac{(b_i - e)d_i' + 1}{(d_i + 1)}\mathbb{E}(h_i^{(0)}|y_i) \tag{3.18}$$

Proof of Theorem 4.3.

$$\begin{aligned}\mathbb{E}(h_i^{(1)}|v_i, d_i) &= \frac{\mu}{d_i + 1} + \sum_{j \in \mathcal{N}_i} \left(\frac{\mu(1-e) - \mu e \times 0}{\sqrt{(d_i+1)(d_j+1)}} b_i + \frac{\mu(1-e) \times 0 - \mu e}{\sqrt{(d_i+1)(d_j+1)}} (1 - b_i) \right) \\ &= \frac{\mu}{d_i + 1} + \sum_{j \in \mathcal{N}_i} \left(\frac{b_i - e}{\sqrt{(d_i+1)(d_j+1)}} \mu \right) \\ &= \left(\frac{1}{d_i + 1} + \frac{b_i - e}{d_i + 1} \sum_{j \in \mathcal{N}_i} \frac{\sqrt{d_i + 1}}{\sqrt{d_j + 1}} \right) \mu \\ &= \left(\frac{1 + (b_i - e)d_i'}{d_i + 1} \right) \mu.\end{aligned} \tag{3.19}$$

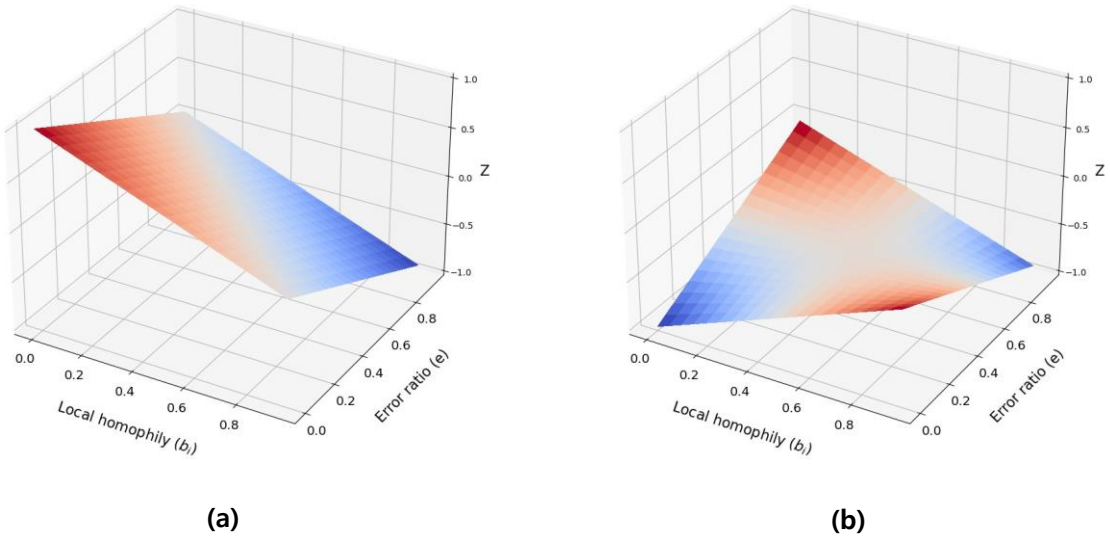
End of proof.

For all theorems, if the coefficient of is smaller than 1, the node feature moves towards the decision boundary and message passing loses its discrimination power [89]. Based on this observation, we can compare the discrimination powers of signed and zero-weight GCNs.

Corollary 4.4 (Binary class, discrimination power). *Omitting the overlapping part of Theorems 4.2 and 4.3, their difference, Z, can be induced by the error ratio (e) and homophily (b_i):*

$$Z = (1 - 2e) - (b_i - e) = 1 - e - b_i, \tag{3.20}$$

where $0 \leq e, b_i \leq 1$.



(a) (b)

Figure 3.2 We plot the Z to compare the discrimination power of signed and zero-weight GCNs. The red and blue colored parts indicate the regions where signed GCN and zero-weight GCN have better performance, respectively.

We visualize Z in Fig. 3.2 (a). Note that the space is half-divided by the plane $Z = 0$ since $\iint_0^1 (1 - e - b) dedb = 0$. When b_i and e are small, Z becomes positive which indicates that signed GCN outperforms zero-weight GCN and vice versa.

Now, let us assume that the error ratio is zero ($e = 0$) identical to the settings of our previous analysis (Fig. 3.1). Under this condition, Z ($= 1 - b_i$) should be non-negative regardless of the homophily ratio ($0 \leq b_i \leq 1$).

$$\int_0^1 \int_0^1 (1 - e - b)\, dedb = \left[1 - \frac{e^2 + b^2}{2}\right]_{e,b=0}^{1} = 0 \qquad (3.21)$$

However, Fig. 3.1 shows that zero-weight GCN generally outperforms signed GCN ($Z \leq 0$) contradicting the Corollary 4.4. Thus, we extend the above theorems to cover a multi-class scenario and point out the limitations in the previous analyses.

3.3.2 Empirical Analysis.

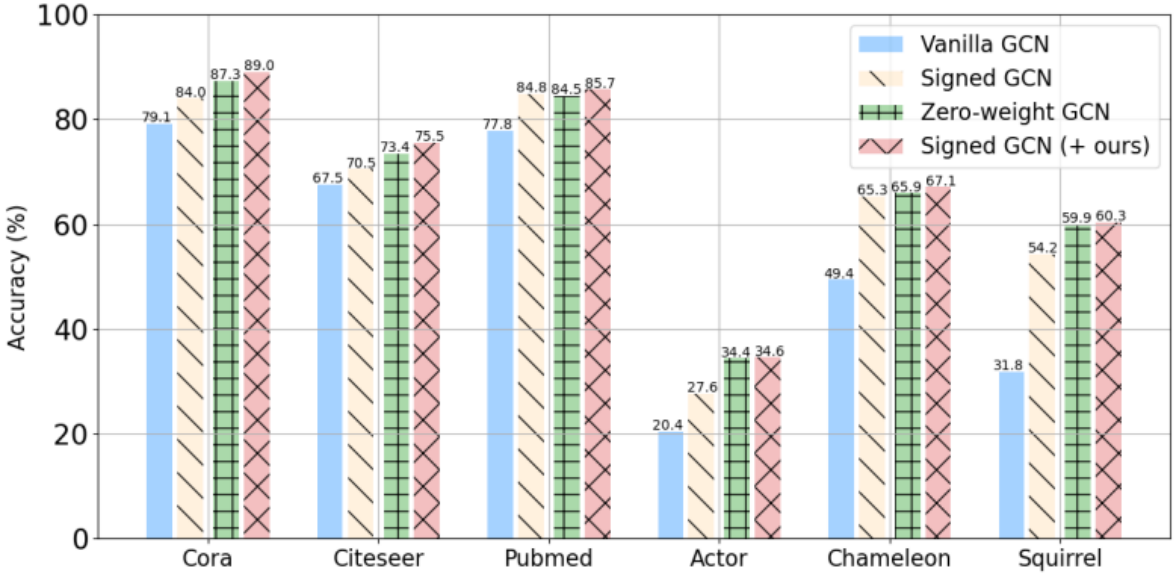


Figure 3.1 Node classification accuracy on six benchmark datasets. Firstly, vanilla GCN utilizes the original graph. The coefficient of heterophilous edges is changed to -1 in signed GCN and to 0 in zero-weight GCN

Vanilla GNNs provide dismal performances in heterophilic datasets, where most edges connect two nodes with different labels. Consequently, finding proper coefficients of entire edges became essential to enhance the overall quality of GNNs.

In Fig. 3.1, we evaluate the node classification accuracy of GCN [42] using six benchmark graphs (the statistical details are shown in Table 1). From the original graph (vanilla GCN), we fabricate two graph variants; one that replaces disassortative edges with -1 (signed GCN), and the other that assigns zero-weights on heterophilous connections (zero-weight GCN). As illustrated in Fig. 3.1, the zero-weight GCN achieves the best performance, followed by the signed GCN. The detailed explanations regarding this phenomenon will be explained in Section 3.5.

3.4.3 Using Signed Messages on Multiple Classes.

Based on the prior analysis, we extend them to multi-class scenarios and point out some drawbacks of using signed propagation for GNNs.

(Message-Passing) The performance of signed GCN depends on the number of classes. Without loss of generality, one can extend the expectation of node features from a binary (Eq. 3.16) to multiple classes through spherical coordinates as below:

$$\mathbb{E}(h_i^{(0)}|y_i) = (\mu, \phi, \theta). \tag{3.22}$$

Here, μ also represents the scale of a vector and the direction is determined by two angles ϕ and θ. Obviously, the above equation satisfies the origin symmetry under binary classes, where (μ, π/2, 0) = −(μ, π/2, π). Through this equation, we can redefine Theorem. 4.1 and 4.2 for multiple class GCNs.

Theorem 4.5 (Multi-class, signed GCN). *Let us assume the label* $y_i = 0$. *For simplicity, we denote the coordinates of the ego* (μ,θ) *as k, and its neighbors* (μ,θ') *as* k', *where* $\theta = 0$ *and* $\theta' = \frac{2\pi j}{c} \neq 0$. *Then, the expectation of* h_i *is defined as:*

$$\mathbb{E}(h_i^{(1)}|y_i, d_i) = \frac{(1-2e)\{b_i k + (b_i - 1)k'\}d_i' + k}{d_i + 1} \quad (3.23)$$

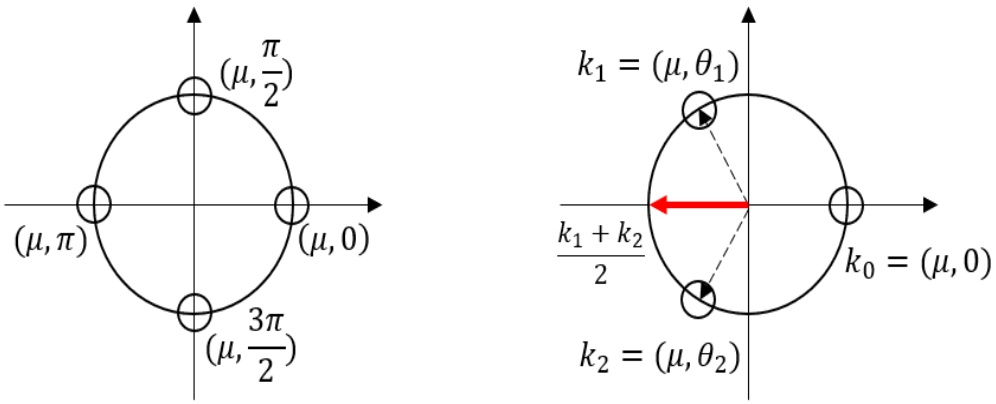


Figure 1.3 We take an example to illustrate the distribution of node features under multi-class scenarios. The right figure represents the aggregation of neighboring nodes (k_1, k_2) under multiple classes.

Proof of Theorem 4.5.

$$\begin{aligned}
\mathbb{E}(h_i^{(1)}|v_i, d_i) &= \frac{k}{d_i+1} + \sum_{j\in\mathcal{N}_i}\left(\frac{k(1-e)-ke}{\sqrt{(d_i+1)(d_j+1)}}b_i + \frac{-k'(1-e)+k'e}{\sqrt{(d_i+1)(d_j+1)}}(1-b_i)\right) \\
&= \frac{k}{d_i+1} + \sum_{j\in\mathcal{N}_i}\left(\frac{k(1-2e)b_i - k'(1-2e)(1-b_i)}{\sqrt{(d_i+1)(d_j+1)}}\right) \\
&= \frac{k}{d_i+1} + \sum_{j\in\mathcal{N}_i}\left(\frac{(1-2e)\{kb_i + k'(b_i-1)\}}{\sqrt{(d_i+1)(d_j+1)}}\right) \\
&= \frac{k}{d_i+1} + \frac{(1-2e)\{kb_i + k'(b_i-1)\}d'}{d_i+1} \\
&= \frac{(1-2e)\{b_i k + (b_i-1)k'\}d_i' + k}{d_i+1}.
\end{aligned} \quad (3.24)$$

End of proof.

As shown in Figure 3.3, we extend a binary classification scenario to a multi-class case. Without loss of generality, we employ spherical coordinates and ensure that μ corresponds to the scale of a vector, while the direction of each vector lies between zero and $\frac{2\pi j}{C}$ with respect to their label j. Here, we assume the label is $y_i = 0$. For simplicity, we replace (μ,$\theta = 0$) as k and (μ,$\theta' \neq \theta$) as k', respectively. Though k' comprises multiple distributions that are proportional to the number of classes, their aggregation always satisfies $|k_{aggr}'| \leq \mu$ since the summation of coefficients $(1 - b_i)$ is lower than 1 and $|k'| \leq \mu$. Referring to Fig. 7c, we can see that $\frac{k_1+k_1}{2} \leq \mu$. Given $b_1 = b_2 = 0.5$, where the aggregation of neighbors always lies in μ. Thus, for brevity, we indicate k'_{aggr} as k' here. Now, we can retrieve the expectation (h_i) of signed GCN as follows.

Theorem 4.6 (Multi-class, zero-weight GCN). *Likewise, the h_i driven by zero-weight GCN is:*

$$\mathbb{E}(h_i^{(1)}|y_i, d_i) = \frac{\{(1-e)b_i k + e(1-b_i)k'\}d_i' + k}{d_i + 1} \tag{3.25}$$

Proof of Theorem 4.6.

$$\begin{aligned}
\mathbb{E}(h_i^{(1)}|v_i, d_i) &= \frac{k}{d_i+1} + \sum_{j\in\mathcal{N}_i}\left(\frac{k(1-e) - ke \times 0}{\sqrt{(d_i+1)(d_j+1)}}b_i + \frac{k'(1-e)\times 0 + k'e}{\sqrt{(d_i+1)(d_j+1)}}(1-b_i)\right) \\
&= \frac{k}{d_i+1} + \sum_{j\in\mathcal{N}_i}\left(\frac{k(1-e)b_i + k'e(1-b_i)}{\sqrt{(d_i+1)(d_j+1)}}\right) \\
&= \frac{k}{d_i+1} + \frac{\{(1-e)b_i k + e(1-b_i)k'\}d_i'}{d_i+1} \\
&= \frac{\{(1-e)b_i k + e(1-b_i)k'\}d_i' + k}{d_i+1}.
\end{aligned} \tag{3.26}$$

End of proof.

Similar to Corollary 4.4, we can compare the separability of the two methods based on their coefficients.

Corollary 4.7 (Multi-class). *The difference of discrimination power (Z) between signed and zero-weight GCN in the multiclass case is:*

$$Z = -eb_i k + (1-e)(b_i - 1)k' \tag{3.27}$$

Then, we can induce the conditional statement as below based on the distribution of aggregated neighbors (k'):

$$Z \in \begin{cases} 1 - e - b_i, & \text{if } k' = -k \\ -2eb - (1 - e - b_i), & \text{if } k' = k. \end{cases} \tag{3.28}$$

Fig. 3.2 (b) plots Z for the multi-class case. The above corollary implies that if the distribution of aggregated neighbor is origin symmetry ($k' = -k$), Z ($= 1-e-b$) becomes identical to the Eq. 3.20. Under this condition, signed propagation might perform well. However, as k' gets closer to k, its discrimination power degrades (Z gets smaller) as shown in the blue areas in Fig. 3.2 (b).

Intuitively, the probability of being $k' = -k$ may decrease as the number of classes increases, which means that the zero-weight GCN generally outperforms the signed GCN in multi-class graphs.

$$\int_0^1 \int_0^1 (-2eb + e + b - 1)\, dedb = \left[\frac{-eb^2 - e^2 b + e^2 + b^2}{2} - 1 \right]_{e,b=0}^{1} = -1 \tag{3.29}$$

(Parameter Update) Though signed propagation contributes to the ego-neighbor separation, it also increases the uncertainty of the predictions. Adequate management of uncertainty is vital in machine learning to generate highly confident predictions [19; 61; 62]. This is closely related to the entropy (e.g., information gain

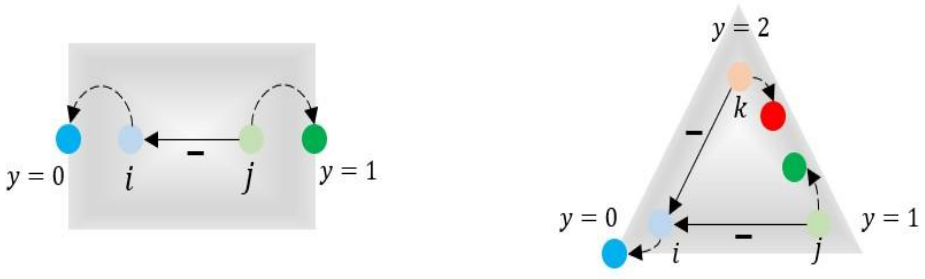
[52]) and recent work [39] formulates two types of uncertainties: the aleatoric and epistemic caused by the data and the model, respectively. But here, we rather focus on the conflict evidence (dissonance) [68; 97], which ramps up the entropy of outputs. One can easily measure the uncertainty of a prediction ( ) using Shannon's entropy [74] as:

$$E(\widehat{y}_i) = -\sum_{j=1}^{C} \widehat{y}_{i,j} log_c \widehat{y}_{i,j}. \tag{3.31}$$

Furthermore, measuring dissonance (*diss*) is also important [97] as it is powerful in distinguishing Out-of-Distribution (OOD) data from conflict predictions [36] and improving classification accuracy:

$$diss(\widehat{y}_i) = \sum_{j=1}^{C} \left(\frac{\widehat{y}_{ij} \sum_{k \neq j} \widehat{y}_{ik} (1 - \frac{|\widehat{y}_{ik} - \widehat{y}_{ij}|}{\widehat{y}_{ij} + \widehat{y}_{ik}})}{\sum_{k \neq j} \widehat{y}_{ik}} \right) \tag{3.32}$$

Figure 3: (a) In binary class graphs, signed propagation contributes to the separation of nodes (*i,j*) and reduces the entropy. (b) In multi-class graphs, the uncertainty of neighboring nodes that are connected with signed edges (*j,k*) increases



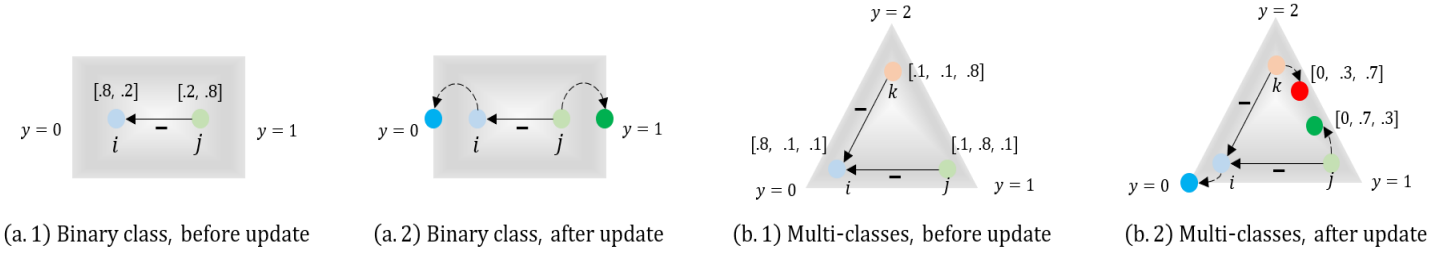
(a) Binary class **(b) Multiple class**

Figure 3.4 (a) In binary class graphs, signed propagation contributes to the separation of nodes (i, j) and reduces the entropy. (b) In multi-class graphs, the uncertainty of neighboring nodes that are connected with signed edges (j, k) increases.

which can be defined only for non-zero elements. We show that signed messages are helpful for ego and neighbor separation. Now, we posit that neighbors connected with signed edges provoke higher entropy (e.g., $E(\hat{y}_i)$ or $diss(\hat{y}_i)$) than the one with a plane or zero-weighted one.

Theorem 4.8. *Under multiple classes, the entropy gap between the signed neighbor* $E(\hat{y}_s)$ *and plane (or zero) one* $E(\hat{y}_p)$ *increases in proportion to the training epoch (t).*

$$E(\widehat{y}_s^{(t+1)}) - E(\widehat{y}_p^{(t+1)}) > E(\widehat{y}_s^{t}) - E(\widehat{y}_p^{t}) \tag{3.33}$$



(a. 1) Binary class, before update (a. 2) Binary class, after update (b. 1) Multi-classes, before update (b. 2) Multi-classes, after update

Figure 3.5 Visualization of the update procedure of node features under (a) binary and (b) multi-class.

Proof of Theorem 4.8.

