Robust Graph Neural Networks via Adaptive Aggregator Selection and Long-range Dependency Modeling

---

A

Thesis

Presented to

the faculty of the School of Engineering and Applied Science

University of Virginia

---

in partial fulfillment

of the requirements for the degree

Master of Science

by

Sikun Guo

December 2022

APPROVAL SHEET

This

Thesis

is submitted in partial fulfillment of the requirements
for the degree of

Master of Science

Author: Sikun Guo

This Thesis has been read and approved by the examing committee:

Advisor: Hongning Wang

Advisor:

Committee Member: Yangfeng Ji

Committee Member: Jundong Li

Committee Member:

Committee Member:

Committee Member:

Committee Member:

Accepted for the School of Engineering and Applied Science:

Jennifer L. West, School of Engineering and Applied Science

December 2022

To my parents, teachers and friends.

Acknowledgements

I would like to express my deepest gratitude to Professor Hongning Wang for advising me along my Master's Thesis research journey. Discussions between us can always generate new ideas and reflections for our project. I not only learn how to get inspiration from others' work but also try to get into the habit of thinking about the essence of a given problem, which is crucial for my future study. Besides research, Professor Wang is also a good friend who offers fantastic help when I encounter life difficulties and support for my extracurricular hobbies like singing. I would like to express my sincere gratitude to my Thesis committee: Professor Yangfeng Ji and Professor Jundong Li. Your suggestions about improving my current research are all to the point and enlightening.

Many thanks to Dr. Lu Lin for her unreserved support for my research and tons of thoughtful discussions. As a senior Ph.D. student when I first joined our research group, she shared her insights of researching on graph mining and taught me how to use the department servers. Without her support and help, I could not have conducted my current research. Best wishes to her faculty journey at Pennsylvania State University. Many thanks to everyone at HCDM lab, our lab makes me feel at home.

The endeavor of my life would not have been possible without unconditional love and support from my parents and other family members. Though on opposite sides of the earth, regular video calls can allow for a family reunion and bring their love to me. Please forgive me when I always share the good news with you but not the bad ones.

Last but not least, I want to thank all my friends in the United States. You help me fit into the new environment and make my life much more colorful.

Abstract

Deep learning on graphs has received increasing attention in recent years, and Graph Neural Networks (GNNs) have achieved remarkable performance across various tasks on graphs like node classification and link prediction, etc. However, recent studies show that GNNs can be vulnerable to adversarial graph structure perturbations. To address this issue, we propose GalNN, a GNN architecture based on adaptive aggregator selection and long-range dependency modeling, which is robust to poisoning attack on both homophily graphs and heterophily graphs. The general principle of adaptive aggregator selection is to learn each neighborhood's profile via the centering node's degree and neighborhood feature variance to guide the aggregation with a set of aggregators. Long-range dependency modeling adopts the manifold learning idea to locate similar nodes in the neighboring regions of a 1D manifold and simulates the aggregation operation with 1D convolution. Our results show that the learned neighborhood profiles can differentiate perturbed neighborhoods from clean neighborhoods on attacked homophilic graphs, and the long-range dependency modeling can make the original heterophily graphs more homophilic. Fusing the information from adaptive aggregator selection and long-range dependency modeling realizes robust node embedding learning.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Graph-structured data are ubiquitous in the real world, ranging from social networks to chemical molecules [14]. The ubiquity has attracted research attention to analyze graphs and conduct data mining on graphs. Currently, the most popular graph mining method is through Graph Neural Networks (GNNs) like GCN [20], GAT [34], and GraphSAGE [13]. GNNs are extended from traditional Deep Neural Networks (DNNs), and their power lies in their ability to leverage node features and graph topology simultaneously. Many GNNs follow a message-passing, aggregate, and update paradigm, allowing nodes on graphs to exchange information with their neighboring nodes and then use the information to update their node embeddings. By stacking GNN layers, each node on the graph can potentially exchange information with neighbors a few hops away from it. Due to the powerful representation learning ability on graphs, GNNs have gained remarkable success across a wide range of applications in computer vision [30], natural language processing [24], social network analysis [29], recommender systems [38], physics [31], chemistry [12], and other fields [8, 5, 15, 21].