Firstly, the true label probability (k) of node p ($\hat{y}_{p,k}$) increases, while other probabilities $\hat{y}_{p,o}$ ($o \neq k$) decrease as follows:

$$\widehat{y}_p^{(t+1)} \in \begin{cases} \widehat{y}_{p,k}^{t} - \eta \bigtriangledown_p \mathcal{L}_{nll}(Y_i, \widehat{Y}_i)_k > \widehat{y}_{p,k}^{t}, \\ \widehat{y}_{p,o}^{t} - \eta \bigtriangledown_p \mathcal{L}_{nll}(Y_i, \widehat{Y}_i)_o < \widehat{y}_{p,o}^{t}, \ \forall\ o \neq k. \end{cases} \tag{3.34}$$

Since we proved that $\bigtriangledown_p L_{nll}(Y_i, \hat{Y}_i)_k < 0$, we analyze the partial derivative $\bigtriangledown_p L_{nll}(Y_i, \hat{Y}_i)_o$ ($\forall o \neq k$).

$$\bigtriangledown_p \mathcal{L}_{nll}(Y_i, \widehat{Y}_i)_o = \frac{\partial \mathcal{L}_{nll}(Y_i, \widehat{Y}_i)_o}{\partial \widehat{y}_{p,o}} = \frac{\partial \mathcal{L}_{nll}(Y_i, \widehat{Y}_i)_o}{\partial \widehat{y}_{i,o}} \cdot \frac{\partial \widehat{y}_{i,o}}{\partial h_{p,o}^{(L)}} \tag{3.35}$$

$$= \frac{1}{\widehat{y}_{i,o}} \cdot (\widehat{y}_{i,o}(1 - \widehat{y}_{i,o})) = 1 - \widehat{y}_{i,o} > 0,$$

On the contrary, the gradient of node *s* has a different sign with node *p*, where we can infer that:

$$\widehat{y}_s^{(t+1)} \in \begin{cases} \widehat{y}_{s,k}^t - \eta \bigtriangledown_s \mathcal{L}_{nll}(Y_i, \widehat{Y}_i)_k < \widehat{y}_{s,k}^t, \\ \widehat{y}_{s,o}^t - \eta \bigtriangledown_s \mathcal{L}_{nll}(Y_i, \widehat{Y}_i)_o > \widehat{y}_{s,o}^t, \ \forall \ o \neq k. \end{cases} \tag{3.36}$$

As the training epoch increases, $\hat{y}_{p,k}$ will converge to 1 resulting in the decrease of $E(\hat{y}_p)$. Conversely, $\hat{y}_{s,k}$ gets closer to 0, which may fail to generate a highly confident prediction and leads to a surge of uncertainty. Thus, one can infer:

$$E(\widehat{y}_s^{(t+1)}) - E(\widehat{y}_p^{(t+1)}) > E(\widehat{y}_s^t) - E(\widehat{y}_p^t) \text{ as } t \to \infty \tag{3.37}$$

As shown in Figure 3.5 a, this can be effective under a binary class, while the signed nodes (*i,j*) in a multi-class case (Fig. 3.5 b) have conflict evidence except for class 0. Taking another example, let us assume that the original probability (before the update) is $y_i^t = [0.6,0.2,0.2]$ with $C = 3$. Then, one can calculate the Shannon's entropy as,

$$E(\widehat{y}_i^t) = -\sum_{j=1}^{3} \widehat{y}_{i,j}^t log_3 \widehat{y}_{i,j}^t \approx 0.8649 \tag{3.38}$$

Without considering node degree, let us assume the gradient of class *k* as

$\bigtriangledown_p L_{nll}(Y_i, \hat{Y}_i)_k = - \bigtriangledown_s L_{nll}(Y_i, \hat{Y}_i)_k = \alpha$, and other classes as

$\bigtriangledown_p L_{nll}(Y_i, \hat{Y}_i)_o = - \bigtriangledown_s L_{nll}(Y_i, \hat{Y}_i)_o = \frac{\alpha}{C-1} (\forall o \neq k)$. If we take $\alpha = 0.1$,

then $\hat{y}_p^{(t+1)}$ and $\hat{y}_s^{(t+1)}$ becomes:

$$E(\widehat{y}_p^{(t+1)}) = E([0.8, 0.1, 0.1]) \approx 0.5817, \ E(\widehat{y}_s^{(t+1)}) = E([0.4, 0.3, 0.3]) \approx 0.9911 \tag{3.39}$$

where we can see that $E(\hat{y}_p^{(t+1)}) < E(\hat{y}_s^{(t+1)})$ after the single iteration.

End of proof.

To summarize, signed messages contribute to the separation of two nodes (Fig 3.5 a), while they also increase the uncertainty of neighboring nodes j, k that propagate signed information to an ego i (Fig. 3.5 b). To deal with this, we employ confidence calibration which will be explained below.

3.5 Methodology

Previously, we pointed out the issues of signed propagation from two perspectives: message-passing and parameter update. Now, we propose two strategies that can be combined with any GNN using signed propagation.

3.5.1 (Message-Passing) Edge weight calibration

Through Corollary 4.7, we analyze the impact of signed propagation based on the distribution of neighbors k'. It was observed that as k' is similar to k, signed propagation reduces the discrimination power. At the same time, it also increases the separability of two nodes during training. Thus, we propose the following strategies: (1) In training, we employ signed messages for ego-neighbor separation. (2) During the validation/test phase, we block the information propagation of highly similar nodes, which may decrease the discrimination power.

As a downstream task of GNNs, the score (e.g., cosine similarity) of two nodes is calculated based on the node features at the l-th layer:

$$\cos(h_i, h_j) = \frac{h_i \cdot h_j}{\|h_i\|_2 \|h_j\|_2}. \tag{3.40}$$

Then, for all edges that satisfy the following conditions, we replace their weights a_{ij} (e.g., attention values) as 0:

$$a_{ij} = \begin{cases} 0 & \text{if } a_{ij} < 0 \wedge \cos(h_i^l, h_j^l) > \epsilon \\ a_{ij} & \text{otherwise} \end{cases} \quad \forall (i,j) \in \mathcal{E}, \tag{3.41}$$

where ε is the hyper-parameter. Through this, the discrimination power remains powerful during training, while securing the separability in the inference phase.

Remark. The replacement of a_{ij} in Eq. 41 is highly scalable for signed GNNs, which focuses on edge-level weight retrieval.

3.5.2 (Parameter Update) Confidence calibration

In Theorem 4.8, we show that signed messages increase the uncertainty of predictions. Here, we propose a simple yet effective solution that can reduce the uncertainty (P2) through confidence calibration. The proposed method, free from entire path configuration, is cost-efficient and fairly powerful. Calibration is one type of self-training method [29; 90] that acts as a regularization term. Even though it has shown to be effective for generic GNNs [84], we notice that the performance gain is much greater when integrated with signed methods. Many algorithms can be used for calibration (e.g., temperature and vector scaling [29]). In this paper, our loss function is defined as,

$$\mathcal{L}_{conf} = \frac{1}{n}\sum_{i=1}^{n}(-max(\widehat{y}_i) + submax(\widehat{y}_i)), \quad (3.42)$$

where $n = |V_{valid} \cup V_{test}|$ is the set of validation and test nodes. Our method is quite similar to prior work [84], but we do not utilize the label of validation sets for a fair comparison. As defined above, it penalizes the maximal and sub-maximal values to be similar in order to suppress the generation of conflict evidence. Since the calibration only utilizes the outputs *y*, it has high b scalability and is applicable to any type of GNNs.

3.5.3 Optimization

Before, we introduce two strategies to improve the quality of signed propagation. For optimization, we apply confidence calibration during training as below:

$$\mathcal{L}_{total} = \mathcal{L}_{GNN} + \lambda\mathcal{L}_{conf}. \quad (3.43)$$

Here, $\mathcal{L}_{GNN}$ indicates any type of GNN. Also, the λ is a hyper-parameter that balances the influence of confidence calibration. After optimization, we employ edge weight calibration during the inference phase. Through this, we observe a significant improvement in signed GCN (+ calib) as demonstrated in Figure 1. We describe the pseudo-code of calibrated FAGCN [2] below.

Algorithm 1 Pseudo-code of calibrated FAGCN

Require: Adjacency matrix (A), initial node features (X), node embedding at l^{th} layer (H^l), attention weight between two nodes (a_{ij}), initialized parameters of FAGCN (θ), edge weight threshold (ϵ), best validation score ($\alpha^* = 0$)

Ensure: Parameters with the best validation score (θ^*)

for training epochs **do**
  # Training (plane forward pass with confidence calibration)
  Retrieve class probability through forward pass, $\widehat{Y} = \sigma(\bar{H}^{(L)})$
  Compute negative-log likelihood loss, $\mathcal{L}_{FAGCN} = \mathcal{L}_{nll}(Y, \widehat{Y})$
  Compute calibration loss, $\mathcal{L}_{conf} = \frac{1}{n}\sum_{i=1}^{n}(-max(\widehat{y}_i) + submax(\widehat{y}_i)), \quad n = |\mathcal{V}_{valid} \cup \mathcal{V}_{test}|$
  Get total loss, $\mathcal{L}_{total} = \mathcal{L}_{FAGCN} + \lambda\mathcal{L}_{conf}$
  Update parameters, $\theta' = \theta - \eta\frac{\partial\mathcal{L}_{total}}{\partial\theta}$
  # Validation (forward pass with edge weight calibration)
  Retrieve l^{th} layer's node embedding, H^l
  Get node $i's$ embedding, h_i^l
  Get attention weights a_{ij} of adjacent nodes (j), $a_{ij} = tanh(g^T\left[h_i^l || h_j^l\right])$ * g^T: learnable vector
  Calculate cosine similarity, $\cos(h_i^l, h_j^l) = \frac{h_i^l \cdot h_j^l}{\|h_i^l\|_2 \|h_j^l\|_2}$.
  if $a_{ij} < 0 \wedge \cos(h_i^l, h_j^l) > \epsilon$, **then**
    Replace a_{ij} as 0
  Using the updated parameters (θ') and calibrated attention weights (a_{ij}), get validation score α
  if $\alpha > \alpha^*$ **then**
    Save the parameters, $\theta^* = \theta'$
    Update best validation score, $\alpha^* = \alpha$

Time complexity of calibrated GNN

We analyze the computational complexity of our method. For brevity, we take vanilla GCN [32] as a base model. Generally, the cost of GCN is known to be proportional to the number of edges and trainable parameters $O(|\mathcal{E}|\theta_{GCN})$. Here, θ_{GCN} is comprised of $O(nz(X)F' + F'C)$ [85], where $nz(\cdot)$ represents the non-zero elements of inputs and F' stand for the hidden dimension, and C is the number of classes. Additionally, our method employs two types of calibration. The first one is

edge weight calibration. For this, we need to retrieve the node features of each layer and calculate the cosine similarity for all connected nodes $|\mathcal{E}|^2$. Thus, the complexity becomes $O(|\mathcal{E}|\theta_{GCN} + L|\mathcal{E}|^2)$. Further, the calibration takes $n = |V_{valid} \cup V_{test}|$ samples as inputs and finds top k samples on each row of y_b. Thus, their complexity can be simply defined as $O(n + k)$. To summarize, the cost of calibrated GCN is $O(2|\mathcal{E}|\theta_{GCN} + L|\mathcal{E}|^2 + n + k)$, which is fairly efficient.

3.6 Experiments

We conducted extensive experiments to validate our theorems and to compare the performances of our method and baselines. We aim to answer the following research questions:

- Q1 Is calibration alleviates the uncertainty issue when integrated with the signed GNNs?
- Q2 Do the signed messages increase the uncertainty of the final prediction?
- Q3 How much impact do the two calibration methodologies have on performance improvement?
- Q4 Is the number of classes correlated with the prediction uncertainty?
- Q5 How does the hyper-parameter λ in Eq. 3.41 affect the performance?

Datasets. The statistical details of datasets are in Table 3.1. (1) Cora, Citeseer, Pubmed [42] are citation graphs, where a node corresponds to a paper and edges are citations between them. The labels are the research topic of the papers. (2) Actor [80] is a co-occurrence graph where actors and co-occurrences in the same movie are represented as nodes and edges, respectively. The labels are five types of actors. (3) Chameleon, Squirrel [72] are Wikipedia hyperlink networks. Each node is a web page and the edges are hyperlinks. Nodes are categorized into five classes based on monthly traffic.

Baselines. We employ several state-of-the-art methods for validation: (1) Plane GNNs: GCN [42], and APPNP [43]. (2) GNNs for heterophilous graphs: GAT [81], GCNII [9], H2GCN [99], and PTDNet [55]. (3) GNNs with signed propagation: GPRGNN [13], FAGCN [2], and GGCN [89].

- General information of the implementations.

All methods including baselines and ours are implemented upon *PyTorch Geometric*. For a fair comparison, we equalize the hidden dimension of the entire methodologies as 64. ReLU with dropout is used for non-linearity and to prevent over-fitting. We employ the log-Softmax as a cross-entropy function. The learning ratio is set to $1e^{-3}$ and the Adam optimizer is taken with weight decay $5e^{-4}$. For training, 20 nodes per class are randomly chosen and the remaining nodes are equally divided into two parts for validation and testing.

- More details about baseline methods.

- GCN [42] is a first-order approximation of Chebyshev polynomials [11]. For all datasets, we simply take 2 layers of GCN.
- APPNP [43] combines personalized PageRank on GCN. We stack 10 layers and set the teleport probability (α) as $\{0.1,0.1,0.1,0.5,0.2,0.3\}$ for Cora, Citeseer, Pubmed, Actor, Chameleon, and Squirrel.
- GAT [81] calculates feature-based attention for edge coefficients. Similar to GCN, we construct 2 layers of GAT. The pair of (hidden dimension, head) is set as (8, 8) for the first layer, while the second layer is (1, number of classes).
- GCNII [9] integrates an identity mapping function on APPNP. We set $\alpha = 0.5$ and employ nine hidden layers. We increase the weight of identity mapping (β) that is inversely proportional to the heterophily of the dataset.

- H_2GCN [99] suggests the separation of ego and neighbors during aggregation. We refer to the publicly available *source code*[1] for implementation.
- PTDNet [55] removes disassortative edges before a message-passing. We also utilize the open-*source code*[2] here.
- GPRGNN [13] generalized the personalized PageRank to deal with heterophily and over-smoothing. Referring *here*[3], we tune the hyper-parameters based on the best validation score for each dataset.
- FAGCN [2] determines the sign of edges using the node features. We implement the algorithm referring *here*[4] and also tune the hyper-parameters with respect to their accuracy.
- GGCN [89] proposes the scaling of degrees and the separation of positive/negative adjacency matrices. We simply take the publicly available *code*[5] for evaluation.

Datasets	Cora	Citeseer	Pubmed	Actor	Cham.	Squirrel
# Nodes	2,708	3,327	19,717	7,600	2,277	5,201
# Edges	10,558	9,104	88,648	25,944	33,824	211,872
# Features	1,433	3,703	500	931	2,325	2,089
# Labels	7	6	3	5	5	5

Table 3.1 Statistical details of six benchmark datasets.

3.6.1 Experimental Results (Q1).

In Table 3.2, we describe the node classification accuracy of each method. A symbol (‡) means that calibration is supplemented to the basic method. Now, let us analyze the results from two perspectives.

[1] https://github.com/GemsLab/H2GCN
[2] https://github.com/flyingdoog/PTDNet
[3] https://github.com/jianhao2016/GPRGNN
[4] https://github.com/bdy9527/FAGCN
[5] https://github.com/Yujun-Yan/Heterophily and oversmoothing

Datasets	Cora	Citeseer	Pubmed	Actor	Chameleon	Squirrel
$\mathcal{H}_g$ (Eq. 1)	0.81	0.74	0.8	0.22	0.23	0.22
GCN	79.0 ± 0.6 (0.17)	67.5 ± 0.8 (0.29)	77.6 ± 0.2 (0.53)	20.2 ± 0.4 (0.29)	49.3 ± 0.5 (0.19)	31.7 ± 0.7 (0.31)
GCN‡	81.0 ± 0.9 (0.12)	71.3 ± 1.2 (0.14)	77.8 ± 0.4 (0.38)	21.7 ± 0.6 (0.62)	49.4 ± 0.6 (0.25)	31.5 ± 0.6 (0.58)
GAT	80.1 ± 0.6 (0.22)	68.0 ± 0.7 (0.25)	78.0 ± 0.4 (0.45)	22.5 ± 0.3 (0.28)	47.9 ± 0.8 (0.17)	30.8 ± 0.9 (0.27)
GAT‡	81.4 ± 0.4 (0.12)	72.2 ± 0.6 (0.08)	78.3 ± 0.3 (0.39)	23.2 ± 1.8 (0.43)	49.2 ± 0.4 (0.16)	30.3 ± 0.8 (0.40)
APPNP	81.3 ± 0.5 (0.15)	68.9 ± 0.3 (0.21)	79.0 ± 0.3 (0.42)	23.8 ± 0.3 (0.49)	48.0 ± 0.7 (0.34)	30.4 ± 0.6 (0.69)
GCNII	81.1 ± 0.7 (0.08)	68.5 ± 1.4 (0.13)	78.5 ± 0.4 (0.20)	25.9 ± 1.2 (0.43)	48.1 ± 0.7 (0.21)	29.1 ± 0.9 (0.24)
H_2GCN	80.6 ± 0.6 (0.16)	68.2 ± 0.7 (0.22)	78.5 ± 0.3 (0.29)	25.6 ± 1.0 (0.34)	47.3 ± 0.8 (0.19)	31.3 ± 0.7 (0.62)
PTDNet	81.2 ± 0.9 (0.24)	69.5 ± 1.2 (0.42)	78.8 ± 0.5 (0.44)	21.5 ± 0.6 (0.33)	50.6 ± 0.9 (0.17)	32.1 ± 0.7 (0.34)
PTDNet†	81.9 ± 0.6 (0.20)	71.1 ± 0.8 (0.31)	79.0 ± 0.2 (0.38)	22.7 ± 0.6 (0.19)	**50.9** ± 0.3 (0.15)	**32.3** ± 0.5 (0.30)
GPRGNN	82.2 ± 0.4 (0.25)	70.4 ± 0.8 (0.43)	79.1 ± 0.1 (0.26)	25.4 ± 0.5 (0.55)	49.1 ± 0.7 (0.25)	30.5 ± 0.6 (0.36)
GPRGNN‡	**84.5** ± 0.2 (0.04)	73.2 ± 0.5 (0.05)	**80.0** ± 0.2 (0.11)	**27.7** ± 1.3 (0.33)	50.7 ± 0.4 (0.18)	31.6 ± 0.4 (0.16)
FAGCN	80.9 ± 0.5 (0.15)	69.8 ± 0.6 (0.17)	79.0 ± 0.5 (0.31)	25.2 ± 0.8 (0.66)	46.5 ± 1.1 (0.25)	30.4 ± 0.4 (0.64)
FAGCN‡	83.7 ± 0.4 (0.09)	**73.7** ± 0.5 (0.08)	79.7 ± 0.2 (0.16)	27.3 ± 0.5 (0.42)	48.6 ± 0.7 (0.13)	31.3 ± 0.5 (0.37)
GGCN	80.0 ± 1.2 (0.38)	69.7 ± 1.6 (0.30)	78.2 ± 0.4 (0.47)	22.5 ± 0.5 (0.47)	48.5 ± 0.7 (0.15)	30.2 ± 0.7 (0.40)
GGCN‡	83.4 ± 0.8 (0.07)	73.1 ± 0.4 (0.05)	78.7 ± 0.3 (0.29)	24.1 ± 0.4 (0.26)	49.8 ± 0.4 (0.07)	30.8 ± 0.6 (0.15)

Table 3.2 (Q1) Mean node classification accuracy (%) with standard deviation. A shadowed grid indicates the best performance. Values in bracket stand for the dissonance defined in Eq. 3.32 and symbol ‡ means that calibration is applied to baseline method.

Homophily ratio plays an important role in GNNs. Three citation networks have higher homophily compared to others. We can see that all methods perform well under homophilic datasets. As homophily decreases, methods that adjust weights depending on associativity outperform plain GNNs. Similarly, using signed messages (GPRGNN, FAGCN, and GGCN) has shown to be effective here. They achieve notable performance for both homophilic and heterophilic datasets, which means the separation of ego and neighbors (H_2GCN) is quite important.

Calibration improves the overall quality and alleviates uncertainty. We apply calibration (‡) to signed GNNs (GPRGNN, FAGCN, and GGCN). We also apply calibration to GCN and GAT. The average improvements of three signed GNNs by calibration are 4.37%, 3.1%, and 3.13%, respectively. The improvements are greater than those of GCN‡ (2.65%) and GAT‡ (1.97%). Additionally, we describe the dissonance (Eq. 21) of each method in a bracket, where the calibrated methods show lower values than the corresponding vanilla model. To summarize, the results indicate

that calibration not only contributes to reducing uncertainty but also improves the accuracy of signed GNNs significantly.

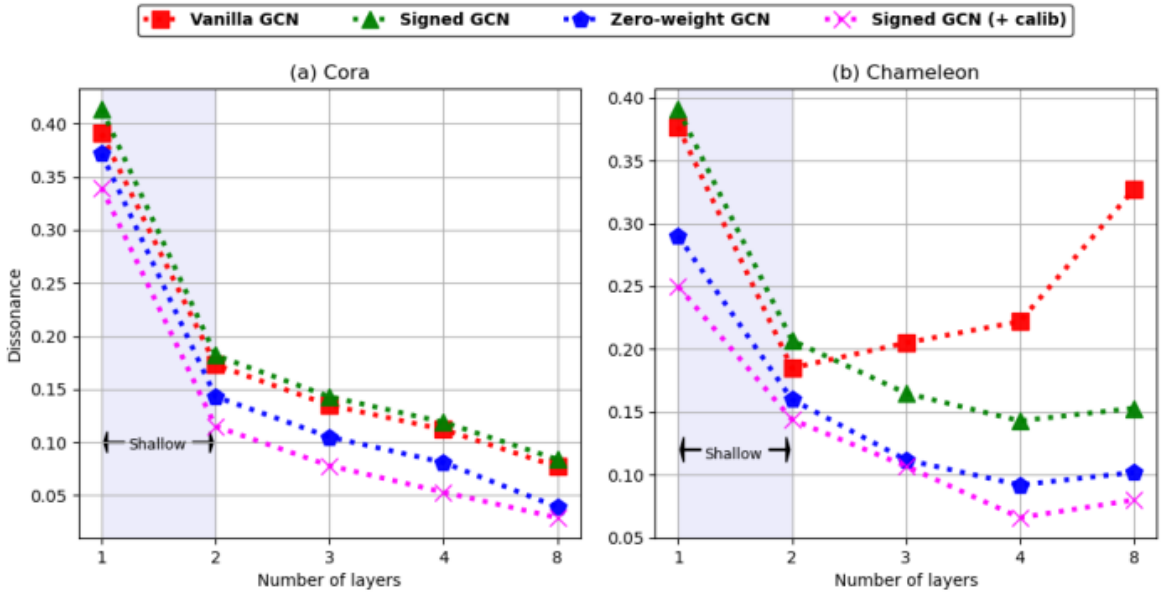


Figure 3.6 (Q2) Comparison of the dissonance on three graph variants; vanilla GCN, signed GCN, and zero-weight GCN.

3.6.2 Correlation of using Signed Messages and the Uncertainty (Q2).

To show that signed messages increase uncertainty, we assume three types of graphs for GCN [42] using four datasets. Specifically, we fabricate two graph variants, signed GCN and zero-weight GCN. Here, we remove the randomness for a fair comparison. The results are illustrated in Fig. 3.6, where the x-axis is the number of layers and the y-axis represents dissonance. Referring to Theorem 4.8, the uncertainty is higher on signed GCN for all shallow layers. As we stack more layers, the entropy of vanilla GCN increases dramatically on heterophilous datasets, the Chameleon and Squirrel. In other words, plain GCN fails to discriminate the ego and neighbors (over-smoothing) and yields low classification accuracy.

Datasets	Cora	Citeseer	Actor	Chameleon
GPRGNN	82.2 ± 0.4	70.4 ± 0.8	25.4 ± 0.5	49.1 ± 0.9
w/ edge calib	83.0 ± 0.6	71.1 ± 1.0	25.9 ± 0.6	49.7 ± 0.7
w/ conf calib	**83.8** ± 0.5	72.6 ± 0.5	**27.5** ± 0.4	**50.2** ± 0.3
FAGCN	80.9 ± 0.5	69.8 ± 0.6	25.2 ± 0.8	46.5 ± 1.1
w/ edge calib	82.1 ± 0.6	71.6 ± 0.7	25.8 ± 0.7	46.9 ± 1.0
w/ conf calib	83.0 ± 0.3	**72.9** ± 0.5	26.3 ± 0.4	48.0 ± 0.8
GGCN	80.0 ± 1.2	69.7 ± 1.6	22.5 ± 0.5	48.5 ± 0.7
w/ edge calib	81.9 ± 1.0	71.0 ± 1.1	23.3 ± 0.6	48.9 ± 0.7
w/ conf calib	82.7 ± 0.9	72.5 ± 0.6	23.6 ± 0.4	49.6 ± 0.5

Table 3.3 (Q3) We measure the improvement of node classification accuracy (%) by applying edge weight and confidence calibration on three baseline methods

3.6.3 Ablation study (Q3).

We conduct an ablation study to analyze the effectiveness of edge weight and confidence calibration. As shown in the above Table, given the two homophilic (Cora, Citeseer) and heterophilic (Actor, Chameleon) graphs, we employ three baseline methods with signed propagation. For each method, we apply edge weight calibration (\textbf{w/ edge calib}) or confidence calibration (\textbf{w/ conf calib}) and measure the node classification accuracy, respectively. Here, we can see that methods with confidence calibration generally outperform edge weight calibration and shows smaller variance. The reason is that confidence calibration reduces the uncertainty of the entire nodes during training, whereas edge calibration is only applied to a small number of edges for testing. Nevertheless, it can also be observed that edge calibration for similar nodes belonging to different classes still contributes to a certain extent to the improvement in performance.

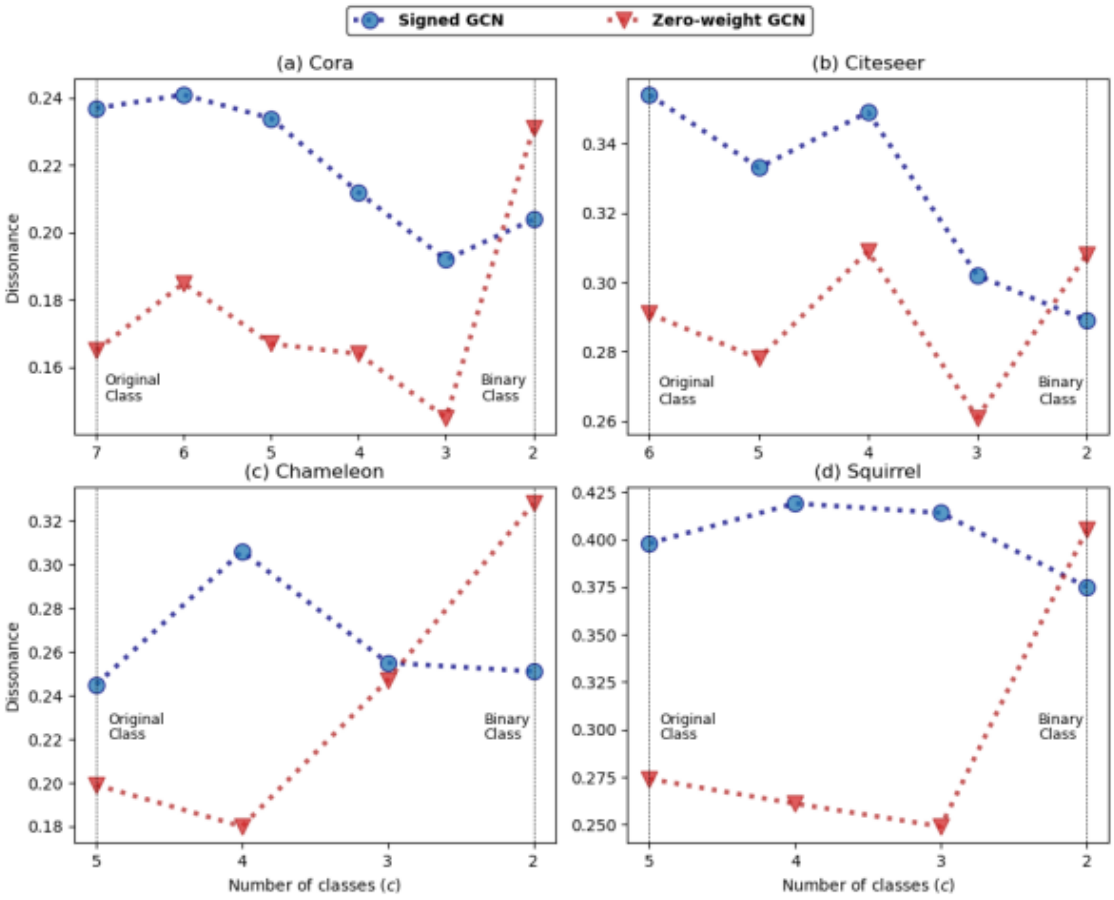


Figure 3.7 (Q4) By differentiating the number of classes, we compare the dissonance of GCN using two graph variants.

3.6.3 Case Study (Q4).

Theoretical analyses confirm that signed messages increase the uncertainty in multi-class graphs (§ 4.3). They have shown to be effective when k' gets closer to $-k$, but this probability is inversely proportional to the number of classes c. To further analyze this phenomenon, we compare the dissonance of two variants of GCN (signed GCN and zero-weight GCN) by decrement of the number of classes (c). Specifically, if the original data contains seven classes (e.g., Cora), we remove all the nodes that belong to the rightmost class to generate a graph with six labels. The results are illustrated in

Fig. 3.7. As can be seen, zero-weight GCN (red) tends to have lower dissonance under multiple classes. However, under binary classes ($c=2$), signed GCN (blue) shows lower uncertainty with the aid of ego-neighbor separation. In the binary case, zero-weight GCN only utilizes homophilous neighbors and fails to generalize under this condition.

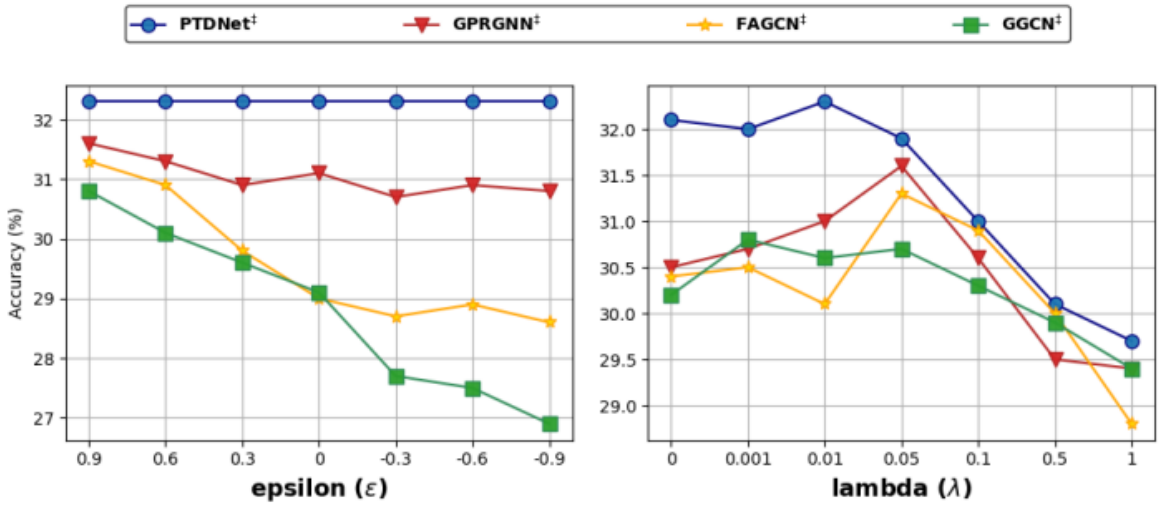


Figure 3.8 (Q5) The effect of hyper-parameter λ in Eq. 3.41 on the classification accuracy of four calibrated methods.

3.6.4 Hyper-parameter Analysis (Q5).

We conduct an experiment to investigate the effect of hyper-parameter ε and λ. We tune the epsilon (threshold of cosine similarity) from -1 to 1 and lambda (impact of confidence calibration) from 0 to 1 as shown in Figure 8. Then, we describe the node classification accuracy on the Squirrel (heterophilic) dataset. The blue line represents PTDNet, while others are signed GNNs. In the left figure, we notice that baseline methods achieve the best performance under the largest epsilon, which means blocking the signed messages of highly similar nodes is advantageous. Here, PTDNet does not change the sign of edges, it shows no variation based on the epsilon. In the right figure, it is notable that finding an appropriate lambda is beneficial for overall performance improvement. Nonetheless, it is also limited by the inherent low capability of base models in heterophilous graphs (low accuracy). Further, assigning

the same weights to $\mathcal{L}_{GNN}$ and $\mathcal{L}_{conf}$ generally downgrades the overall performance, which necessitates the usage of validation sets.

3.7 Conclusion

In this work, we provide a new theoretical perspective on using signed messages for node embedding under multi-class benchmark datasets. Firstly, we show that signed messages contribute to the separation of heterophilous neighbors in a binary class, which is consistent with conventional studies. Then, we extend previous theorems to a multi-class scenario and point out two critical limitations of using signed propagation: (1) it decreases the separability of two nodes, while (2) increasing the probability of generating conflict evidence. Based on the observations, we calibrate signed GNNs to reduce uncertainty and secure robustness. Through experimental analysis, we show that our method is beneficial for both homophilic and heterophilic graphs. We claim that our theorems can provide insights to develop a better aggregation scheme for future GNN studies.

Chapter 4

Finding Heterophilic Neighbors via Confidence-based Subgraph Matching

Graph Neural Networks (GNNs) have proven to be powerful in many graph-based applications. However, they fail to generalize well under heterophilic setups, where neighbor nodes have different labels. To address this challenge, we employ a confidence ratio as a hyper-parameter, assuming that some of the edges are disassortative (heterophilic). Here, we propose a two-phased algorithm. Firstly, we determine edge coefficients through subgraph matching using a supplementary module. Then, we apply GNNs with a modified label propagation mechanism to utilize the edge coefficients effectively. Specifically, our supplementary module identifies a certain proportion of task-irrelevant edges based on a given confidence ratio. Using the remaining edges, we employ the widely used optimal transport to measure the similarity between two nodes with their subgraphs. Finally, using the coefficients as supplementary information on GNNs, we improve the label propagation mechanism which can prevent two nodes with smaller weights from being closer. The experiments on benchmark datasets show that our model alleviates over-smoothing and improves performance.

4.1 Introduction

The investigation of graph-structured data has gained significant attention in various fields; physics [28], protein-protein interactions [26], and social networks [23]. Integrated with deep neural networks (DNNs) [45], graph neural networks (GNNs) have achieved state-of-the-performance by concurrently modeling node features and network structures [73; 16; 42; 32; 81]. Specifically, the message passing plays an important role by aggregating features from neighboring nodes [28]. Consequently, GNNs often have shown the best performance in various tasks including semi-supervised node classification and link prediction.

However, recent studies reveal that GNNs gain advantages of message passing under limited conditions, e.g., high assortativity of subject networks [59]. In this paper, we assume two types of networks; homophilic (assortative) ones where most edges connect two nodes with the same label, and heterophilic graphs where the most connections are disassortative. Most prior work on GNNs assumes that connected nodes are likely to possess the same label, and thus, they fail to attain sufficient performance for many real-world heterophilic datasets [66]. Many clever schemes have been introduced to solve the problem. Some of them specify different weights for each connection [81; 92; 2; 40], or remove disassortative edges [94; 22; 55]. Others employ distant nodes with similar features [67; 93; 37] or apply different aggregation boundary based on the central nodes [85]. Nonetheless, there is a question to be addressed: *is it necessary to specify different weights for GNNs?*

(a) **(b)** **(c)**

Figure 4.1 Node classification accuracy (%) of GCN on different datasets; (a) Cora, and (b) Chameleon. For each graph, we randomly prune a certain proportion of assortative / disassortative edges and plot their performance. We also describe a special case of (c) helpful aggregation scenario under disassortative graphs.

To answer the above question, we conduct an investigation using two representative datasets; one is an assortative citation network called Cora [58], and the other one is Chameleon [72] which contains many disassortative links between Wikipedia web pages. In Figure 4.1, we randomly prune a certain ratio of assortative / disassortative edges and describe the node classification accuracy of GCN [42]. Through this study, we observe two characteristics; (1) for Cora, the performance increases as the assortative edges are maintained, while disassortative edges are removed. On the contrary, Chameleon data is rather heterophilic and thus, the disassortative links play an important role as the number of remaining assortative edges becomes smaller. To analyze this result, we take Figure 4.1 (c) as an example. Though the graph is heterophilic, two central nodes share the same types of neighborhoods (2 green, 1 yellow) that can contribute to distinguishing them from others [54; 57]. (2) the removal of assortative edges has a greater impact on the overall performance than disassortative ones. For example, the performance in Cora using the original graph is 79.8 % (top right). If we remove all disassortative edges, it attains 88.1 % (top left), whereas eliminating assortative links becomes 51.1 % (bottom right). To summarize,

we conjecture that removing a small proportion of assortative edges can be harmful, and thus, assigning accurate weights are fundamental for GNNs. Now, the problem is; how can we figure out these coefficients correctly and utilize them?

To achieve this, we focus on the GAM [78] that suggests a supplementary module with label propagation. Specifically, the supplementary module of GAM only utilizes a central node to debilitate noises. However, referring to Figure 4.1, excluding all links of assortative and disassortative shows the lowest performance, which is the same as GAM's method. To solve this limitation, our supplementary module focuses on the widely used optimal transport [69; 87; 60; 44] to measure similarity between two subgraphs. In addition, we further apply a confidence ratio to deal with multiple disassortative links. Then, considering these predictions as supplementary edge coefficients, we apply label propagation [6] between a certain proportion of high confident edges, while the others are considered disassortative and the connected nodes are prevented from being similar. Our contributions can be summarized as follows:

- We introduce a confidence-based subgraph matching to retrieve edge coefficients accurately. Our model is scalable and generalizes well for both homophilic / heterophilic graphs, which can be achieved by varying the values of the confidence ratio.
- Assuming that a certain proportion of entire edges are disassortative, we improve the label propagation to keep two nodes with a lower similarity score from being closer. Specifically, we divide the edge coefficients into two parts, which can guide the positive pairs to be similar and vice versa.
- We conduct extensive experiments on publicly available datasets to validate the above suggestions. The ablation studies indicate the superiority of subgraph matching techniques for retrieving class sharing probability.

4.2 Related Work

Graph neural networks (GNNs) have shown substantial improvement for semi-supervised classification tasks. Most of them can be categorized into two types; spectral-based and spatial-based methods. The first one utilizes structural information of the entire graph through Laplacian decomposition [33] that requires high computational costs O(n3). To reduce their complexity, GCN [42] suggests a first-order approximation of Chebyshev polynomials [16] and utilizes features of neighboring nodes by simply stacking convolutional layers. Ada-GNN [21] further employs an adaptive frequency filter to capture different perspectives of nodes. However, these algorithms inevitably aggregate noisy adjacent nodes, where they assume two connected nodes are likely to share the same label.

Recently, some algorithms focus on the retrieval of edge coefficients using the node features. For example, GAT [81] measures the relevance between two nodes by applying an attention layer to their features. Similarly, Masked-GCN [92] estimates attribute-wise similarity for precise propagation. GNN-Explainer [94] identifies the set of important edges and features that maximize the mutual information of the final prediction. Nonetheless, these methods may fail to generalize well under a heterophilic graph, where the message passing inevitably makes two connected nodes similar.

To solve this problem, FAGCN [2] selects whether to propagate low-frequency or high-frequency signals by enabling edges to have negative coefficients. L2Q [85] parameterizes the aggregation boundary of each node to deal with heterophily. SuperGAT [40] differentiates between friendly and noisy neighbors based on their homophily and node degrees. However, these methods also implicate noisy information since they work as a downstream task of GNNs. Some argue that graph

sparsification [98; 18] is considerable for graph denoising. For example, PTDNet [55] adopts nuclear norm to prune edges between communities. Yet, it also implicates risk for pruning positive edges and is not powerful enough for classification compared to classical GNNs.

As another branch, non-local neural networks [83; 53] have gained increasing attention for capturing long-range dependencies. Since previous GNNs only utilize local adjacent nodes, they fail to deal with heterophilic graphs. Instead of directly specifying coefficients, finding distant but similar nodes has increased the representational power of GNNs. Specifically, Geom-GCN [67] further exploits distant nodes within a specific boundary and executes grid-based aggregation. Simp-GCN [37] mixes the original adjacency matrix with a feature-based similarity matrix through learnable parameters. Nonetheless, they implicate two limitations. Firstly, operating as a downstream task of GNNs may inevitably contain noisy information after aggregation. Secondly, measuring relevance between two nodes can be biased (or risky) under a semi-supervised setting that has few labeled samples [52].

Apart from retrieving edge coefficients, a strategy for utilizing this information is also considerable. For example, P-reg [90] simply utilizes entire edges to provide additional information for GNNs. NGM [6] integrates label propagation (LP) with GNNs, while GAM [78] further parameterize edge coefficients. However, these methods are highly localized and fail to discriminate less important edges under the global aspect. Further, they show limited performance for precise prediction under our experiments. Instead, we focus on pairwise matching between two subgraphs that are independent of GNN modules. Using the mechanism of optimal transport (OT) [86; 44; 60], we integrate a confidence-based denoising network to secure robustness, followed by our label propagation.

4.3 Notations

Please refer to the definition of notations in Section 2.

4.4 Methodology

Figure 4.2 illustrates the overall architecture of our model which consists of two parts. On the right side, we describe the GNN module with label propagation which takes the predicted edge weights for training. The left one stands for the subgraph matching that provides edge coefficients as supplementary information.

The two modules do not share loss or parameters and are updated independently. In Section 4.4.1, we first introduce methodologies for retrieving edge coefficients, followed by our subgraph matching module. In Section 4.4.2, we suggest strategies to utilize these predictions effectively through label propagation.

4.4.1 Retrieving Edge Coefficients.

Recently, many efforts have been dedicated to specifying edge coefficients, and we categorize them into two types. Firstly, in Section 4.4.1.1, we take previous methods that only utilize central nodes for classification. Secondly, in Section 4.4.1.2, we describe previous algorithms that further utilize the adjacent nodes for prediction. Finally, we discuss the advantages and limitations of these methods and describe our subgraph matching module in Section 4.4.1.3.

4.4.1.1 Retrieving Edge Coefficients using a Central Node.

These types of methods include message passing, but only a central node is used for similarity measure, not a subgraph. With the slight abuse of notation, let us assume the h_i, h_j as hidden representations of two nodes i, j.

Graph agreement model (GAM) [78] introduces an auxiliary model to predict a same class probability w_{ij} between two nodes i, j as below:

$$w_{ij} = MLP((h_i - h_j)^2). \tag{4.1}$$

The *MLP* is a fully-connected network with non-linear activation. GAM works well under the heterophilic graph since they do not utilize neighboring nodes. However, as the homophily ratio of the graph increases, we notice that they show significantly lower performance even compared to the plain GCN [42].

Graph attention network (GAT) [81] applies layer-wise attention as a downstream task of GNN as below:

$$w_{ij}^l = \frac{exp(\sigma(a^l[h_i^l||h_j^l]))}{\sum_{k\in N_i} exp(\sigma(a^l[h_i^l||h_k^l]))}. \tag{4.2}$$

GAT specifies different weights for each layer, where a^l is a learnable vector at the $l-th$ layer. Compared to GAM, a softmax function normalizes the weights that are highly dependent on the degree of each node, which makes it harder to determine their importance. Further, the message passing can degrade the performance since the edge coefficients w_{ij} always maintain a positive value.

FAGCN [2] improves GAT from two perspectives; replacing softmax with degree-based normalization, and adopting different activation function as below:

$$w_{ij}^l = tanh(a^l[h_i^l||h_j^l]). \tag{4.3}$$

The main difference lies in *tanh*, where the negative value of coefficients can maintain high-frequency signals. However, we notice that their accuracy decreases as the homophily of networks increases (e.g., Cora), where all coefficients converged to a positive value and fail to figure out heterophilic edges.

PTDNet [55] removes task-irrelevant edges by applying randomness ϵ and decaying factor γ. Here, the coefficients w_{ij} can be derived as below:

$$w_{ij}^{l} = \sigma((\log \epsilon^{l} - \log(1-\epsilon^{l}) + MLP(h_i^l, h_j^l))/\gamma) \quad (4.4)$$

The random value follows $\epsilon^l \sim$ *Uniform*(0,1), and decaying factor γ depends on the iteration number. They apply nuclear norm on the entire edges w to remove connections between communities. However, we notice that randomness can impede precise prediction, and nuclear norm does not always lead to optimal results.

Summarizing the above methodologies, prediction based on the central node implicates two major problems. Firstly, excluding message passing (GAM) can lead to over-fitting, where it contains limited information. Though other methods incorporate neighboring nodes, the noisy neighbors also participate in the aggregation process, which can impede robustness and incur over-smoothing issues [81]. Secondly, directly employing the coefficients as an adjacency matrix is highly risky, where the elimination of assortative edges hurts the overall performance of GNNs (please refer to Figure 4.1). To solve these limitations, we focus on subgraph matching algorithms which will be introduced in the upcoming section.

4.4.1.2 Retrieving Edge Coefficients using Subgraphs.

In this section, we describe some methods of measuring the similarity between two subgraphs. Recently, applying optimal transport (OT) on subgraphs [69; 87] has shown great improvement, which is a mathematical framework for measuring

distances (similarity) between objects. For example, let us assume two subgraphs G_i,G_j that contain m,n nodes, respectively. Then, we can define transport (coupling) matrix $P \in R^{m \times n}$ between two subgraphs that meets $P1_n = \frac{1}{m}1_m$ and $P^T 1_m = \frac{1}{n}1_n$. The objective of OT is to find matrix P that minimizes the function below:

$$\min_P \sum_{ij} S(\mathcal{G}_i, \mathcal{G}_j) P_{ij} - \epsilon H(P). \tag{4.5}$$

Here, S is a cost function and $H(\cdot)$ is entropy regularized Kantorovich relaxation with regularizer ϵ. However, finding P_{ij} for all pairs of (i, j) requires a high computational cost.