However, machine learning methods can suffer from vulnerabilities of adversarial attacks [1, 11, 3], which means unnoticeable perturbations may compromise the models and significantly degrade the performance on downstream tasks. Recent studies indicate that Graph Neural Networks are not immune to adversarial attacks as well [36, 17]. Adversarial attacks on graphs often focus on manipulating the connectiv-

ity of the graphs. That is, adding, deleting, or rewiring edges among the nodes in an imperceptible way. For example, as is shown in Figure 1-1, slightly rewiring the connectivity around node 5 by deleting the edge to node 4 and adding the edge to node 3, a GNN can misclassify node 5 from an orange node to a blue node. Besides, the effectiveness of adversarial attacks on graphs often inherit transferability, which indicates that an effective attack aiming at one GNN victim model is also very likely to compromise other GNN models or other graph mining methods [25]. The fragility of GNNs and the transferability of attacks make GNNs less robust in the adversarial setting, leading to huge trustworthiness issues when applying them to real-world problems, especially problems requiring high-security guarantees. For instance, in the finance credit-scoring system, a fraud user may pretend to be a normal user by increasing its connectivity with high credit normal users to evade fraud detectors based on GNNs. In social media, a fake news reporter may associate himself/herself with high-impact Internet influencers to facilitate the spread of fake news. Therefore, it's urgent to study adversarial attacks on graphs and the according countermeasures to increase the robustness of GNNs.

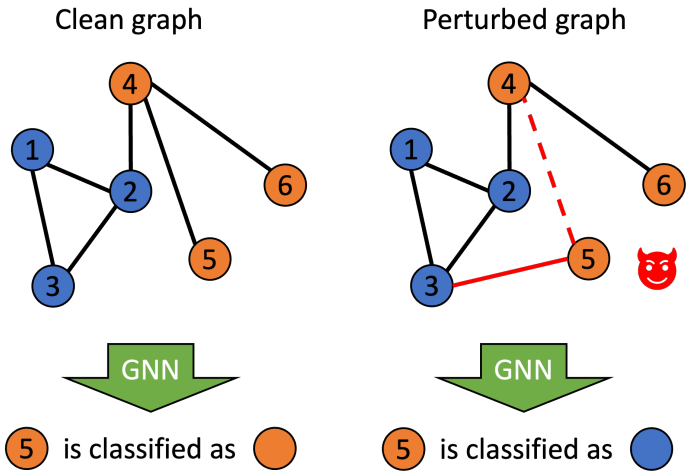

Figure 1-1: Illustration of the adversarial attacks on graph topology and the possible effects. The GNN model deals with a node classification task. The GNN model can correctly classify node 5 as an orange node on the clean graph. However, after an adversary deletes the edge between node 4 and node 5 and adds an edge between node 3 and node 5, node 5 is wrongly classified as a blue node.

To tackle the attackers' behavior and prevent potential harmful effects, plenty of

efforts have been devoted to improving GNNs' adversarial robustness. Existing methods can be categorized into 3 classes. Namely, graph purification, GNN architecture modification, and adversarial training [25]. Some representative methods for graph purification are GCN-SVD [9], GCN-Jaccard [35], and Pro-GNN [18]. GCN-SVD [9] vaccinates GNNs by preprocessing the suspicious graph's adjacency matrix using the singular-value-decomposition-based low-rank approximation. Unlike GCN-SVD [9], which utilizes the low-rank property of clean graph's adjacency matrix, GCN-Jaccard [35] prunes the graph based on the neighboring nodes' Jaccard similarity score. To exploit information from graph structure and node features at the same time, based on the sparse and low-rank assumption for the adjacency matrix and feature smoothness assumption, Pro-GNN [18] adds regularizations to the objective function, aiming to learn the graph structure and robust node representations jointly. Some methods also specify the vulnerabilities in GNN architecture and modify the architecture accordingly. GNNGuard [39] learns to assign higher weights to edges connecting similar nodes while pruning edges between unrelated nodes in the message-passing stage to mitigate potentially harmful effects of adversarial edges. RGCN [41] tries to absorb the effects of adversarial attacks in the aggregation stage by modeling latent node representations as gaussian distributions. It also applies attention mechanism to penalize nodes with high neighborhood feature variance in the message-passing stage. Adversarial training-related methods generally model the defense as a min-max optimization problem, which inserts adversarial edges during training while minimizing the task-specific training loss [37, 7, 16]. More details about adversarial attacks and defenses will be discussed in section 2.3 and section 2.4.