Linear optimal transport [60] employ reference points r to solve the above limitation, which can be retrieved through k-means clustering or calculating Wasserstein barycenter [15] based on each class of training nodes. Here, elements that are assigned to the same cluster (reference) are pooled together and thus, reducing the pair-wise calculation. Specifically, matrix $P \in R^{C \times N}$ splits or assigns the entire node N to references r (C stands for the number of reference points). P can be obtained through multiple ways (e.g., Sinkhorn's algorithm [76]), which calculates a relevance between the inputs and reference points as below:

$$p_i^* = \arg\min_{p \in P_i} \sum_{k=1}^{C} \sum_{l=1}^{N_i} p_{kl} \|r_k - h_l\|^2 \tag{4.6}$$

, where N_i is the number of nodes in subgraph i. Let us assume the hidden representations of two subgraphs as $h_i \in R^{Ni \times F}, h_j \in R^{Nj \times F}$ whose feature dimension is F. Using $p_i^* \in R^{C \times Ni}$, $p_j^* \in R^{C \times Nj}$ in Equation 50 that splits the mass of subgraph

h_i, h_j to multiple references $h'_i, h'_j \in \mathrm{R}^{C \times F}$, one can measure their similarity through matching function M (MLP) as below:

$$h'_i = p^*_i h_i, \;\; h'_j = p^*_j h_j$$
$$w_{ij} = M(h'_i, h'_j). \tag{4.7}$$

Monge map [44] does not split mass, while an injective mapping is applied for each subgraph as follows:

$$h'_i = B(p^*_i, h_i), \;\; h'_j = B(p^*_j, h_j)$$
$$w_{ij} = M(h'_i, h'_j). \tag{4.8}$$

Similar to linear optimal transport [60], each subgraph h_i, h_j can be mapped to new points $h'_i, h'_j \in \mathrm{R}^{C \times F}$ through optimal transport p^* (please refer to Eq. 4.7). The difference lies in a barycentric projection B that ensures no mass splitting (please read this paper [44] for more details).

Using the insight of these methods [60; 44], the subgraph matching has the advantage of using adjacent nodes for predictions. However, they also implicate a limitation of handling noisy neighbors, since they utilize the entire nodes of the subgraph to measure their similarity. To deal with this, we now introduce our method that utilizes a confidence ratio as below.

4.4.1.3 Our subgraph Matching using a Confidence Ratio.

In Figure 4.2, we describe the overall architecture of our Confidence-based Subgraph Matching. ConSM calculates a similarity between two subgraphs (probability of sharing the same label) using optimal transport and confidence ratio as follows:

1. Sampling: We randomly sample two labeled nodes, whose labels can be the same or different. For sampling, the size of the positive and negative pairs should be the

same to avoid a model being biased. We further utilize their 2-hop adjacent nodes as inputs.

2. Prune: We measure a score of entire edges through reference points. Then, based on a confidence ratio, we maintain top-k confident ones while removing others.
3. Map and aggregation: Given two subgraphs G_i,G_j, we first map nodes to low-dimensional embedding h_i,h_j and assign them to the nearest reference points r through the Monge map. Then, we aggregate the nodes that belong to the same reference points by pooling operation (e.g., mean).
4. Prediction: We measure the similarity of the two graphs and also retrieve the class probability of a central node.

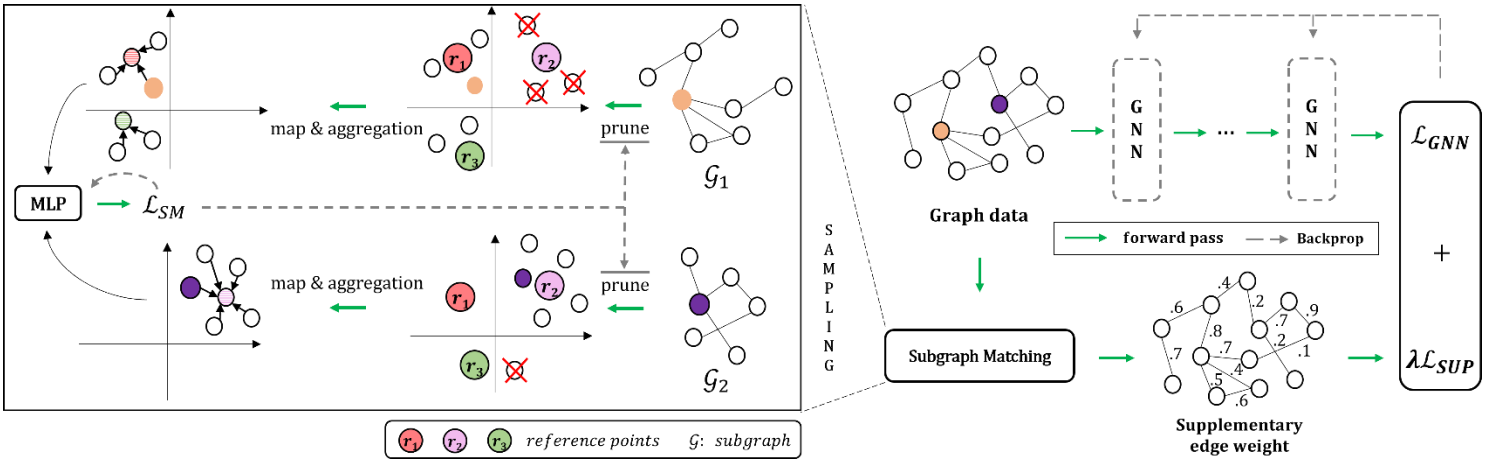


Figure 4.2 The overall framework of our model. It consists of two parts; one for the subgraph matching module which generates supplementary edge coefficients, and the other one is the GNN module that utilizes weights for label propagation.

Now, we describe the details of our method below.

Sampling. We adopt an auxiliary module for retrieving edge coefficients that are independent of the GNN module. Here, two nodes are randomly sampled based on their class. If two nodes share the same class (positive pair), we assume the label of this pair as 1 and otherwise 0. To prevent a class imbalance problem, the same number

of positive and negative pairs are sampled. Compared to GAM [78], we utilize the subgraph of a central node (adjacent nodes within 2-hop) to improve prediction accuracy.

Prune. Unlike previous method [34] that applied *embedding* → *aggregation* → *mapping*, we suggest *embedding* → *pruning* → *mapping* → *aggregation*' to handle noisy edges. Specifically, we only utilize a certain proportion of edges based on their scores and a confidence ratio (ζ). We first describe our scoring function. Using the initial node features X, we can retrieve their low-dimensional embedding $h \in \mathrm{R}^{N\times F}$ through an encoder (MLP) as:

$$h = Encoder(X). \tag{4.9}$$

Similarly, the embedding of reference points is $r \in \mathrm{R}^{C\times F}$, which can be obtained through class-wise averaging of training nodes. Then, we can measure a score (e.g., cosine similarity) $S \in \mathrm{R}^{N\times C}$ between nodes h and references r as below:

$$S = h \cdot r^{T} \tag{4.10}$$

, where the row of S represents a score of each node with respect to the reference points. Given two nodes i, j, we can retrieve their similarity $w_{ij} = S_i \cdot S_j$. Consequently, the w of the entire edges can be obtained, and thus, we manage to maintain top-k edges $k = \lfloor \zeta \times |\mathrm{E}| \rfloor$ while removing others.

Map and aggregation. Using the remaining edges, the adjacency matrix can be reconstructed. With the slight abuse of notation, given two nodes i, j and their subgraphs $\mathrm{G}_i, \mathrm{G}_j$, we assume that their subgraph embedding h_i, h_j can be retrieved as below:

$$h_i = Encoder(\mathcal{G}_i),\ \ h_j = Encoder(\mathcal{G}_j) \tag{4.11}$$

, which is similar to Equation 4.9. $h_i \in \mathrm{R}^{m\times F}, h_j \in \mathrm{R}^{n\times F}$ consists of m and n nodes, respectively. Referring Equation 4.7 and 4.8, we can map each node in subgraph through Monge map as below:

$$h'_i = B(p^*_i, h_i),\ \ h'_j = B(p^*_j, h_j) \tag{4.12}$$

The $h'_i, h'_j \in R^{C\times F}$ is the output of subgraph after mapping and aggregation. Though linear OT is also considerable, we choose the Monge Map which shows the better performance.

Prediction. Finally, using the concatenation of h'_i, h'_j as an input of matching function M, we can estimate their similarity as below:

$$w_{ij} = M(h'_i \oplus h'_j) \tag{4.13}$$

If two inputs share the same label, the value of w_{ij} should be closer to 1, and otherwise 0. We further employ node classification function $f(\cdot)$ (MLP) to predict the label of each subgraph's central node h^e_i *and* h^e_j, where L_{nll} is negative log-likelihood function:

$$\mathcal{L}_{SM} = \sum_{Y_i \neq Y_j} |w_{ij}| + \sum_{Y_i = Y_j} |w_{ij} - 1| + \mathcal{L}_{nll}(f(h^e_i), Y_i) + \mathcal{L}_{nll}(f(h^e_j), Y_j). \tag{4.14}$$

Our subgraph matching module can be trained through Equation 58, and we describe the overall procedure in Algorithm 2.

Algorithm 2 Confidence-based Subgraph Matching

Require: Adjacency matrix A, initialized parameters θ_{SM}, number of entire training epochs K_e, learning ratio η
Ensure: Supplementary edge coefficients w

for number of entire training epochs K_e **do**
  Get embedding of nodes
  Get reference points r by averaging embedding of training nodes per each class
  Find top-k confident edges
  Using the confident edges, reconstruct adjacency matrix within 2-hop as $\tilde{A} = A \vee A^2$
  Sampling positive or negative node pair (i, j)
  for each node pair (i, j) **do**
    Find the subgraphs $\mathcal{G}_i = \tilde{A}[i]$, $\mathcal{G}_j = \tilde{A}[j]$, and obtain their embedding h_i, h_j
    Apply Monge map for each subgraph
    Retrieve the edge coefficient through
    Compute the loss $\mathcal{L}_{SM}$ using
    Update $\theta'_{SM} = \theta_{SM} - \eta \frac{\partial \mathcal{L}_{SM}}{\partial \theta_{SM}}$
  end for
end for
Calculate supplementary edge coefficients w using θ'_{SM}

4.4.2 GNNs with Supplementary Edge Weights.

Recent studies focus on the strategy to better utilize edge weights. For example, some of them directly construct adjacency matrix [81; 55], while others employ label propagation (LP) [6; 78; 82] on GNNs to deal with uncertainty as below:

$$\mathcal{L} = \mathcal{L}_{GNN} + \lambda \mathcal{L}_{LP}. \tag{4.15}$$

L_{GNN} is a widely used loss function for semi-supervised node classification (e.g., GCN [42]) that is defined as follows:

$$\begin{aligned} \widehat{Y} &= softmax(\widehat{A}\sigma(\widehat{A}XW_0)W_1), \\ \mathcal{L}_{GNN} &= \mathcal{L}_{nll}(Y, \widehat{Y}). \end{aligned} \tag{4.16}$$

, where W is a learnable matrix. Though many recently proposed methods [81; 88; 43; 9; 2] are considerable for GNNs, here, we select GCN to show the efficacy of our method. Back into Equation 4.15, λ is a regularizer and L_{LP} gives additional penalties as below:

$$\mathcal{L}_{LP} = \alpha_1 \sum_{i,j \in LL} w_{ij} d(i,j) + \alpha_2 \sum_{i,j \in LU} w_{ij} d(i,j) + \alpha_3 \sum_{i,j \in UU} w_{ij} d(i,j). \quad (4.17)$$

The notation $\{L,U\}$ denotes labeled and unlabeled nodes, where LU means that only a single node is labeled. $w_{ij} \in \{0,1\}$ is a binary value that represents a connection between two nodes i,j, and d is a dissimilarity measuring function (e.g., cosine similarity). Here, $\{a_1,a_2,a_3\}$ acts as a hyper-parameter. Recently, graph agreement model (GAM) contemplates the limitation of fixed w_{ij}, and substitute it as a parameterized model $w_{ij} = g(X_i,X_j)$, where g is a fully-connected networks. However, these methods implicate two limitations. Firstly, they have shown inferior performance for discriminating task-irrelevant edges under semi-supervised learning. Secondly, the estimated w_{ij} scales from zero to one, even making disassortative nodes similar (P-reg [90] also implicates this limitation). Thus, we improve Equation 4.17 as below:

$$\mathcal{L}_{SUP} = \alpha_1 \left(\sum_{i,j \in LU, w_{ij} > k} w_{ij} d(i,j) + \sum_{i,j \in LU, w_{ij} \leq k} (1 - w_{ij})(1 - d(i,j)) \right)$$
$$+ \alpha_2 \left(\sum_{i,j \in UU, w_{ij} > k} w_{ij} d(i,j) + \sum_{i,j \in UU, w_{ij} \leq k} (1 - w_{ij})(1 - d(i,j)) \right). \quad (4.18)$$

By sorting the score of entire edges w, we can retrieve a threshold k based on a given confidence ratio. In Equation 4.18, weights w_{ij} that are greater than k are trained to reduce dissimilarity $d(i,j)$, while others are guided to be dissimilar $1 - d(i,j)$. Referring to Equation 4.15, we replace L_{LP} with our L_{SUP} and define L_G as below:

$$\mathcal{L}_G = \mathcal{L}_{GNN} + \lambda \mathcal{L}_{SUP}. \quad (4.19)$$

As described in GAM, we exclude edges between labeled nodes ($i,j \in LL$) and set α_1 = 1.0, α_2 = 0.5. We set $0.01 \leq \lambda \leq 0.1$ which is proportional to the disassortativity of dataset.

Algorithm 3 Overall Optimization of ConSM

Require: Initialized parameters θ_{SM} and θ_G, number of entire training epochs K_e, best validation score $\beta' = 0$, learning ratio η

Ensure: Trained parameters θ'_{SM}, θ'_G

for number of entire training epochs K_e **do**
  for training samples **do**
    Compute the $\mathcal{L}_{SM}$ using Eq. 58
    Update $\theta'_{SM} = \theta_{SM} - \eta \frac{\partial \mathcal{L}_{SM}}{\partial \theta_{SM}}$
  end for
  Retrieve edge coefficients w using θ'_{SM}
  for training samples **do**
    Compute the $\mathcal{L}_G$ using Eq. 63
    Compute the validation score β
    Update $\theta'_G = \theta_G - \eta \frac{\partial \mathcal{L}_G}{\partial \theta_G}$
    if β ¿ β' **then**
      Save updated parameters θ'_G
      $\beta' = \beta$
  end for
  Load θ'_G if exists
end for

4.4.3 Optimization Strategy.

So far, we define losses of our subgraph matching with label propagation in Equation 4.19. Let us assume the parameters of the subgraph matching module θ_{SM} and the GNN module θ_G without sharing parameters. Here, we notice that our ConSM implicates two limitations for optimization. Firstly, it is hard to determine whether the subgraph matching module θ_{SM} is converged or not. Secondly, the predicted edge coefficients may implicate uncertainty, which can impede the training of GNNs. To solve this, in Algorithm 3, we suggest saving parameters θ'_G only if it attains the best validation score (line 13). Then, before we compute the loss of the next training sample, we can load these parameters if they exist (line 14). Through this mechanism, we can guide GNNs to achieve better performance apart from the uncertainty of supplementary weights.

4.4.4 Computational Complexity Analysis.

The computational costs of our model can be divided into two parts. The first one is a vanilla GCN [42] model whose complexity is known as $O(|E|P_{GCN})$, where they are proportional to the number of entire edges $|E|$ and the size of learnable matrices PGCN. The second term is our ConSM which computes the similarity between two subgraphs. Instead of naively calculating Wasserstein distance $O(n^3\log(n))$, we conduct linear mapping [44] and measure Euclidean distance, which can be retrieved through simple matrix multiplication. Consequently, our computational cost can be defined as $O(|E|P_{GCN} + |E_M|P_{ConSM})$, where $|E_M|$ stands for the number of edges included in sampled subgraph, and P_{ConSM} is the set of parameters in subgraph matching module.

4.5 EXPERIMENTS

In this section, we compare our ConSM with several state-of-the-art methods using a homophilic and heterophilic graph dataset. In particular, we aim to answer the following research questions:

- RQ1: Does ConSM improves node classification accuracy compared to the state-of-the-art approaches?
- RQ2: How much does ConSM accurately specify task-irrelevant edges in terms of graph denoising?
- RQ3: Does the confidence ratio for the subgraph matching module affects the overall classification result?
- RQ4: Can ConSM alleviates over-smoothing for stacking many layers effectively?

4.5.1 Dataset Description and Baselines.

Dataset description. We conduct investigations with the following publicly available dataset. The statistical details are described in Table 3.1, where we categorize them into two types; assortative and disassortative networks. The explanations of each dataset are demonstrated below.

- Assortative networks. For assortative data, we adopt widely used benchmark graphs; Cora, Citeseer, and Pubmed [42]. Here, each node represents a paper and the edge denotes a citation between two papers. Node features stand for the bag-of-words of paper, and each node has a unique label based on its relevant topic.
- Disassortative networks. We adopt Actor co-occurrence graph [80] and Wikipedia network [72] as disassortative graphs. For Actor co-occurrence data, the node stands for an actor, and the edges are co-occurrence on the same Wikipedia pages.

The node label denotes five types based on the keywords of an actor. Similarly, the Wikipedia network consists of Chameleon and Squirrel, where the edges are hyperlinks between web pages. The node features are several informative nouns and we classify them into five categories based on their monthly traffic.

Baselines. Using the above datasets, we compare our method with the state-of-the-art baselines. A brief explanation of these methods can be seen as follows:

- MLP [70] employs a feed-forward neural network that only utilizes a central node for classification.
- GCN [42] is a traditional GNN models that suggests first order approximation of Chebyshev polynomials [16] to localize spectral filters.
- DropEdge [71] randomly removes edges under a given probability to alleviate over-fitting problem.
- GAT [81] specifies different weights between two nodes, while ignoring graph Laplacian matrix.
- GIN [88] pointed out the limited discriminative power of GCN, suggesting a graph isomorphism network that satisfies the injectiveness condition.
- APPNP [43] combines personalized PageRank with GCN that improves prediction accuracy, while reducing computational complexity.
- GCNII [9] integrates identity mapping to redeem the deficiency of APPNP.
- GAM [78] adopts the graph agreement model under the assumption that not all edges correspond to sharing the same label between nodes.
- H_2GCN [99] suggests ego-neighbor separation and hop-based aggregation to deal with heterophilic graph.
- FAGCN [2] further utilizes high-frequency signal beyond low-frequency information in GNNs.

- PTDNet [55] proposes a topological denoising network to prune task-irrelevant edges as a downstream task of GNNs.

4.5.2 Experimental Setup.

All methods are implemented in *PyTorch Geometric*[7], with Adam optimizer (weight decay $5e^{-4}$) and proper learning ratio ($1e^{-3}$). We set the embedding dimension as 64 for all methods, but diversifying it can improve the overall performance [56]. Here, we adopt 2 layers of GNNs for all baselines, while APPNP, GCNII, and GIN further utilize 2 layers of fully-connected networks for classification. We apply ReLU as an activation function except for PTDNet (Sigmoid is used here). The Softmax is applied on the last hidden layers for classification. For all datasets, we randomly select 20 samples per class as a training set, and the rest is for validation and testing. The performance is evaluated based on a test set accuracy that achieved the best validation score.

4.5.3 Results and Discussion (RQ1).

In Table 5 and 6, we describe experimental results of baselines and our method that are conducted under homophilic / heterophilic datasets. Here, let us assume the homophily ratio h as below:

$$h = \frac{\#\text{ of edges that connect nodes with same label}}{\#\text{ of entire edges}} \tag{4.20}$$

Results on homophilic graph datasets In Table 4.3, we first discuss performance on three homophilic datasets, where most of the connected nodes share the same label. We conduct experiments over 10 times and report the mean and variance of test

accuracy. We also describe the performance of MLP to show the influence of message passing on graph datasets. Firstly, for methods that employ GCN as a backbone (marked with †), our approach achieves state-of-the-art performance on multiple benchmark datasets. Specifically, our method outperforms GCN over 3.7 %, 5.2 %, 1.6 %, respectively. Among baselines, in *Citeseer*, DropEdge shows better performance than GCN which has relatively low homophily than other networks. Above all, APPNP and GAM achieve the best performance with the aid of label propagation, followed by GAT adopting an attention mechanism. For our experiments, GCNII shows lower performance than APPNP, which means that emphasizing the identity feature is not suitable for homophilic data. The design choice of rest algorithms (H_2GCN, FAGCN, PTDNet) are for heterophilic graphs, where they fail to achieve notable improvements over GCN.

Datasets	Cora	Citeseer	Pubmed
Hom. ratio (h)	0.81	0.74	0.8
MLP	54.2 ± 0.5 %	53.7 ± 1.7 %	69.7 ± 0.4 %
GCN	80.5 ± 0.4 %	67.5 ± 0.6 %	78.4 ± 0.2 %
DropEdge†	80.6 ± 0.4 %	68.4 ± 0.5 %	78.3 ± 0.3 %
GAT	81.4 ± 0.4 %	69.3 ± 0.9 %	78.6 ± 0.5 %
GIN	78.2 ± 0.2 %	65.1 ± 0.6 %	76.8 ± 0.8 %
APPNP	82.1 ± 0.4 %	69.2 ± 0.7 %	78.7 ± 0.4 %
GCNII	81.1 ± 0.3 %	67.0 ± 0.4 %	78.5 ± 0.8 %
GAM†	81.3 ± 0.6 %	70.4 ± 0.3 %	79.2 ± 0.1 %
H_2GCN	80.5 ± 0.2 %	68.9 ± 0.5 %	78.9 ± 0.2 %
FAGCN	81.5 ± 0.5 %	68.6 ± 0.2 %	79.0 ± 0.1 %
PTDNet†	81.5 ± 0.7 %	69.4 ± 0.6 %	76.9 ± 1.1 %
Ours†	**83.6*** ± 0.3 %	**71.2*** ± 0.2 %	**79.6*** ± 0.1 %

Table 4.3 (RQ1) Node classification accuracy (%) on homophilic citation networks. Bold* symbol indicates the best performance, and methods with † are built upon GCN.

Datasets	Actor	Chameleon	Squirrel
Hom. ratio (h)	0.22	0.23	0.22
MLP	**27.9*** ± 1.1 %	41.2 ± 1.8 %	26.5 ± 0.6 %
GCN	21.4 ± 0.6 %	49.4 ± 0.7 %	33.1 ± 1.2 %
DropEdge†	21.9 ± 0.4 %	49.2 ± 0.8 %	31.0 ± 1.4 %
GAT	23.2 ± 0.8 %	48.2 ± 0.6 %	30.1 ± 0.9 %
GIN	23.6 ± 0.5 %	40.1 ± 3.7 %	23.0 ± 2.4 %
APPNP	21.7 ± 0.2 %	45.2 ± 0.7 %	30.6 ± 0.7 %
GCNII	25.7 ± 0.4 %	45.1 ± 0.5 %	28.8 ± 0.3 %
GAM†	21.7 ± 0.3 %	49.5 ± 0.5 %	33.9 ± 0.4 %
H_2GCN	22.8 ± 0.3 %	46.6 ± 1.2 %	29.8 ± 0.7 %
FAGCN	26.9 ± 0.4 %	46.7 ± 0.6 %	29.5 ± 0.7 %
PTDNet†	21.5 ± 0.5 %	49.7 ± 0.9 %	32.6 ± 0.7 %
Ours†	27.7 ± 0.4 %	**52.3*** ± 0.4 %	**35.7*** ± 0.5 %

Table 4.4 (RQ1) Node classification accuracy (%) on heterophilic citation networks. Bold* symbol indicates the best performance, and methods with † are built upon GCN.

In addition to the homophilic network, we conduct the experiments under heterophilic data with the same settings and plot the results in Table 4.4. As can be seen, these graphs are generally disassortative with a low homophilic ratio h, which can impede the advantages of message passing in GNNs. Surprisingly, MLP achieves the best performance for *Actor*, followed by our ConSM, FAGCN, and GCNII. Given that GCNII outperforms APPNP, we guess that a central node is highly important for the *Actor* network. Nonetheless, these methods fail to outperform GCN for different datasets. Instead, our method achieves the best accuracy on both *Chameleon*, and *Squirrel*. Based on the results that GAM shows outstanding performance for this kind of network, the supplementary model generalizes well under a heterophilic structured dataset. Under our experiments, H_2GCN, FAGCN, and PTDNet have shown to achieve lower scores, which will be discussed in the upcoming section.

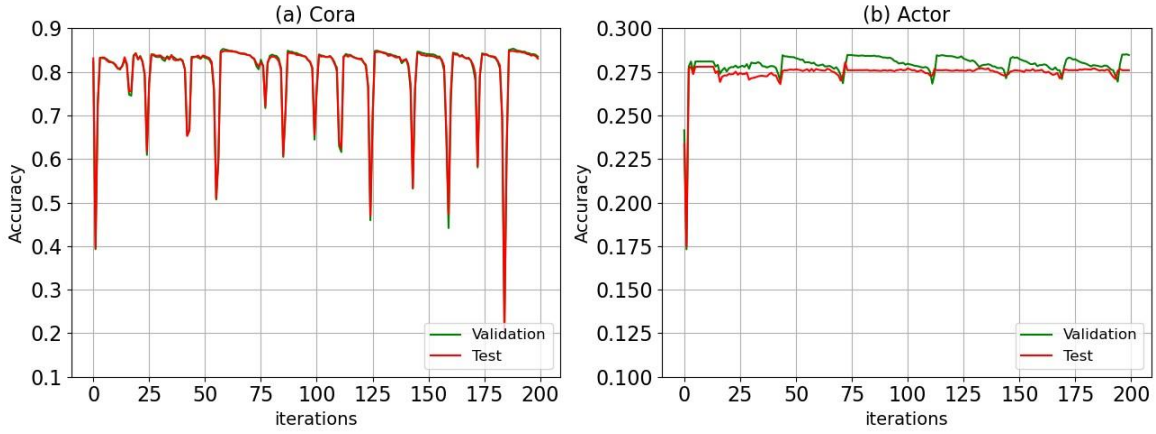


Figure 4.3 (RQ1) Convergence analysis on (a) Cora, and (b) Actor. Each figure contains validation (green) and test (red) accuracy of node classification.

To better understand the convergence of ConSM, in Figure 4.3, we describe validation and test accuracy for training. The x-axis illustrates iterations, while the y-axis is classification accuracy. As described in Algorithm 3, a single iteration is a combination of training subgraph matching modules, followed by training GNN

layers. Here, the validation and test accuracy vary significantly, but ConSM manages to achieve better performance as iteration increases. This is because ConSM loads parameters of the best validation score (please refer to Algorithm 3), which can prevent the uncertainty of supplementary information precisely.

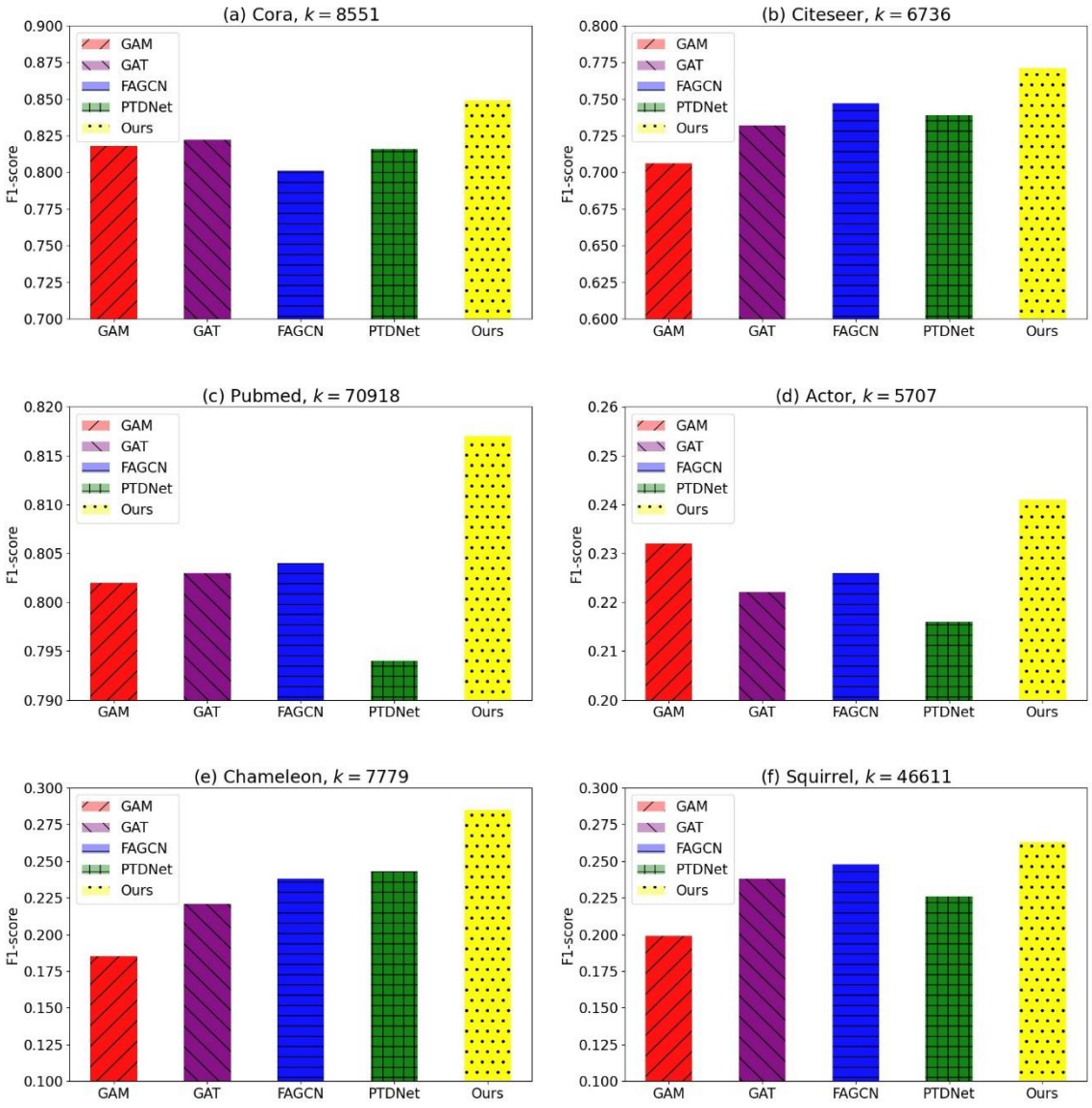


Figure 4.4 (RQ2) We measure F1-score to evaluate edge classification performance on six graph datasets. Here, we adopt our model with four baselines that specify edge coefficients.

4.5.4 Edge Classification (RQ2).

To validate whether ConSM can predict edge coefficients correctly, we examine the accuracy of our method and several state-of-the-art approaches. Here, we assume the label of edges that connect two nodes with the same class as 1 and vice versa.

For each method, we sort their predicted coefficients and select $k-th$ largest value as a threshold, which is equal to the number of positive edges. Specifically, for (a) Cora, $h = 0.81$ and $E = 10{,}558$ (please refer Table 3 and 5), and thus, $k = \lfloor 10{,}558 \times 0.81 \rfloor$ which is described in Figure 4.4. We adopt F1-score that has shown to be effective for binary classification as below:

$$\text{F1-score} = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \tag{4.21}$$

We first introduce some details of baselines, followed by a discussion on the experimental results. (1) GAM: as described in Equation 45, the agreement model generates the same class probability. We employ their edge coefficients with the best validation result. (2) GAT: we exclude node-wise normalization (e.g., softmax), which can be highly sensitive to the degree of central nodes. Then, using the representations of the final hidden layer, we retrieve the attention value of the entire edges. Multihead attention is applied for the front layers, while the final layer only employs single-head attention. (3) FAGCN: they retrieve the coefficients following the Equation 47. Similar to GAT, we exploit the attention values of the last hidden representations. The hyper-parameters are tuned referring [2]. (4) PTDNet: similar to previous studies, we adopt the generated graphs using final representations of GNNs. The hyper-parameters are remaining the same as their *implementations*[8]. (5) Ours: the coefficients of our model can be retrieved through subgraph matching module.

In Figure 4.4, we can see that GAM shows the lowest performance for most graph datasets, except for (d) Actor. It is not surprising since they only utilize a central node

for a prediction. Here, GAT relatively outperforms GAM with the aid of message passing and attention layer. Except for (a) Cora, FAGCN achieves better performance than GAT, which describes the effectiveness of high-frequency signals. Notably, PTDNet is not shown to be powerful enough, where the edge pruning between communities fails to generalize on most graph datasets. Comparatively, our model improves the F1-score significantly for all datasets, which justifies the necessity of confidence-aware subgraph matching.

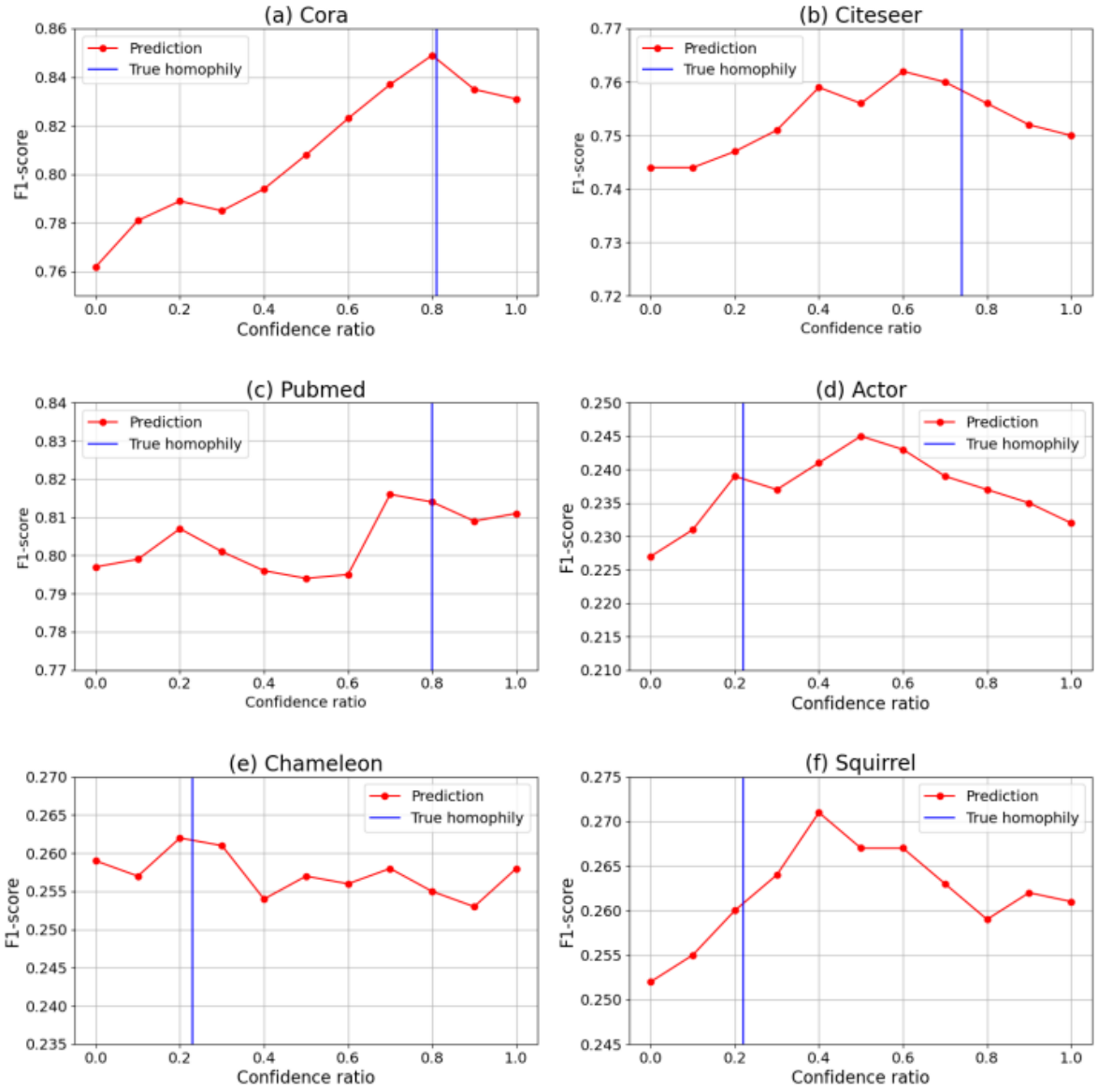


Figure 4.5 (RQ3) We differentiate the confidence ratio of subgraph matching module, and describe F1-score on six graph datasets

4.5.5 Parameter Sensitivity Analysis (RQ3).

In this section, we further measure edge classification scores by differentiating a hyper-parameter of the subgraph matching module. To deal with heterophily, we introduced a confidence ratio (ζ) to reflect data homophily, assuming that connected nodes may not share the same labels. In Figure 4.5, we plot F1-score on six datasets by varying ζ from 0 to 1. We also describe true homophily ratio (please refer h in Table 5 and 6) as blue lines. If $\zeta = 0$, the supplementary module does not utilize neighboring nodes for a prediction, while $\zeta = 1$ means that it fully utilizes adjacent nodes.

Here, a confidence ratio (ζ) that shows the best F1-score fairly aligns well with the true homophily ratio h, and the selection of ζ is important for precise prediction. Though $\zeta = 0.5$ is quite different from $h = 0.22$ for (d) Actor, we insist that disassortative neighbors can also contribute to improving classifications, as we described in Figure 4.1. Nonetheless, we admit that a choice of ζ is quite sensitive, and may require human efforts to achieve the best accuracy.

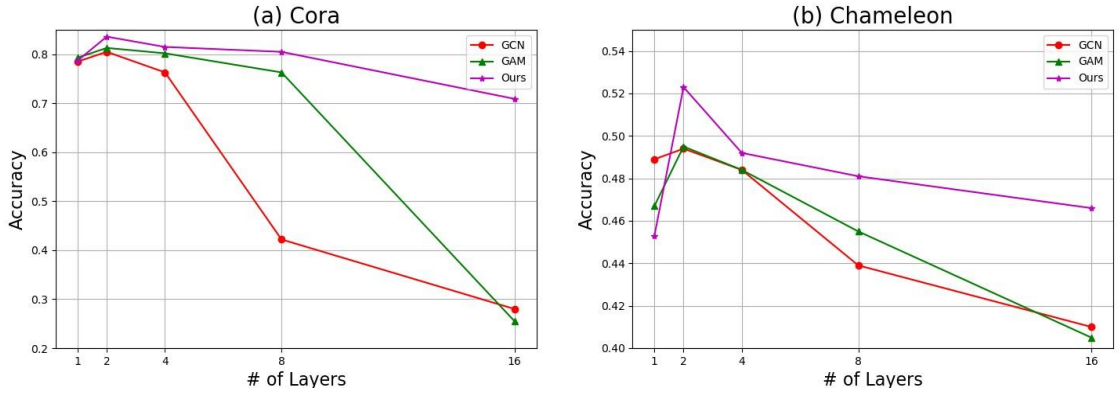


Figure 4.6 (RQ4) Evaluation on over-smoothing using (a) Cora, and (b) Chameleon dataset. We plot the accuracy of two baselines and our method using a different number of layers.

4.5.6 Analysis on Over-smoothing (RQ4).

Over-smoothing is a fundamental problem for GNNs when stacking multiple layers [47; 96]. Here, we scrutinize this phenomenon by differentiating the depth of layers as {1, 2, 4, 8, 16}, and report the node classification accuracy on (a) Cora, and (b) Chameleon. In Figure 4.6, we describe the results of GCN, GAM, and our ConSM. GCN shows the best performance at 2 layers on both datasets. However, they degrade slightly at 4 layers and dramatically decrease beyond it. This means that GCN itself cannot alleviate the over-smoothing problem. Though GAM remains relatively stable compared to GCN, they also suffer from smoothing when stacking more layers. Comparatively, ConSM consistently achieves the best performance, and the accuracy does not decrease severely for deeper layers (e.g., 16 layers). We suggest that the integration of well-classified edge coefficients with label propagation effectively controls this problem, which shows the effectiveness of our method.

4. 6 Conclusion

In this work, we suggest a confidence ratio to deal with multiple disassortative edges for semi-supervised node classification. We pointed out the significance of configuring edge weights precisely, and thus, we propose to measure the similarity between two connected nodes using their subgraphs. Further, based on the observations that directly applying the predicted weights are highly risky, we integrate label propagation with our confidence ratio to secure robustness and improve the overall performance. The extensive experiments for both homophilic and heterophilic setups well describe the superiority of our model.

Chapter 5

Limitation of Real-world Graph Datasets under Semi-supervised Setting

Previous research on Graph Neural Networks (GNNs) in semi-supervised settings has mostly focused on finding suitable graph filters for both homophilic and heterophilic graphs. While these techniques have proven effective, they can still suffer from sparsity in initial node features, where they have only a few non-zero elements for many graph datasets. This can result in overfitting of the first projection matrix (or hyperplane), where the dimensions with zero inputs are not updated during training. To address this issue, we propose a novel data augmentation strategy, which flips the initial features and the hyperplane simultaneously. This creates additional training space and leads to more accurate updates of the learnable parameters, thereby improving robustness during inference while reducing the variance of predictions. To the best of our knowledge, this is the first attempt to mitigate the overfitting problem caused by input features. Our experiments on real-world datasets show that the proposed technique can increase node classification accuracy by up to 40.2 % compared to state-of-the-art baselines.

5.1 Introduction

Graph Neural Networks (GNNs) have gained a lot of attention due to the growing availability of graphical data. By integrating node features with network structures, GNNs have shown powerful abilities for node and graph embedding, resulting in improved performance in downstream tasks [16; 42; 81]. Message-passing, which aggregates features from neighboring nodes through repeated updates, is considered a key component of GNNs [28].

GNNs generally perform well on homophilic graphs [59], where most connected nodes are likely to have the same label. However, the inadequacy of message-passing in heterophilic graphs has been identified in a recent study [67]. To solve this, various solutions have been proposed, such as assigning different weights to edges [81; 92; 2; 40; 14], eliminating disassortative connections [55], embracing distant nodes with high similarity as neighbors [94; 37], or adopting node-specific propagation with trainable boundaries [85]. The proper aggregation scheme and extension of virtual neighbors are clearly important for GNNs. However, we raise another question: are there other factors beyond aggregation schemes?

Contrary to previous methods, our focus is on the training of weight matrices (hyperplanes). We have observed that when the initial features have few non-zero elements (e.g., bag-of-words representation), a shortage of training samples in semi-supervised settings can result in the overfitting of specific dimensions in the first layer parameters. This can negatively impact the quality of predictions for test nodes with untrained features in those dimensions.

To optimize the first layer projection matrix better, we focused on perturbing the initial features. As a common data augmentation technique, dimensional shifting could be used which is commonly used in computer vision [75]. However, this was

found to be unsuitable for GNNs with bag-of-words features, as it would disrupt the semantic information. Unlike convolutional neural networks, which promote local invariance [95], GNNs use a multi-layer perceptron that is not translation invariant. Adding noise to the inputs was also considered [100], but it was discovered that this would incur several complex consequences, such as additional decoding requirements, precise hyper-parameter selection, and normalization issues [7].

Our proposed solution involves flipping the initial features and parameters simultaneously, which can ensure local invariance. This approach is inspired by shifting parameters [41] and rotating neural networks [51] that preserve the volume of gradients and initial features. We also utilize a dual-path network [11] that allows paired operations in both the original and flipped spaces [52]. This flipping mechanism can address the issue of zero gradients caused by sparse inputs and enhance the semantic learning of each dimension. It's worth mentioning that the proposed algorithm is applicable to various schemes and can be integrated with different message-passing algorithms.

In this paper, we apply the flipping mechanism to three popular methods; MLP, GCN, and GAT. We observe that they achieve an average gain of 16.5 %, 24.2 %, and 17.8 % compared to the vanilla models, respectively. These results show that flipping improves the overall performance significantly while securing robustness. The contributions of this paper can be summarized as follows:

- We demonstrate that GNNs are highly sensitive to initial feature vectors and their performance can be significantly improved through flip-based augmentation.
- We propose a flipping mechanism that transposes both the initial features and hyperplane. Unlike previous methods that focus on aggregation schemes, our approach examines back-propagation and provides precise guidance for each component of a first hyperplane.

- The proposed flipping mechanism is orthogonal to the plane methods. By applying it to MLP, GCN, and GAT, we develop three flipping variants. Through extensive experiments on real-world benchmark graphs, the flipping variants outperform all existing state-of-the-art baselines significantly.

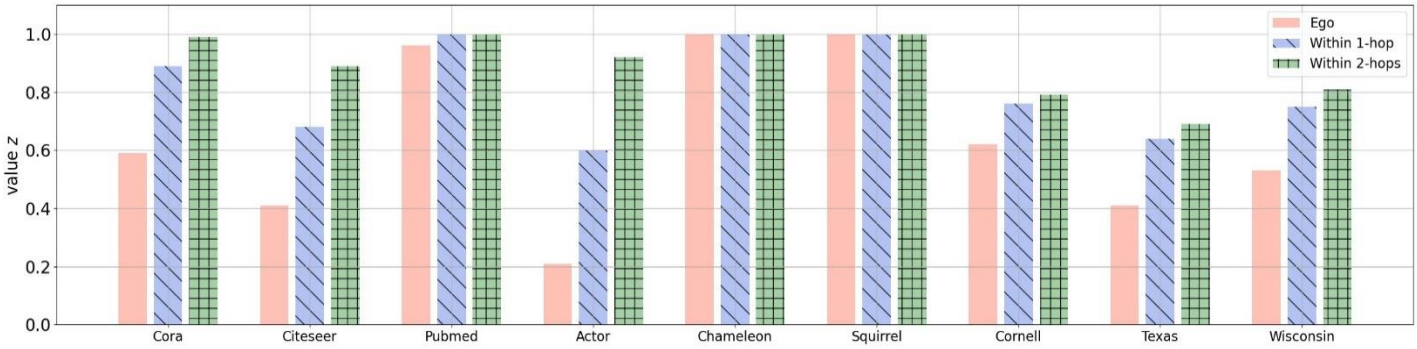


Figure 5.1 Initial feature distribution of benchmark graph datasets. The definition of value z is described in Equation 5.1.