Nevertheless, many existing defense methods have limitations such as partially leveraged information (e.g., GCN-SVD: topology information only, GCN-Jaccard: node feature information only), tuning difficulty (e.g., Pro-GNN), unsatisfactory scalability (e.g., Pro-GNN, GCN-SVD), impractical input assumption (e.g., adversarial training), and performance compromise on clean graphs (e.g., GCN-SVD). Besides, few works consider defense for real-world heterophily graphs since most of them focus on defense for homophily graphs.

To bridge the aforementioned gap, in this work, we propose GalNN, a GNN architecture based on adaptive aggregator selection and long-range dependency modeling, which is robust to poisoning graph structure attacks on both homophily graphs and heterophily graphs. GalNN consists of 3 phases: long-range dependency modeling, adaptive aggregator selection, and robust information fusion. Long-range dependency modeling aims to build an auxiliary graph and provide a set of node embeddings only learned from node features. By ignoring the observed graph's topology, the constructed auxiliary graph and the learned node embeddings can ideally ignore any topology perturbations. For adaptive aggregator selection, GalNN learns neighborhood profiles for centering nodes based on the neighborhood's degree and the node feature variance. In each layer of GalNN, a given node's neighborhood profile consists of weights for 4 aggregators: the mean aggregator, the maximum aggregator, the minimum aggregator, and the median aggregator. The final aggregated output is the weighted average of the outputs from the 4 aggregators guided by the centering nodes' neighborhood profiles. The key insight here is that each aggregator is capable of aggregating helpful information from the neighborhood for updating the centering node's representation, and the effects of adversarial edges can be mitigated by adaptively reweighting the aggregated outputs from each aggregator since it's less likely for adversarial edges to compromise all the aggregators. Robust information fusion revolves around how to combine the information from long-range dependency modeling and adaptive aggregator selection. After we have the auxiliary graph topology and node embeddings from long-range dependency modeling, we have 3 information fusion schemes: neighborhood fusion, embedding fusion, and neighborhood fusion plus embedding fusion. By fusing the information, we can learn node embeddings with robustness guarantees suitable for downstream tasks. We will elaborate more on our proposed method in Section 3. Our contributions can be summarized as follows:

- **Contaminated neighborhoods on homophily graphs can be differentiated from clean neighborhoods by the learned neighborhood profiles, and harmful effects of adversarial edges can be mitigated by the adaptive aggregator selection of GalNN.**

- **Malign damage of adversarial attacks on heterophily graphs can be alleviated by long-range dependency modeling, which also helps construct graphs that are more homophilic to leverage the power of GNNs.**

- **Robust information fusion provides a new defense paradigm that is effective for graphs across the whole homophily score spectrum.**

The rest of the Thesis is organized as follows: In Chapter 2, we introduce the necessary background of this work, in which we will cover the math notations, background for GNNs, related adversarial attacks, and defenses for graph-structured data. Followed by Chapter 3, in which we will elaborate on the details of GalNN's design. We will show GalNN's effectiveness through the experiments and analysis in Chapter 4. In the end, we will discuss the limitations of GalNN and shine a light on some future directions for robust GNNs in Chapter 5.

Chapter 2

Background

In this chapter, we will briefly cover the preliminaries of the Thesis. In Section 2.1, we will cover the necessary math notations of machine learning on graphs for the following discussion. Section 2.2 introduces the background of Graph Neural Networks. Adversarial attacks and defenses on graphs will be outlined in Section 2.3 and Section 2.4.