5.2 Preliminary

This section begins with the commonly used notations in Graph Neural Networks (GNNs), which will be utilized throughout this paper. Next, we conduct an empirical analysis to illustrate an overview of the feature distribution in benchmark datasets. Finally, we introduce the mechanism of GNN from the perspectives of feature projection and message-passing.

5.2.1 Notations.

Here, we separate the weight matrix for the first layer of the GNN into two parts, W_o and W_f, where the subscripts o and f denote the original and flipped spaces, respectively. Additionally, for gradient analysis, we take the symbol $\bigtriangledown$ to represent the partial derivative of the loss function. The goal of this work is to solve a node classification task in a semi-supervised setting where only a subset of nodes $\mathrm{V}_L \subset \mathrm{V}$ is labeled. Our goal is how to better utilize the given features to predict the classes of unlabeled nodes $\mathrm{V}_U = \mathrm{V} - \mathrm{V}_L$.

5.2.2 Empirical Analysis.

Given the node set S and their initial features $X \in R^F$, the ratio of non-zero feature dimension (z) in S can be defined as below:

$$z = \frac{\sum_{o=1}^{|j|}(1 - \delta_{j_o,0})}{dim(x)}, \quad j = \sum_{v \in S} X_v. \tag{5.1}$$

Firstly, we obtain $j \in R^F$ by adding the feature vectors of subset node X_v. The number of non-zero elements in vector j can be defined through the Kronecker delta function δ, where $\delta_{j_o,0} = 1$ if the o^{th} element in j is 0. Finally, we can retrieve z by dividing the numerator into the feature vector dimension $dim(x) = F$.

In Figure 2, we display the z by varying the range of node set S from ego to their 2-hop neighboring nodes. As seen, z increases with the range due to the availability of more features during training. Additionally, the scale of z varies significantly for each graph, dependent on the type of input (please refer to the dataset description in § 4.1). In essence, the lower the value of z, the greater the performance improvement obtained from flipping. To further examine this phenomenon, we provide a theoretical explanation in terms of gradient update and variance reduction.

5.2.3 Graph Neural Network.

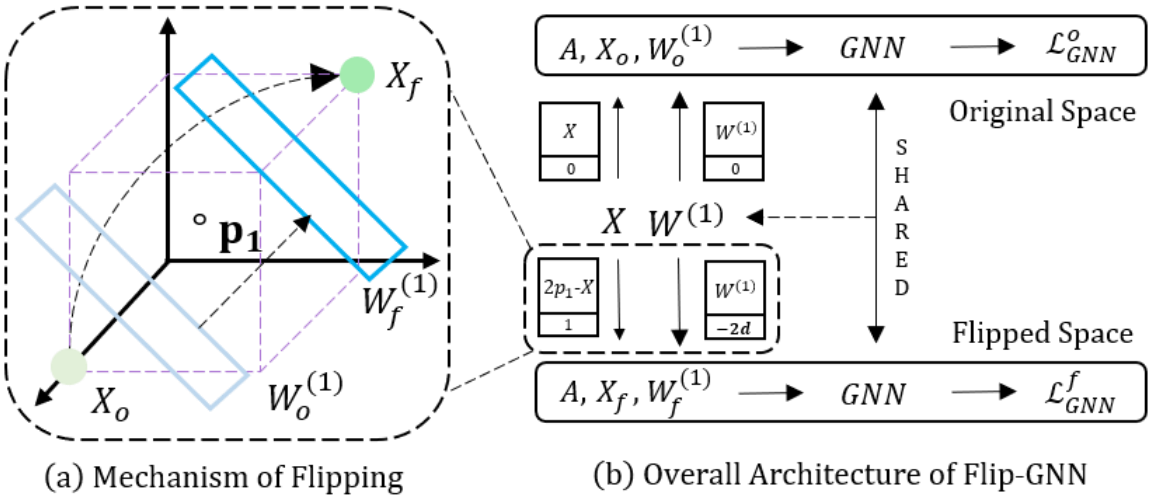
The basic form graph neural network is given by:

$$\begin{gathered} H^{(l+1)} = \sigma(\bar{H}^{(l+1)}), \quad \bar{H}^{(l+1)} = AH^{(l)}W^{(l)} \quad (l \geq 1) \\ \widehat{Y} = softmax(\bar{H}^{(L)}). \end{gathered} \tag{5.2}$$

Previously, we defined A as an adjacency matrix that is used for message-passing. With the slight abuse of notation, let us assume $A = I + D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$ for the remaining part of this paper, which is commonly used in GCN [42]. $H^{(1)} = X$ is an initial feature of nodes and $\bar{H}^{(l)}$ is their hidden representation at the *l-th* layer. $H^{(l)}$ can be retrieved through an activation function σ (e.g., ReLU). GNNs obtain the final prediction Y_b, by applying softmax on the final representation ($\bar{H}^{(L)}$). Here, $W^{(l)}$ is the trainable weight matrices shared across all nodes. They are updated through negative log-likelihood function (L_{nll}) between the predicted Y_b and true label Y as below:

$$\mathcal{L}_{GNN} = \mathcal{L}_{nll}(\widehat{Y}, Y) \tag{5.3}$$

Generally, GNNs focus on improving aggregation schemes to determine an appropriate message-passing [81; 43; 9; 2]. For instance, GCN [42] uses a normalized Laplacian matrix, while GAT [81] creates an aggregation matrix by calculating the attention score between nodes. However, the exploration of input features has not been given as much attention in prior studies, where we highlight the need for further investigation below.



(a) Mechanism of Flipping (b) Overall Architecture of Flip-GNN

Figure 5.2 (a) Mechanism of flipping and (b) overall architecture of Flip-GNN.

5.3 Methodology

5.3.1 Motivation.

We first explain the limitation of generic GNNs. While appropriate aggregation schemes are undoubtedly essential for efficient message-passing, as explained below, this alone cannot solve the improper learning caused by sparsity in the initial features. To be more specific, we can define the update of weight matrix $W^{(l)}$ as below:

$$\nabla_{W^{(l)}} J = (AH^{(l)})^T \nabla_{\bar{H}^{(l+1)}} J, \quad l = 1, \ldots, L. \tag{5.4}$$

The $J = L_{GNN}$ is a full-batch loss defined in Equation 5.4. Intuitively, zero or small valued components in A can obstruct the gradient flow between dissimilar nodes. Nonetheless, there arises a problem when updating the parameters of initial layer:

$$\begin{aligned} \nabla_{W^{(1)}} J &= (AH^{(1)})^T \nabla_{\bar{H}^{(2)}} J \\ &= (AX)^T \nabla_{\bar{H}^{(2)}} J. \end{aligned} \tag{5.5}$$

Simply, the gradient of $W^{(1)}$ is derived by differentiating J with respect to $\bar{H}^{(2)}$, where the value of AX determines the scale of a gradient. Thus, the gradients become zero for certain dimensions with zero inputs ($\forall i \in F : X_i = 0 \Rightarrow \nabla_{\bar{H}^{(2)}} J = 0$).

Now, we can see that the update of $W^{(1)}$ relies on the sparseness of the input features, especially for zero elements. Because a deficiency in training samples is common in semi-supervised settings, we focus on removing zeros in X and guide $W^{(1)}$ to learn the precise meaning of each dimension.

Augmentation of input features. One may consider that shifting, an accepted technique in computer vision, is a simple remedy to the inadequate gradient update problem. To implement shifting, a small valued vector X_s is added to the input features as $X = X + X_s$. Although shifting has been shown to improve the quality of the initial features, it may not be applicable to GNNs, as multi-layer perceptron is not shift-invariant, which can lead to decreased robustness. Additionally, shifting changes the magnitude of an input, necessitating complex neural network normalization. Another alternative could be magnitude-conserving rotation, but it may force some components to take negative values.

5.3.2 Flipped Graph Neural Network.

We present a scheme that simultaneously flips both the feature vectors and the hyperplane. If the original feature vector has elements in the range [0, 1], then its symmetric transposition through $p_1 = (0.5,...,0.5)$ will also lie in the same range (as seen

in the hypercube in Figure 5.2 (a), which illustrates that a feature vector $X_o = (1,0,0)$ is transposed to $X_f = (0,1,1)$. This is also applied to the hyperplane W_o, where Figure 5.3 shows the original and flipped spaces in the upper and lower panels, respectively.

In the proposed method, both spaces share the same parameters, while the initial features and the first hyperplane are slightly tuned for each iteration. Here, we assume GCN as a base model.

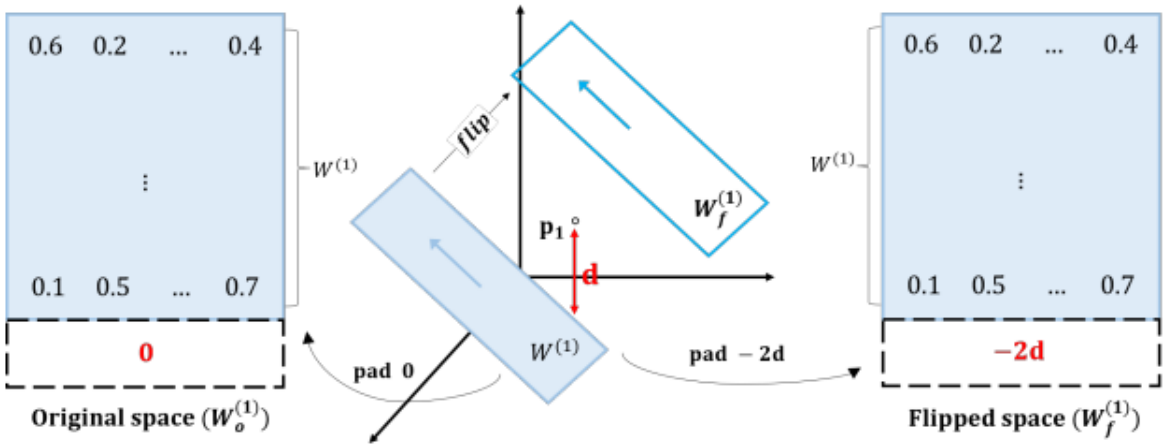


Figure 5.3 (a) Distance d from $W^{(1)}$ to p_1. (b) $W_f^{(1)}$ is retrieved by padding -2d to the last dimension of $W^{(1)}$.

Original space. As described in Figure 5.2 (b), the upper panel illustrates the plane GCN. It takes X_o and $W_o^{(1)}$ as inputs which are the zero-padded version of the initial feature matrix X and the first hyperplane $W^{(1)}$ as below:

$$X_o = \begin{pmatrix} X \\ 0 \end{pmatrix}, \quad W_o^{(1)} = \begin{pmatrix} W^{(1)} \\ 0 \end{pmatrix} \tag{5.6}$$

Though the last dimension is only utilized in the flipped space, zero-padding is required to ensure dimensional consistency as $W^{(1)}$ is utilized in both spaces. Now, we can compute the loss $J_o = \mathrm{L}_{GNN}(Y, Y_o)$ using Eq. 67 and 68, and update the parameters W. Before introducing the flipped space, we first define a symbol $p_1 = (0.5,...,0.5) \in \mathrm{R}^F$, which serves as an anchor point for flipping. While many points can be used as an anchor (e.g., the mean of all nodes), we take the central point of F-dimensional hypercube $(1,...,1) \in \mathrm{R}^F$ as the anchor. Many graph datasets adopt bag-of-words features, and their feature vectors correspond to the corners of the hypercube.

Flipped space. The flipped feature X_f transposes X through p_1 (Fig. 5.3 a) and pads 1 as below:

$$X_f = \begin{pmatrix} 2p_1 - X \\ 1 \end{pmatrix}. \tag{5.7}$$

We should also flip the first hyperplane $W^{(1)}$ by calculating a distance vector $d \in \mathrm{R}^F$ between $W^{(1)}$ and p_1 as:

$$W_f^{(1)} = \begin{pmatrix} W^{(1)} \\ -2d \end{pmatrix}, \ \textit{where } d = \sum_{j=0}^{F-1} (p_1 \otimes W^{(1)})[:j], \tag{5.8}$$

where $\otimes$ is an element-wise product. As shown in Figure 5.3 (b), $W_f^{(1)}$ is retrieved by padding $-2d$ to the last element, which makes the outputs of the two spaces origin-symmetric to each other, i.e., $X_o W_o^{(1)} = -X_f W_f^{(1)}$. This is why we flip the hyperplane concurrently, as it preserves the pairwise distance of the hidden node representations. Thus, after the first convolution layer, we should multiply $\sigma(AX_f W_f^{(1)})$ by a negative constant before applying the next convolution layer to ensure consistency between the two spaces as,

$$H_f^{(2)} = -\sigma(AX_f W_f^{(1)}) \tag{5.9}$$

Through Eq. 5.9, the equality holds ($\forall_l : l \geq 2 \rightarrow H_f^{(l)} = H_o^{(l)}$). Thus, the following layer in the flipped space is identical to the one in the original space as below:

$$H_f^{(l+1)} = \sigma(\bar{H}_f^{(l+1)}), \ \ \bar{H}_f^{(l+1)} = AH_f^{(l)} W^{(l)} \ \ (l \geq 2). \tag{5.10}$$

Finally, the loss L_{GNN}^f is given by:

$$\mathcal{L}_{GNN}^f = \mathcal{L}_{nll}(Y, \hat{Y}_f), \ \ \textit{where } \hat{Y}_f = softmax(\bar{H}_f^{(L)}). \tag{5.11}$$

We describe the overall mechanism of our method in Alg. 4.

Algorithm 4 The overall mechanism of Flip-GCN

Require: Adjacency matrix A, node features X, parameters θ_G, epoch K, best valid and test acc. $\gamma' = \delta' = 0$, learning rate=η

Ensure: Best test accuracy δ'

for number of training epoch $2K$ **do**
    # Original space
    for training samples **do**
        Given X_o and $W_o^{(1)}$, retrieve the $\mathcal{L}_{GNN}^{o}$ (Eq. 68)
        Update $\theta_G^{k+1} = \theta_G^k - \eta \frac{\partial \mathcal{L}_{GNN}^{o}}{\partial \theta_G^k}$
    end for
    Compute the validation γ and test score δ
    if $\gamma > \gamma'$ **then**
        $\gamma' = \gamma, \delta' = \delta$
    # Flipped space
    for training samples **do**
        Get flipped features X_f (Eq. 72)
        Get flipped hyperplane $W_f^{(1)}$ (Eq. 73)
        Compute $\mathcal{L}_{GNN}^{f}$ (Eq. 76)
        Update $\theta_G^{k+2} = \theta_G^{k+1} - \eta \frac{\partial \mathcal{L}_{GNN}^{f}}{\partial \theta_G^{k+1}}$
    end for
    Compute the validation γ and test score δ
    if $\gamma > \gamma'$ **then**
        $\gamma' = \gamma, \delta' = \delta$
end for

5.3.3 Optimization.

We define two loss functions in Eq. 5.3 and 5.11. Before gradient analyses, please recall that the equation below holds

$$\nabla_{W_f^{(1)}} J_f = (AX_f)^T \nabla_{\bar{H}_f^{(2)}} J_f, \;\; \bar{H}_f^{(2)} = AX_f W_f^{(1)}. \quad (5.12)$$

The above equation implies that the outputs (or gradients) of the two spaces are equivalent after the second layers:

$$\nabla_{W_o^{(l)}} J_o = \nabla_{W_f^{(l)}} J_f \;\; (l \geq 2). \quad (5.13)$$

Like J_o, the $J_f = L_{GNN}^f$ is a full-batch gradient in the flipped space. Though Sigmoid or Tanh guarantees a perfect symmetry $J_o = J_f$, we employ ReLU for better performance. Now, referring to Eq. 5.13, we define the gradients of the first hyperplane $W^{(1)}$ on both spaces.

In the original space, update W_o as,

$$\nabla_{W_o^{(1)}} J_o = (AX_o)^T \nabla_{\bar{H}_o^{(2)}} J_o, \;\; \bar{H}_o^{(2)} = AX_o W_o^{(1)}. \quad (5.14)$$

In the flipped space, update W_f as,

$$H_o^{(2)} = \sigma(AX_o W_o^{(1)}) = H_f^{(2)} = -\sigma(AX_f W_f^{(1)}), \quad (5.15)$$

Proof of convergence. Convergence is one crucial aspect of algorithm design. Here, we show that our optimization guarantees the convergence of $W^{(1)}$. If the activation function ensures origin symmetry, we can redefine Eq. 5.14 and 5.15 as:

$$\begin{aligned} \nabla_{W_o^{(1)}} J_o &= (AX)^T \nabla_{\bar{H}_o^{(2)}} J_o, \\ \nabla_{W_f^{(1)}} J_f &= -(A(2p_1 - X))^T \nabla_{\bar{H}_f^{(2)}} J_f. \end{aligned} \quad (5.16)$$

Here, the component-wise gradient of $W^{(1)}$ is proportional to that of X and $2p_1 - X$. Also, it gets closer to a local optimum $W_*^{(1)}$ as the iteration T continues:

$$\mathbb{E}[W_T^{(1)} - W_*^{(1)}]^2 \propto \frac{\log T}{T} \tag{5.17}$$

since the two-layer neural networks with a ReLU activation converge to a local minimum. Note that gradient ∇J and parameters $|W_o^{(l)}|, |W_f^{(l)}|$ are all bounded. These properties guarantee the convergence of Flip-GNN [8]. Since the scale of gradients depends on the number of activated dimensions, we adjust them using α,β to stabilize our model as below:

$$\begin{aligned} W_o &= W_o - \alpha \nabla_{W_o} J_o, \\ W_f &= W_f - \beta \nabla_{W_f} J_f. \end{aligned} \tag{5.18}$$

5.4 Theoretical Analysis

Data augmentation is closely related to empirical risk minimization, which can be explained through the *bias-variance tradeoff* [10]. Here, we prove that flipping acts as an augmentation strategy by generalizing the trainable parameters and reducing the variance of predictions. Firstly, let us assume the plane estimator as $g(X_o) = GNN(X_o)$, which is trained only with the original feature X_o, and the augmented network as $\bar{g}(X) = GNN(X)$ that uses both features $X = X_o \cup X_f$. We can easily see that the function g is invariant to flipping since $g(X_o, W_o) = g(X_f, W_f)$, where X_f preserves the pair-wise distance between nodes. Consequently, the bias term vanishes, where we can decompose $g(X)$ by the law of total variance as below:

$$\begin{aligned} Var(g(X)) &= Var(\mathbb{E}[g(X)]) + \mathbb{E}[Var(g(X))] \\ &= Var(\bar{g}(X)) + \mathbb{E}[Var(g(X))]. \end{aligned} \tag{5.19}$$

Here, $Var(\mathrm{E}[g(X)]) = Var(\bar{g}(X))$ since they share the same marginal distribution. Further, the difference of their mean, $\mathrm{W}_1(\mathrm{E}[g(X)], \mathrm{E}[\bar{g}(X)])$, which equals to the

Wasserstein distance (e.g., L_2) between two distributions is independent of the total variance. Based on this observation, we can induce the condition below:

$$Var(\bar{g}(X)) \leq Var(g(X)). \tag{5.20}$$

Finally, we show the losses of two networks follow:

$$\mathcal{L}(\bar{g}(X)) - \mathcal{L}(g(X)) \in -\mathbb{E}\left[tr(Var(g(X))\right], \tag{5.21}$$

which means the performance gain of the augmented model over the plain method depends on the variance reduction. One can induce a tighter bound of Eq. 5.19 and 5.21 using *Loewner order* [10], but we omit the detailed derivation for brevity.

5.5 Experiments

This section describes the experiments for the performance analysis. We focused our efforts to find answers to the following research questions:

- RQ1: Does flipping effectively address the issue of multiple zero-valued components in the features?
- RQ2: Does flipping ensure convergence?
- RQ3: How significant is the difference between the gradients from the original and the flipped spaces?
- RQ4: How does the performance of flipping change as the number of training samples increases?

Datasets	Cora	Citeseer	Pubmed	Actor	Chameleon	Squirrel	Cornell	Texas	Wisconsin
# Nodes	2,708	3,327	19,717	7,600	2,277	5,201	183	183	251
# Edges	10,558	9,104	88,648	25,944	33,824	211,872	295	309	499
# Features	1,433	3,703	500	931	2,325	2,089	1,703	1,703	1,703
# Classes	7	6	3	5	5	5	5	5	5
# Training Nodes	140	120	60	100	100	100	25	25	25
# Validation Nodes	1,568	2,207	18,657	3,750	1,088	2,550	79	79	113
# Test Nodes	1,000	1,000	1,000	3,750	1,089	2,551	79	79	113

Table 5.1 Statistical details of nine benchmark datasets.

5.5.1 Datasets and Baselines.

Details of datasets. Our experiments are conducted on nine datasets whose statistical details are described in Table 7. We also measure the assortativity of each dataset as below:

$$h = \frac{\sum_{(i,j)\in\mathcal{E}} \mathbb{1}(Y_i = Y_j)}{|\mathcal{E}|} \tag{5.23}$$

- Cora, Citeseer, Pubmed [42] are citation networks. The node features in Cora and Citeseer are binary bag-of-words while Pubmed consists of TF-IDF values.
- Actor [80] is an actor co-occurrence graph. The node feature encodes the keywords in the actor's Wikipedia web pages with binary values.
- Chameleon, Squirrel [72] are taken from Wikipedia web pages and have non-zero positive or negative values. The maximum values in each dataset are 46.4 and 70.4 while the minimum values are -0.57 and -0.99, respectively, which might not be suitable for our method.
- Cornell, Texas, Wisconsin contain web pages from cs departments of multiple universities. The node features are binary bag-of-words like the citation networks.

Baselines. For evaluation, we employ several traditional methods including MLP [70], GCN [42], DropEdge [71], and GIN [88]. Further, we compare GAT [81], GATv2 [4], APPNP [43], GCNII [9], H_2GCN [99], and FAGCN [2] which are designed for heterophilous graphs. Finally, some regularization-based algorithms like P-reg [90] and Ortho-GCN [30] are compared here.

Datasets	Cora	Citeseer	Pubmed	Actor	Chameleon	Squirrel	Cornell	Texas	Wisconsin
z (Eq. 1)	0.59	0.41	0.96	0.21	1.0	1.0	0.62	0.41	0.53
Homophily (Eq. 21)	0.81	0.74	0.8	0.22	0.23	0.22	0.11	0.06	0.16
MLP	53.2 $\pm 0.5\%$	53.7 $\pm 1.7\%$	69.7 $\pm 0.4\%$	27.9 $\pm 1.1\%$	41.2 $\pm 1.8\%$	26.5 $\pm 0.6\%$	60.1 $\pm 1.2\%$	65.8 $\pm 5.0\%$	73.5 $\pm 5.4\%$
GCN†	79.1 $\pm 0.7\%$	67.5 $\pm 0.3\%$	77.8 $\pm 0.2\%$	20.4 $\pm 0.6\%$	49.4 $\pm 0.7\%$	31.8 $\pm 0.9\%$	39.4 $\pm 4.3\%$	47.6 $\pm 0.7\%$	40.5 $\pm 1.9\%$
DropEdge†	79.0 $\pm 0.5\%$	67.4 $\pm 0.2\%$	77.0 $\pm 0.3\%$	20.2 $\pm 0.4\%$	48.9 $\pm 1.1\%$	30.9 $\pm 0.8\%$	46.7 $\pm 3.5\%$	47.5 $\pm 2.6\%$	43.3 $\pm 4.6\%$
Ortho-GCN†	80.6 $\pm 0.4\%$	69.5 $\pm 0.3\%$	76.9 $\pm 0.3\%$	21.4 $\pm 1.6\%$	46.7 $\pm 0.5\%$	31.3 $\pm 0.6\%$	45.4 $\pm 4.7\%$	53.1 $\pm 3.9\%$	46.6 $\pm 5.8\%$
GIN	77.3 $\pm 0.8\%$	66.1 $\pm 0.6\%$	77.1 $\pm 0.7\%$	24.6 $\pm 0.8\%$	49.1 $\pm 0.7\%$	28.4 $\pm 2.2\%$	42.9 $\pm 4.6\%$	53.5 $\pm 3.0\%$	38.7 $\pm 0.5\%$
GAT	80.1 $\pm 0.6\%$	68.0 $\pm 0.7\%$	78.0 $\pm 0.4\%$	22.5 $\pm 0.3\%$	46.9 $\pm 0.8\%$	30.8 $\pm 0.9\%$	42.1 $\pm 3.1\%$	49.2 $\pm 4.4\%$	45.8 $\pm 5.3\%$
GATv2	79.5 $\pm 0.5\%$	67.4 $\pm 0.6\%$	76.2 $\pm 0.5\%$	22.1 $\pm 2.0\%$	48.3 $\pm 0.4\%$	28.9 $\pm 1.2\%$	38.0 $\pm 3.8\%$	52.5 $\pm 1.7\%$	41.7 $\pm 5.1\%$
APPNP	81.2 $\pm 0.4\%$	68.9 $\pm 0.3\%$	79.0 $\pm 0.4\%$	21.5 $\pm 0.2\%$	45.0 $\pm 0.5\%$	30.3 $\pm 0.6\%$	49.8 $\pm 3.6\%$	56.1 $\pm 0.2\%$	45.7 $\pm 1.7\%$
GCNII	80.8 $\pm 0.7\%$	69.0 $\pm 1.4\%$	78.8 $\pm 0.4\%$	26.1 $\pm 1.2\%$	45.1 $\pm 0.5\%$	28.1 $\pm 0.7\%$	62.5 $\pm 0.5\%$	69.3 $\pm 2.1\%$	63.2 $\pm 3.0\%$
H_2GCN	79.5 $\pm 0.6\%$	67.4 $\pm 0.5\%$	78.7 $\pm 0.3\%$	25.8 $\pm 1.2\%$	47.3 $\pm 0.9\%$	31.1 $\pm 0.5\%$	59.8 $\pm 3.7\%$	66.3 $\pm 4.6\%$	61.5 $\pm 4.4\%$
P-reg†	80.0 $\pm 0.8\%$	69.2 $\pm 0.7\%$	77.4 $\pm 0.4\%$	20.9 $\pm 0.5\%$	49.1 $\pm 0.1\%$	**33.6*** $\pm 0.4\%$	44.9 $\pm 3.1\%$	58.5 $\pm 4.2\%$	53.7 $\pm 2.6\%$
FAGCN	81.0 $\pm 0.3\%$	68.3 $\pm 0.6\%$	78.9 $\pm 0.4\%$	26.7 $\pm 0.8\%$	46.8 $\pm 0.6\%$	29.9 $\pm 0.5\%$	46.5 $\pm 1.7\%$	53.8 $\pm 1.2\%$	51.0 $\pm 4.1\%$
Flip-MLP	61.4 $\pm 0.7\%$	60.3 $\pm 0.5\%$	74.1 $\pm 0.5\%$	**35.9*** $\pm 0.5\%$	43.5 $\pm 1.2\%$	28.5 $\pm 0.8\%$	**70.5*** $\pm 4.1\%$	**79.2*** $\pm 3.6\%$	**80.5*** $\pm 5.1\%$
vs MLP (+ %)	+ 15.4 %	+ 12.1 %	+ 6.3 %	**+ 28.7** %	+ 2.3 %	+ 2.1 %	**+ 17.3** %	**+ 20.4** %	**+ 9.5** %
α, β	1, 0.1	1, 1	1, 0.01	0.1, 0.1	1, 1	1, 1	0.1, 0.1	0.1, 0.01	1, 1
Flip-GCN†	82.7 $\pm 0.5\%$	72.4 $\pm 0.4\%$	**79.2*** $\pm 0.2\%$	28.6 $\pm 0.3\%$	**50.4*** $\pm 0.5\%$	32.4 $\pm 0.3\%$	49.4 $\pm 0.6\%$	62.2 $\pm 1.8\%$	52.3 $\pm 2.4\%$
vs GCN (+ %)	+ 4.6 %	+ 7.3 %	**+ 1.8** %	+ 40.2 %	**+ 2.0** %	+ 1.9 %	+ 20.4 %	+ 30.7 %	+ 29.1 %
α, β	0.1, 0.01	0.01, 0.001	1, 0.01	$1e^{-4}$, $1e^{-4}$	1, $1e^{-4}$	1, 0.01	1, $1e^{-4}$	1, $1e^{-4}$	0.01, 0.001
Flip-GAT	**83.1*** $\pm 0.6\%$	**72.8*** $\pm 0.4\%$	78.5 $\pm 0.2\%$	30.3 $\pm 0.8\%$	48.3 $\pm 0.3\%$	31.9 $\pm 0.5\%$	51.9 $\pm 1.4\%$	61.0 $\pm 1.1\%$	54.5 $\pm 2.1\%$
vs GAT (+ %)	**+ 3.7** %	**+ 7.1** %	+ 0.6 %	+ 34.7 %	+ 3.0%	+ 3.6 %	+ 20.2 %	+ 22.0 %	+ 19.0 %
α, β	1, 0.1	0.01, 0.001	1, 0.001	0.1, 0.1	1, 0.1	1, 1	0.1, $1e^{-4}$	0.1, $1e^{-4}$	0.1, 0.001

Table 5.2 (RQ1) Node classification accuracy (%) on nine benchmark datasets. Bold with an asterisk (*) symbol indicates the best performance, and methods with † are built upon GCN. We show α, β that achieves the best accuracy (Eq. 5.18).

5.5.2 Results and Discussion (RQ1).

Flipping can be integrated into various neural networks. We apply it to three representative models; MLP, GCN, and GAT. In Table 5.2, we observe that all flipping variants (Flip-MLP, Flip-GCN, and Flip-GAT) perform significantly better than their base models. Now, we analyze these results from two perspectives.

Performance gain of flipping is sensitive to the Z-value of each dataset. Since flipping is designed to reduce overfitting caused by the sparsity in initial features, we can presume that the non-zero element ratio (z-value) is the key factor that determines the performance gains of flipping. Indeed, flipping attains larger performance gains on low z-value datasets than on higher ones. For the three datasets with higher z-values (Pubmed, Chameleon, and Squirrel), the advancement of flipping over their vanilla models (e.g., Flip-MLP vs MLP) is relatively small. Nonetheless, the average gain of flipping was 3 %, 1.9 %, and 2.4 %, respectively, indicating the effectiveness of flipping even on datasets with large z-values.

On datasets with low z-values, three flipping variants obtain remarkable advancements over their originals, achieving performance gains of 16.5 %, 24.2 %, and 17.8 % on average. Notably, flipping methods perform best except for Squirrel

(with high z-value and low homophily). This may imply that a slight perturbation to the input features can have a greater impact than aggregation scheme modifications under semi-supervised settings.

Relatedness between the homophily ratio and performance. Message-passing GNNs utilize the homophily property commonly observed in graphs [50; 89]. In three citation graphs, GNNs outperform Multi-Layer Perceptron (MLP) due to higher homophily ratios in these graphs. However, in other datasets like Actor and three WebKB networks, Flip-MLP achieves the best accuracy among the baselines indicating that message-passing fails to generalize well in the presence of high heterophily. The performance gain of Flip-MLP is higher than Flip-GNNs on homophilic graphs, but GNNs benefit more from flipping on heterophilic graphs. Although several baselines achieve notable performance, our flipping methods outperform all of these algorithms on the overall datasets, demonstrating the effectiveness across various GNN architectures.

5.5.3 Convergence Analysis (RQ2).

One may argue that flipping could negatively impact the stability of the algorithms due to the operations in two spaces. Figure 5.4 illustrates the performance of vanilla GCN and GAT compared to Flip-GCN in both spaces as a function of the number of iterations. We show the results from four datasets only due to the limited space. The performance of GCN (blue), GAT (pink), Flip-GCN (o) in the original space (red), and Flip-GCN (f) in the flipped space (green) are depicted with different colors. The x-axis represents the training epochs, and the y-axis shows the node classification accuracy.

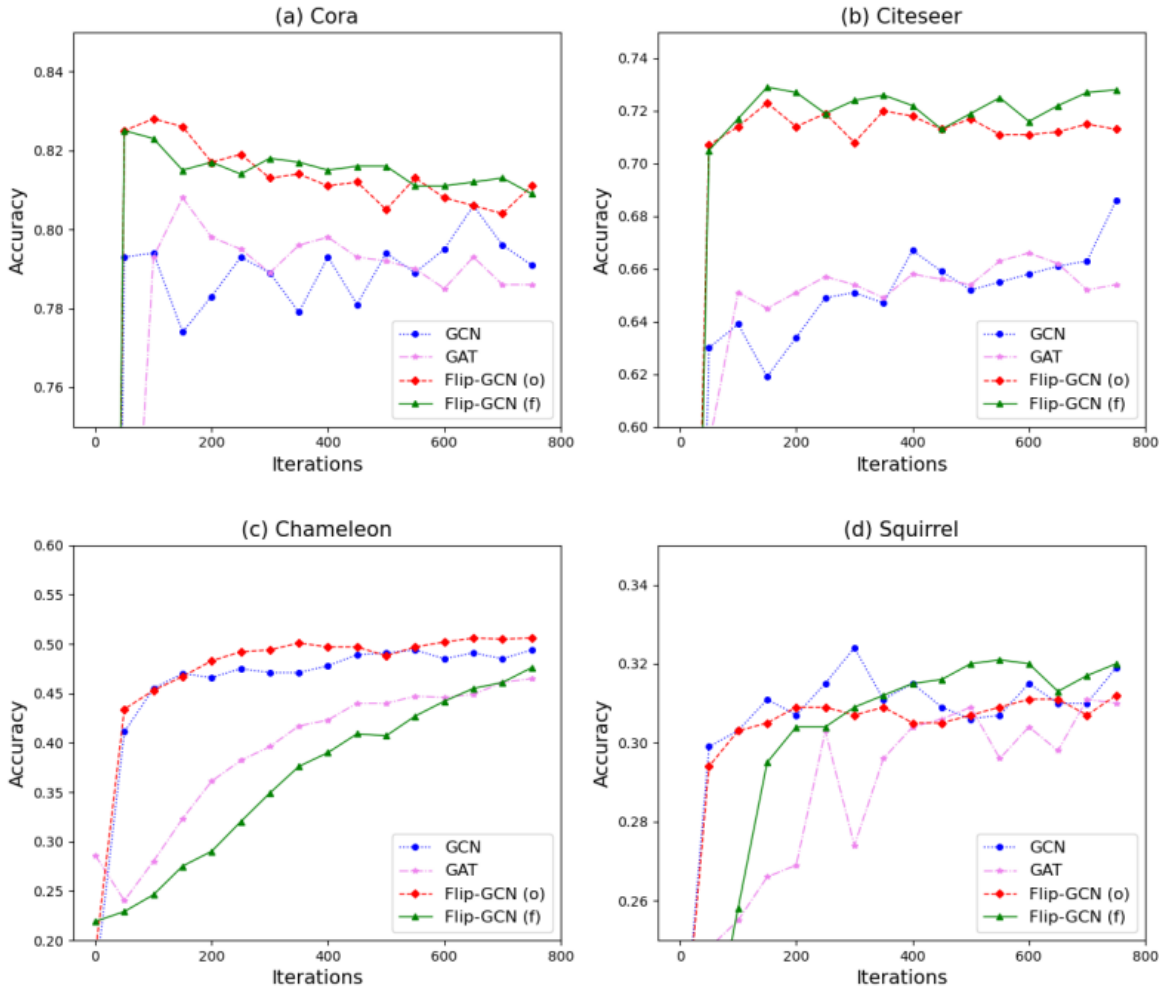


Figure 5.4 (RQ2) Performance of GCN, GAT, and Flip-GCN for each iteration. The performance of Flip-GCN is measured in the original (o) and flipped (f) space, respectively.

Through this figure, we can see that the Flip-GCN achieves higher performance in both spaces. On the Chameleon graph, we notice that Flip-GCN (f) surpasses the baselines after 1000 epochs. Compared to GCN and GAT, the flip-based method demonstrates stability and fast convergence, as seen in the Chameleon and Squirrel datasets. The results confirm our analysis which asserts that flipping reduces the variance of predictions as described in Section 5.4. Though we admit that the performance gain of Flip-GCN is dependent on the type of initial features, flipping leads to the faster and more stable convergence of the parameters. In conclusion, as a data augmentation strategy, flipping leads to improved performance on datasets with

multiple zero elements while ensuring robustness, which is an important characteristic in semi-supervised settings.

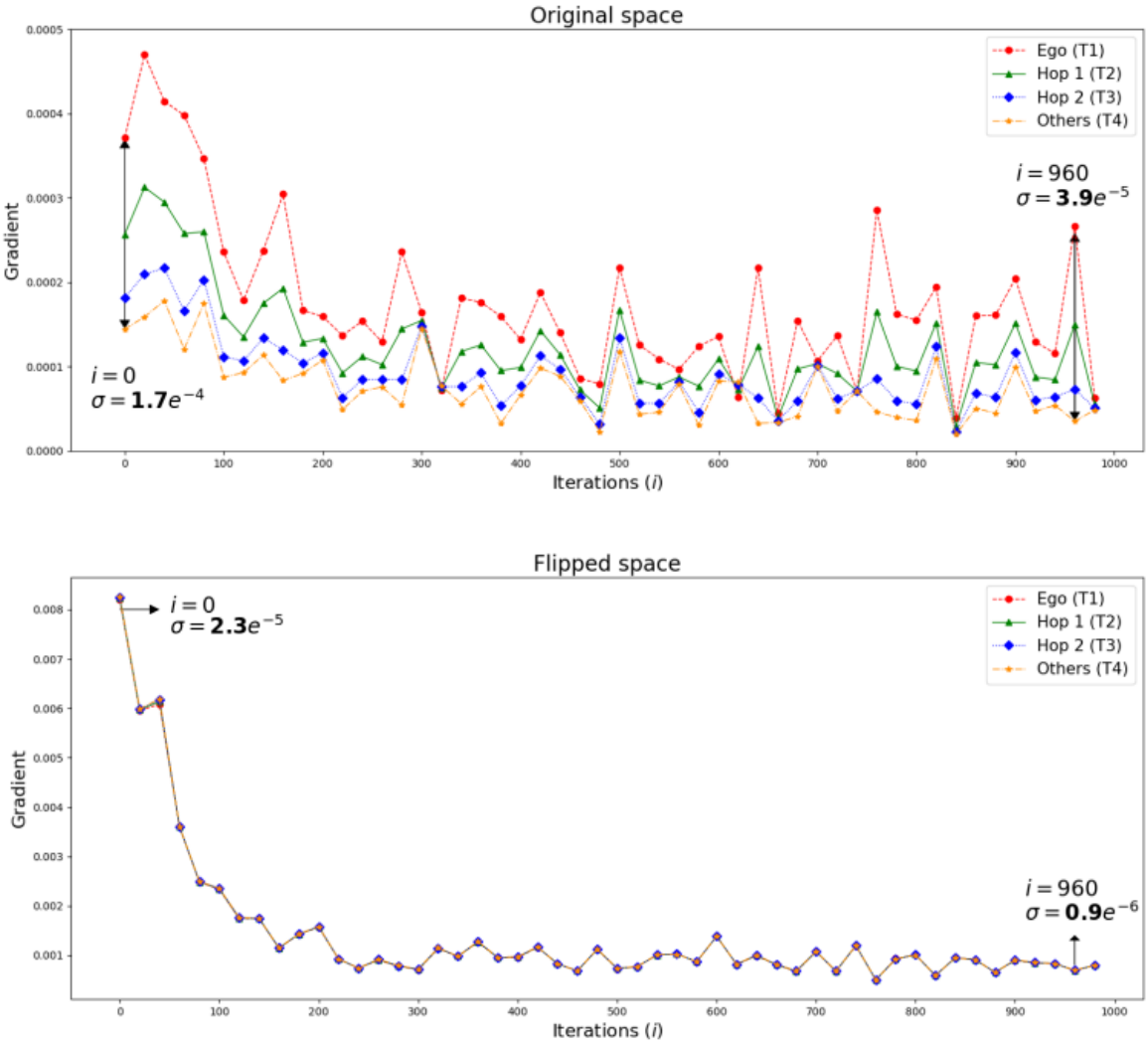


Figure 5.5 (RQ3) Using the Cora dataset, we plot the magnitude of the first projection matrix gradients and their standard deviation (σ) during training epochs (i).

5.5.4 Analysis of Gradients on Two Spaces (RQ3).

Figure 5.5 analyzes the gradient of the first projection matrix during the training phase with the Cora dataset. We define four neighbor types applying different ranges of neighboring: T1, T2, T3, and T4. T1 only consists of the features of the central node (Ego). T2 and T3 include the features of 1-hop and 2-hop neighbors, respectively. T4

has the remaining feature. We prioritize the types from T1 to T4 to avoid overlapping and double-counting of features (note that all $T \in \mathrm{R}^F$ are binarized vectors).

In Figure 5.5, the average and standard deviation of gradients in two spaces are plotted. In the original space (left), the largest gradient is given to features from T1 (red). This is due to the property of GCN, where the gradients generally decrease w.r.t. the hop counts. Also, the features in T4 (orange) have the smallest values, suggesting that they are mostly excluded during training, while only slight updates by weight regularization. On the other hand, in the flipped space (right), all types tend to have a similar magnitude with a small deviation (σ). The results indicate that most dimensions are updated during training in the flipped space.

Dataset	Cora			Chameleon			Cornell		
L/C	20	40	80	20	40	80	5	10	20
z	0.59	0.76	0.91	1	1	1	0.62	0.72	0.84
MLP	53.2	56.9	62.1	41.2	46.3	49.1	60.1	68.8	73.3
MLP+F	61.4	65.5	70.4	43.5	49.9	51.8	**70.5**	**86.4**	**95.8**
GCN	79.1	82.8	83.4	49.4	53.5	55.7	39.4	50.7	53.3
GCN+F	82.7	**84.5**	**85.5**	**50.4**	**54.1**	**55.9**	49.4	54.9	72.8
GAT	80.1	82.4	83.0	46.9	52.4	54.1	42.1	48.9	53.3
GAT+F	**83.1**	84.0	84.6	48.3	52.9	54.4	51.9	59.5	64.4

Table 5.3 (RQ4) Node classification accuracy (%) w.r.t. the different number of training samples. The symbol (+F) means that flipping is applied on a base method.

5.5.5 Varying the Size of Training Samples (RQ4).

In this experiment, we aim to investigate the impact of labeled sample size on performance. Table 5.3 displays the z-value of ego nodes varying the number of labeled nodes per class (L/C) for three graphs. Here, we adjust the number of training samples to analyze the effect of the size of labeled nodes on performance.

Firstly, we can see that GCN and GAT outperform MLP for Cora and Chameleon, while MLP surpasses two models in the Cornell dataset. Apart from this, we observe that the performance improvement from flipping decreases as the L/C increases. This is because, as more training nodes are available, the initial features start to cover most dimensions (high z-value) and the plain models can effectively update the first weight matrix without flipping.

In the Chameleon graph, flipping does not have a significant impact on performance as the number of samples increases. This is because the initial features of the dataset contain many non-zero components and have high maximum values. And as the number of labeled nodes increases, the performance of the vanilla also increases. The same trend can be observed for GAT, where the performance gap between GAT and GAT+F becomes smaller as the number of labeled nodes per class (L/C) increases. However, flipping still improves the performance of the base models in other graphs (Cora and Cornell) significantly.

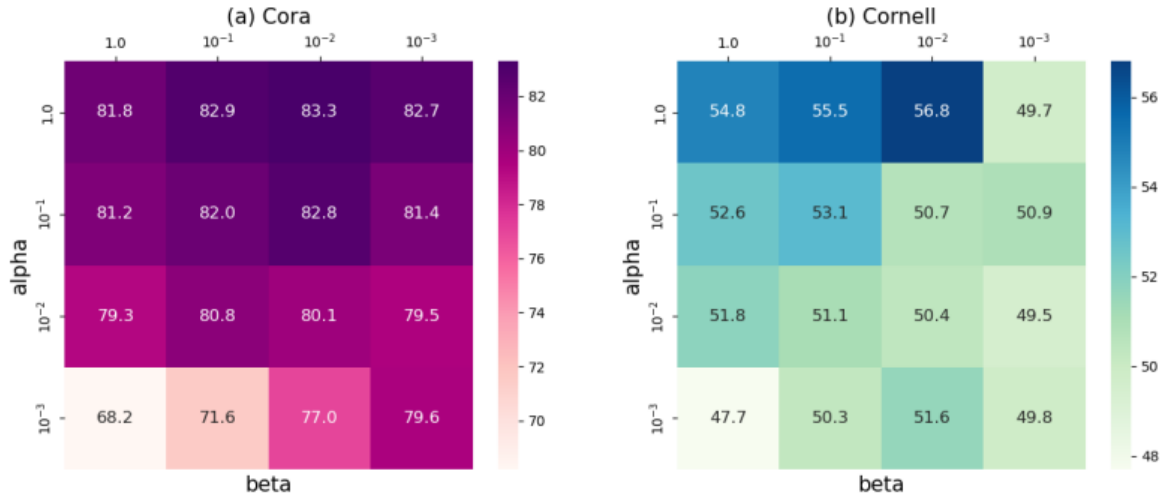


Table 5.6 (RQ4) Parameter sensitivity analysis using Flip-APPNP as a base model

5.5.5 Varying the Size of Training Samples (RQ5).

We investigated how two hyperparameters, α and β in Eq. 21, affect the overall performance of our model. In Figure 8, we illustrate the node classification accuracy of Flip-APPNP by changing α and β (relative weights of the gradient in two types of spaces) through grid search. As can be seen, we employ two types of datasets: Cora and Cornell. Flip-APPNP generally outperforms plain APPNP when α is close to 1. Since the original space allows for fast optimization with a small number of elements, the performance decreases in proportion to α. Furthermore, we noticed that assigning small values to β achieves better performance, where the scale of gradients in a flipped space is generally larger than the original ones (please refer to Fig. 7).