2.1 Notations

A graph can be represented as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ is the set of nodes, and $\mathcal{E}$ is the set of edges connecting nodes. Each node in $\mathcal{V}$ can be denoted as $v_i \in \mathcal{V}$ and each edge connecting node v_i and v_j can be denoted as $e_{ij} \in \mathcal{E}$. $A \in \mathbb{R}^{N \times N}$ represents the adjacency matrix of graph $\mathcal{G}$, where N is the total number of nodes on graph. If there exists an edge e_{ij}, then $\mathbf{A}_{ij} = 1$, otherwise $\mathbf{A}_{ij} = 0$. Note that we only discuss undirected graphs in this work, so $e_{ij} = e_{ji}$ and $\mathbf{A}_{ij} = \mathbf{A}_{ji}$. We define the 1-hop neighborhood of v_i as $\mathcal{N}(v_i) = \{v_j | e_{ij} \in \mathcal{E}\}$. The summation of elements in the i-th row of $\mathbf{A}$ or the number of nodes in $\mathcal{N}(v_i)$ is defined as the degree of v_i, which is denoted as d_i. Therefore, we can construct the degree matrix as a diagonal matrix $\mathbf{D}$, where $\mathbf{D}_{ii} = d_i$. Nodes on the graph may be associated with node features, which can be represented as a feature matrix $\mathbf{X} \in \mathbb{R}^{N \times F}$, where F is the feature dimensions, and the i-th row of $\mathbf{X}$, $\mathbf{x}_i$, represents the node feature vector for node

v_i. For node classification tasks, each node has a label y_i. With node labels, we can use homophily scores to describe the relationship between graph topology and node labels. The most widely used homophily scores are edge homophily score [43] and node homophily score [28]. Edge homophily score $H_{edge}(\mathcal{G})$ measures the proportion of edges connecting nodes with the same labels:

$$H_{edge}(\mathcal{G}) = \frac{|\{e_{ij}|e_{ij} \in \mathcal{E} \wedge y_i = y_j\}|}{|\mathcal{E}|} \tag{2.1}$$

Local node homophily score $H_{node}(\mathcal{G})$ focuses on a given node v_i and evaluates the proportion of v_i's edges connecting nodes with the same labels as v_i's:

$$H_{node}(v_i) = \frac{|\{e_{ij}|e_{ij} \in \mathcal{N}(v_i) \wedge y_i = y_j\}|}{|\mathcal{N}(v_i)|} \tag{2.2}$$

With the local node homophily score, the global node homophily score can be easily extended as the average of all the nodes' homophily score on a graph $\mathcal{G}$:

$$H_{node}(\mathcal{G}) = \frac{1}{|\mathcal{V}|} \sum_{v_i \in \mathcal{V}} \frac{|\{e_{ij}|e_{ij} \in \mathcal{N}(v_i) \wedge y_i = y_j\}|}{|\mathcal{N}(v_i)|} \tag{2.3}$$

All the homophily scores range from 0 to 1. A homophily score close to 1 indicates strong homophily, while a score close to 0 indicates strong heterophily. Figure 2-1

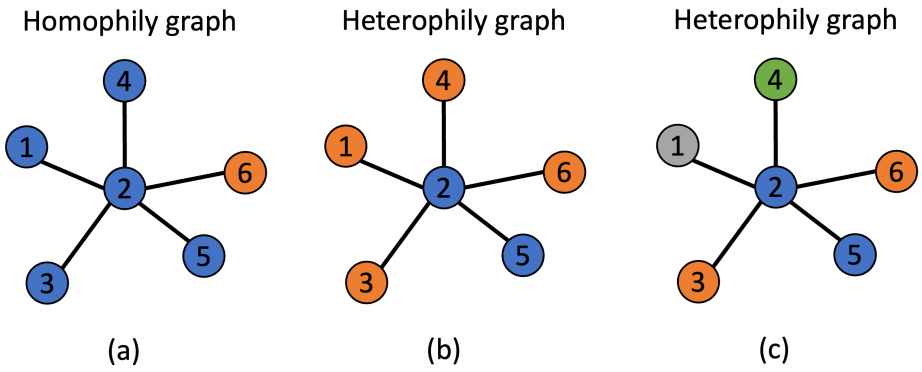

Figure 2-1: Illustration of homophily/heterophily graphs. (a) is a homophily graph with $H_{edge}(\mathcal{G}) = 0.8$. (b) is a heterophily graph with $H_{edge}(\mathcal{G}) = 0.2$. (c) is a heterophily graph with $H_{edge}(\mathcal{G}) = 0.2$, but it has more heterophilic patterns compared to (b).