5.6 Related Work

Graph Neural Networks. Generally, GNNs can be divided into two categories: spectral-based and spatial-based. Spectral-based GNN is based on the mathematical foundation for graph convolution in the spectral domain using the Laplacian matrix [5; 16; 21]. On the other hand, spatial-based GNNs aggregate information from local neighborhoods from a spatial perspective, leading to the development of many aggregation schemes for handling noisy connections [81; 67; 99; 13; 2]. The issue of sparse initial features, however, has not received much attention in the literature.

Generalization of neural networks. In the field of neural network generalization, many approaches have been proposed [8; 25; 84]. Several suggested the normalization of deep neural networks [35] while others applied regularization to all adjacent nodes [90] or integrated label propagation to give further information [82]. More recently, the orthogonal GCN [30] attacks the gradient vanishing problem at the initial few layers of GNNs. RawlsGCN [38] claims the unfairness of gradient update which is biased to nodes with a large degree. Though these methods show notable improvements under the semi-supervised scenario, they fail to solve the problem that is inherently occurred by a characteristic of initial features. In this paper, we solve this problem through a simple yet effective method, flipping.

5.7 Conclusion

Existing GNNs have primarily focused on optimizing the aggregation strategy while neglecting the type of initial features. In this paper, we examine the correlation between zero elements in input vectors and their impact on the first layer of neural networks. We introduce a co-training approach that involves learning the gradient flows in both the original and flipped spaces, and adaptively adjusting the parameters. Additionally, we provide a theoretical understanding that flipping reduces prediction variance while maintaining stable convergence. By incorporating flipping into three base methods, we observe an improvement in node classification accuracy, demonstrating that our approach is scalable and effective. In future work, we hope to apply flipping to other variations of GNNs to enhance their performance.

Bibliography

[1] BARANWAL, A., FOUNTOULAKIS, K., AND JAGANNATH, A. Graph convolution for semi-supervised classification: Improved linear separability and out-of-distribution generalization. arXiv preprint arXiv:2102.06966 (2021).

[2] BO, D., WANG, X., SHI, C., AND SHEN, H. Beyond low-frequency information in graph convolutional networks. arXiv preprint arXiv:2101.00797 (2021).

[3] BODNAR, C., DI GIOVANNI, F., CHAMBERLAIN, B. P., LIO`, P., AND BRONSTEIN, M. M. Neural sheaf diffusion: A topological perspective on heterophily and oversmoothing in gnns. arXiv preprint arXiv:2202.04579 (2022).

[4] BRODY, S., ALON, U., AND YAHAV, E. How attentive are graph attention networks? arXiv preprint arXiv:2105.14491 (2021).

[5] BRUNA, J., ZAREMBA, W., SZLAM, A., AND LECUN, Y. Spectral networks and locally connected networks on graphs. arXiv preprint arXiv:1312.6203 (2013).

[6] BUI, T. D., RAVI, S., AND RAMAVAJJALA, V. Neural graph learning: Training neural networks using graphs. In Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining (2018), pp. 64–71.

[7] CAI, T., LUO, S., XU, K., HE, D., LIU, T.-Y., AND WANG, L. Graphnorm: A principled approach to accelerating graph neural network training. In International Conference on Machine Learning (2021), PMLR, pp. 1204–1215.

[8] CHEN, J., ZHU, J., AND SONG, L. Stochastic training of graph convolutional networks with variance reduction. arXiv preprint arXiv:1710.10568 (2017).

[9] CHEN, M., WEI, Z., HUANG, Z., DING, B., AND LI, Y. Simple and deep graph convolutional networks. In International Conference on Machine Learning (2020), PMLR, pp. 1725–1735.

[10] CHEN, S., DOBRIBAN, E., AND LEE, J. H. A group-theoretic framework for data augmentation. The Journal of Machine Learning Research 21, 1 (2020), 9885–9955.

[11] CHEN, Y., LI, J., XIAO, H., JIN, X., YAN, S., AND FENG, J. Dual path networks. Advances in neural information processing systems 30 (2017).

[12] CHEN, Z., MA, T., AND WANG, Y. When does a spectral graph neural network fail in node classification? arXiv preprint arXiv:2202.07902 (2022).

[13] CHIEN, E., PENG, J., LI, P., AND MILENKOVIC, O. Adaptive universal generalized pagerank graph neural network. arXiv preprint arXiv:2006.07988 (2020).

[14] CHOI, Y., CHOI, J., KO, T., BYUN, H., AND KIM, C.-K. Finding heterophilic neighbors via confidence-based subgraph matching for semi-supervised node classification. In Proceedings of the 31st ACM International Conference on Information & Knowledge Management (2022), pp. 283–292.

[15] CUTURI, M., AND DOUCET, A. Fast computation of wasserstein barycenters. In International conference on machine learning (2014), PMLR, pp. 685–693.

[16] DEFFERRARD, M., BRESSON, X., AND VANDERGHEYNST, P. Convolutional neural networks on graphs with fast localized spectral filtering. Advances in neural information processing systems 29 (2016).

[17] DERR, T., MA, Y., AND TANG, J. Signed graph convolutional networks. In 2018 IEEE International Conference on Data Mining (ICDM) (2018), IEEE, pp. 929–934.

[18] DESAI, U., BANDYOPADHYAY, S., AND TAMILSELVAM, S. Graph neural network to dilute outliers for refactoring monolith application. In Proceedings of the AAAI Conference on Artificial Intelligence (2021), vol. 35, pp. 72–80.

[19] DEVRIES, T., AND TAYLOR, G. W. Learning confidence for out-of-distribution detection in neural networks. arXiv preprint arXiv:1802.04865 (2018).

[20] DI GIOVANNI, F., ROWBOTTOM, J., CHAMBERLAIN, B. P., MARKOVICH, T., AND BRONSTEIN, M. M. Graph neural networks as gradient flows. arXiv preprint arXiv:2206.10991 (2022).

[21] DONG, Y., DING, K., JALAIAN, B., JI, S., AND LI, J. Graph neural networks with adaptive frequency response filter. arXiv preprint arXiv:2104.12840 (2021).

[22] ENTEZARI, N., AL-SAYOURI, S. A., DARVISHZADEH, A., AND PAPALEXAKIS, E. E. All you need is low (rank) defending against adversarial attacks on graphs. In Proceedings of the 13th International Conference on Web Search and Data Mining (2020), pp. 169–177.

[23] FAN, W., MA, Y., LI, Q., HE, Y., ZHAO, E., TANG, J., AND YIN, D. Graph neural networks for social recommendation. In The world wide web conference (2019), pp. 417–426.

[24] FANG, Z., XU, L., SONG, G., LONG, Q., AND ZHANG, Y. Polarized graph neural networks. In Proceedings of the ACM Web Conference 2022 (2022), pp. 1404–1413.

[25] FENG, J., AND SIMON, N. Sparse-input neural networks for high-dimensional nonparametric regression and classification. arXiv preprint arXiv:1711.07592 (2017).

[26] FOUT, A., BYRD, J., SHARIAT, B., AND BEN-HUR, A. Protein interface prediction using graph convolutional networks. Advances in neural information processing systems 30 (2017).

[27] FRIEDKIN, N. E. A structural theory of social influence. Cambridge University Press, 1998.

[28] GILMER, J., SCHOENHOLZ, S. S., RILEY, P. F., VINYALS, O., AND DAHL, G. E. Neural message passing for quantum chemistry. In International conference on machine learning (2017), PMLR, pp. 1263–1272.

[29] GUO, C., PLEISS, G., SUN, Y., AND WEINBERGER, K. Q. On calibration of modern neural networks. In International conference on machine learning (2017), PMLR, pp. 1321–1330.

[30] GUO, K., ZHOU, K., HU, X., LI, Y., CHANG, Y., AND WANG, X. Orthogonal graph neural networks. In Proceedings of the AAAI Conference on Artificial Intelligence (2022), vol. 36, pp. 3996–4004.

[31] GUO, Y., AND WEI, Z. Clenshaw graph neural networks. arXiv preprint arXiv:2210.16508 (2022).

[32] HAMILTON, W., YING, Z., AND LESKOVEC, J. Inductive representation learning on large graphs. Advances in neural information processing systems 30 (2017).

[33] HAMMOND, D. K., VANDERGHEYNST, P., AND GRIBONVAL, R. Wavelets on graphs via spectral graph theory. Applied and Computational Harmonic Analysis 30, 2 (2011), 129–150.

[34] HUANG, J., SHEN, H., HOU, L., AND CHENG, X. Signed graph attention networks. In International Conference on Artificial Neural Networks (2019), Springer, pp. 566–577.

[35] HUANG, L., QIN, J., ZHOU, Y., ZHU, F., LIU, L., AND SHAO, L. Normalization techniques in training dnns: Methodology, analysis and application. arXiv preprint arXiv:2009.12836 (2020).

[36] HUANG, T., WANG, D., AND FANG, Y. End-to-end open-set semi-supervised node classification with out-ofdistribution detection. In Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI22 (2022), IJCAI.

[37] JIN, W., DERR, T., WANG, Y., MA, Y., LIU, Z., AND TANG, J. Node similarity preserving graph convolutional networks. In Proceedings of the 14th ACM International Conference on Web Search and Data Mining (2021), pp. 148– 156.

[38] KANG, J., ZHU, Y., XIA, Y., LUO, J., AND TONG, H. Rawlsgcn: Towards rawlsian difference principle on graph convolutional network. In Proceedings of the ACM Web Conference 2022 (2022), pp. 1214–1225.

[39] KENDALL, A., AND GAL, Y. What uncertainties do we need in bayesian deep learning for computer vision? Advances in neural information processing systems 30 (2017).

[40] KIM, D., AND OH, A. How to find your friendly neighborhood: Graph attention design with self-supervision. arXiv preprint arXiv:2204.04879 (2022).

[41] KIM, H., RASCH, M., GOKMEN, T., ANDO, T., MIYAZOE, H., KIM, J.-J., ROZEN, J., AND KIM, S. Zero-shifting technique for deep neural network training on resistive cross-point arrays. arXiv preprint arXiv:1907.10228 (2019).

[42] KIPF, T. N., AND WELLING, M. Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907 (2016).

[43] KLICPERA, J., BOJCHEVSKI, A., AND GUNNEMANN¨ , S. Predict then propagate: Graph neural networks meet personalized pagerank. arXiv preprint arXiv:1810.05997 (2018).

[44] KOLOURI, S., NADERIALIZADEH, N., ROHDE, G. K., AND HOFFMANN, H. Wasserstein embedding for graph learning. arXiv preprint arXiv:2006.09430 (2020).

[45] LECUN, Y., BENGIO, Y., AND HINTON, G. Deep learning. nature 521, 7553 (2015), 436–444.

[46] LEI, R., WANG, Z., LI, Y., DING, B., AND WEI, Z. Evennet: Ignoring odd-hop neighbors improves robustness of graph neural networks. arXiv preprint arXiv:2205.13892 (2022).

[47] LI, Q., HAN, Z., AND WU, X.-M. Deeper insights into graph convolutional networks for semi-supervised learning. In Thirty-Second AAAI conference on artificial intelligence (2018).

[48] LI, X., ZHU, R., CHENG, Y., SHAN, C., LUO, S., LI, D., AND QIAN, W. Finding global homophily in graph neural networks when meeting heterophily. arXiv preprint arXiv:2205.07308 (2022).

[49] LI, Y., AND YUAN, Y. Convergence analysis of two-layer neural networks with relu activation. Advances in neural information processing systems 30 (2017).

[50] LIM, D., HOHNE, F., LI, X., HUANG, S. L., GUPTA, V., BHALERAO, O., AND LIM, S. N. Large scale learning on non-homophilous graphs: New benchmarks and strong simple methods. Advances in Neural Information Processing Systems 34 (2021), 20887–20902.

[51] LIN, M., JI, R., XU, Z., ZHANG, B., WANG, Y., WU, Y., HUANG, F., AND LIN, C.-W. Rotated binary neural network. Advances in neural information processing systems 33 (2020), 7474–7485.

[52] LIU, H., HU, B., WANG, X., SHI, C., ZHANG, Z., AND ZHOU, J. Confidence may cheat: Self-training on graph neural networks under distribution shift. In Proceedings of the ACM Web Conference 2022 (2022), pp. 1248–1258.

[53] LIU, M., WANG, Z., AND JI, S. Non-local graph neural networks. IEEE Transactions on Pattern Analysis and Machine Intelligence (2021).

[54] LUAN, S., HUA, C., LU, Q., ZHU, J., ZHAO, M., ZHANG, S., CHANG, X.-W., AND PRECUP, D. Is heterophily a real nightmare for graph neural networks to do node classification? arXiv preprint arXiv:2109.05641 (2021).

[55] LUO, D., CHENG, W., YU, W., ZONG, B., NI, J., CHEN, H., AND ZHANG, X. Learning to drop: Robust graph neural network via topological denoising. In Proceedings of the 14th ACM International Conference on Web Search and Data Mining (2021), pp. 779–787.

[56] LUO, G., LI, J., SU, J., PENG, H., YANG, C., SUN, L., YU, P. S., AND HE, L. Graph entropy guided node embedding dimension selection for graph neural networks. arXiv preprint arXiv:2105.03178 (2021).

[57] MA, Y., LIU, X., SHAH, N., AND TANG, J. Is homophily a necessity for graph neural networks? arXiv preprint arXiv:2106.06134 (2021).

[58] MCCALLUM, A. K., NIGAM, K., RENNIE, J., AND SEYMORE, K. Automating the construction of internet portals with machine learning. Information Retrieval 3, 2 (2000), 127–163.

[59] MCPHERSON, M., SMITH-LOVIN, L., AND COOK, J. M. Birds of a feather: Homophily in social networks. Annual review of sociology 27, 1 (2001), 415–444.

[60] MIALON, G., CHEN, D., D'ASPREMONT, A., AND MAIRAL, J. A trainable optimal transport embedding for feature aggregation and its relationship to attention. arXiv preprint arXiv:2006.12065 (2020).

[61] MOON, J., KIM, J., SHIN, Y., AND HWANG, S. Confidence-aware learning for deep neural networks. In international conference on machine learning (2020), PMLR, pp. 7034–7044.

[62] MUKHERJEE, S., AND AWADALLAH, A. Uncertainty-aware self-training for few-shot text classification. Advances in Neural Information Processing Systems 33 (2020), 21199–21212.

[63] NEWMAN, M. E. Assortative mixing in networks. Physical review letters 89, 20 (2002), 208701.

[64] NT, H., AND MAEHARA, T. Revisiting graph neural networks: All we have is low-pass filters. arXiv preprint arXiv:1905.09550 (2019).

[65] OONO, K., AND SUZUKI, T. Graph neural networks exponentially lose expressive power for node classification. arXiv preprint arXiv:1905.10947 (2019).

[66] PANDIT, S., CHAU, D. H., WANG, S., AND FALOUTSOS, C. Netprobe: a fast and scalable system for fraud detection in online auction networks. In Proceedings of the 16th international conference on World Wide Web (2007), pp. 201–210.

[67] PEI, H., WEI, B., CHANG, K. C.-C., LEI, Y., AND YANG, B. Geom-gcn: Geometric graph convolutional networks. arXiv preprint arXiv:2002.05287 (2020).

[68] PERRY, C. Machine learning and conflict prediction: a use case. Stability: International Journal of Security and Development 2, 3 (2013), 56.

[69] PEYRE´, G., CUTURI, M., ET AL. Computational optimal transport: With applications to data science. Foundations and Trends® in Machine Learning 11, 5-6 (2019), 355–607.

[70] POPESCU, M.-C., BALAS, V. E., PERESCU-POPESCU, L., AND MASTORAKIS, N. Multilayer perceptron and neural networks. WSEAS Transactions on Circuits and Systems 8, 7 (2009), 579–588.

[71] RONG, Y., HUANG, W., XU, T., AND HUANG, J. Dropedge: Towards deep graph convolutional networks on node classification. arXiv preprint arXiv:1907.10903 (2019).

[72] ROZEMBERCZKI, B., DAVIES, R., SARKAR, R., AND SUTTON, C. Gemsec: Graph embedding with self clustering. In Proceedings of the 2019 IEEE/ACM international conference on advances in social networks analysis and mining (2019), pp. 65–72.

[73] SCARSELLI, F., GORI, M., TSOI, A. C., HAGENBUCHNER, M., AND MONFARDINI, G. The graph neural network model. IEEE transactions on neural networks 20, 1 (2008), 61–80.

[74] SHANNON, C. E. A mathematical theory of communication. The Bell system technical journal 27, 3 (1948), 379–423.

[75] SHIJIE, J., PING, W., PEIYI, J., AND SIPING, H. Research on data augmentation for image classification based on convolution neural networks. In 2017 Chinese automation congress (CAC) (2017), IEEE, pp. 4165–4170.

[76] SINKHORN, R., AND KNOPP, P. Concerning nonnegative matrices and doubly stochastic matrices. Pacific Journal of Mathematics 21, 2 (1967), 343–348.

[77] SONG, Z., ZHANG, Y., AND KING, I. Towards an optimal asymmetric graph structure for robust semi-supervised node classification. In Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (2022), pp. 1656–1665.

[78] STRETCU, O., VISWANATHAN, K., MOVSHOVITZ-ATTIAS, D., PLATANIOS, E., RAVI, S., AND TOMKINS, A. Graph agreement models for semi-supervised learning. Advances in Neural Information Processing Systems 32 (2019).

[79] SUN, Y., DENG, H., YANG, Y., WANG, C., XU, J., HUANG, R., CAO, L., WANG, Y., AND CHEN, L. Beyond homophily: Structure-aware path aggregation graph neural network. In Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22 (7 2022), L. D. Raedt, Ed., International Joint Conferences on Artificial Intelligence Organization, pp. 2233–2240. Main Track.

[80] TANG, J., SUN, J., WANG, C., AND YANG, Z. Social influence analysis in large-scale networks. In Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining (2009), pp. 807–816.

[81] VELICKOVIC, P., CUCURULL, G., CASANOVA, A., ROMERO, A., LIO, P., AND BENGIO, Y. Graph attention networks. stat 1050 (2017), 20.

[82] WANG, H., AND LESKOVEC, J. Unifying graph convolutional neural networks and label propagation. arXiv preprint arXiv:2002.06755 (2020).

[83] WANG, X., GIRSHICK, R., GUPTA, A., AND HE, K. Non-local neural networks. In Proceedings of the IEEE conference on computer vision and pattern recognition (2018), pp. 7794–7803.

[84] WANG, X., LIU, H., SHI, C., AND YANG, C. Be confident! towards trustworthy graph neural networks via confidence calibration. Advances in Neural Information Processing Systems 34 (2021), 23768–23779.

[85] XIAO, T., CHEN, Z., WANG, D., AND WANG, S. Learning how to propagate messages in graph neural networks. In Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining (2021), pp. 1894–1903.

[86] XU, H., LUO, D., AND CARIN, L. Scalable gromov-wasserstein learning for graph partitioning and matching. Advances in neural information processing systems 32 (2019).

[87] XU, H., LUO, D., ZHA, H., AND DUKE, L. C. Gromov-wasserstein learning for graph matching and node embedding. In International conference on machine learning (2019), PMLR, pp. 6932–6941.

[88] XU, K., HU, W., LESKOVEC, J., AND JEGELKA, S. How powerful are graph neural networks? arXiv preprint arXiv:1810.00826 (2018).

[89] YAN, Y., HASHEMI, M., SWERSKY, K., YANG, Y., AND KOUTRA, D. Two sides of the same coin: Heterophily and oversmoothing in graph convolutional neural networks. arXiv preprint arXiv:2102.06462 (2021).

[90] YANG, H., MA, K., AND CHENG, J. Rethinking graph regularization for graph neural networks. In Proceedings of the AAAI Conference on Artificial Intelligence (2021), vol. 35, pp. 4573–4581.

[91] YANG, L., LI, M., LIU, L., WANG, C., CAO, X., GUO, Y., ET AL. Diverse message passing for attribute with heterophily. Advances in Neural Information Processing Systems 34 (2021), 4751–4763.

[92] YANG, L., WU, F., WANG, Y., GU, J., AND GUO, Y. Masked graph convolutional network. In IJCAI (2019), pp. 4070– 4077.

[93] YANG, T., WANG, Y., YUE, Z., YANG, Y., TONG, Y., AND BAI, J. Graph pointer neural networks. arXiv preprint arXiv:2110.00973 (2021).

[94] YING, Z., BOURGEOIS, D., YOU, J., ZITNIK, M., AND LESKOVEC, J. Gnnexplainer: Generating explanations for graph neural networks. Advances in neural information processing systems 32 (2019).

[95] ZHANG, X., LIU, L., XIE, Y., CHEN, J., WU, L., AND PIETIKAINEN, M. Rotation invariant local binary convolution neural networks. In Proceedings of the IEEE International Conference on Computer Vision Workshops (2017), pp. 1210–1219.

[96] ZHAO, L., AND AKOGLU, L. Pairnorm: Tackling oversmoothing in gnns. arXiv preprint arXiv:1909.12223 (2019).

[97] ZHAO, X., CHEN, F., HU, S., AND CHO, J.-H. Uncertainty aware semi-supervised learning on graph data. Advances in Neural Information Processing Systems 33 (2020), 12827–12836.

[98] ZHENG, C., ZONG, B., CHENG, W., SONG, D., NI, J., YU, W., CHEN, H., AND WANG, W. Robust graph representation learning via neural sparsification. In International Conference on Machine Learning (2020), PMLR, pp. 11458– 11468.

[99] ZHU, J., YAN, Y., ZHAO, L., HEIMANN, M., AKOGLU, L., AND KOUTRA, D. Beyond homophily in graph neural networks: Current limitations and

effective designs. Advances in Neural Information Processing Systems 33 (2020), 7793–7804.

[100] ZHU, Y., XU, Y., YU, F., LIU, Q., WU, S., AND WANG, L. Graph contrastive learning with adaptive augmentation. In Proceedings of the Web Conference 2021 (2021), pp. 2069–2080.