Including the upcoming ones, all the math notations for the Thesis are summarized in Table 2.1.

2.2 Graph Neural Networks

Inspired by Convolutional Neural Networks (CNNs) in computer vision and natural language processing domain, Graph Neural Networks (GNNs) extend the convolution operations from grid-like Euclidean data to non-Euclidean graph data. The essential idea of Graph Neural Networks is iteratively learning and updating the node embeddings using the information from its neighbors and itself. Though GNNs have many variants like GCN [20], GAT [34], and GraphSAGE [13], the general framework of GNNs' each layer consists of 3 stages: message passing, aggregation, and update.

Message passing: Given a neighborhood $\mathcal{N}(v_i)$ centered at v_i, message passing defines how each neighboring node in $\mathcal{N}(v_i)$ communicates with or how to send message to centering node v_i:

$$\{\mathbf{m}_{ij}^l\} = \mathsf{MESSAGE}(\{\mathbf{h}_j^l | v_j \in \mathcal{N}(v_i)\}) \tag{2.4}$$

Here, $\mathsf{MESSAGE}(\cdot)$ denotes the message passing function. k denotes the GNN's layer index. $\mathbf{m}_{ij}^l$ denotes the message sent to v_i from v_j at layer l. $\mathbf{h}_j^l$ is the node embedding of v_j at layer l. If $k = 1$, then $\mathbf{h}_j^l = \mathbf{x}_j$. Message passing is a per-edge operation.

Aggregation: Having all the $\mathbf{m}_{ij}$ from $\mathcal{N}(v_i)$, v_i needs to further process them to prepare for the update of its embedding. This process can be generalized as the aggregation operation:

$$\mathbf{a}_i^l = \mathsf{AGGREGATE}(\{\mathbf{m}_{ij}^l\}) \tag{2.5}$$

Here, $\mathsf{AGGREGATE}(\cdot)$ denotes the aggregation operation, and $\mathbf{a}_i^l$ denotes the aggregated output for v_i at layer l. Aggregation is a per-neighborhood operation.

Update: The update operation calculates the node embeddings at the next layer

Table 2.1: Notations used throughout the Thesis

Notations	Descriptions
$\mathcal{G} = (\mathcal{V}, \mathcal{E})$ or $\mathcal{G} = (\mathbf{A}, \mathbf{X})$	Graph
$\mathcal{V}$	Node set
v_i	Node i
$\mathcal{E}$	Edge set
e_{ij}	Edge from node i to node j
$\mathbf{A} \in \mathbb{R}^{N \times N}$	Adjacency matrix of graph
$\bar{\mathbf{A}} \in \mathbb{R}^{N \times N}$	Adjacency matrix of graph with self loop
N	Number of nodes
d_i	Degree of node i
$\mathbf{D} \in \mathbb{R}^{N \times N}$	Degree matrix of graph
$\mathbf{D} \in \mathbb{R}^{N \times N}$	Degree matrix of graph with self loop
$\mathcal{N}(v_i)$	1-hop neighborhood of node i
$\bar{\mathcal{N}}(v_i)$	1-hop neighborhood of node i with self loop
$\mathbf{X} \in \mathbb{R}^{N \times F}$	Node feature matrix
F	Node feature dimensions
$\mathbf{x}_i$	Node feature of node i
y_i	Node label of node i
$H_{edge}(\mathcal{G})$	Global edge homophily score of graph
$H_{node}(v_i)$	Local node homophily score of node i
$H_{node}(\mathcal{G})$	Global node homophily score of graph
$\mathsf{MESSAGE}(\cdot)$	Message passing function
$\mathbf{m}_{ij}^l$	Message sent to node i from node j at layer l
$\mathsf{AGRREGATE}(\cdot)$	Aggregation function
$\mathbf{a}_i^l$	Aggregation output for node i at layer l
$\mathsf{UPDATE}(\cdot)$	Update function
$\mathbf{h}_i^l$	node embedding of node i at layer l
$\mathbf{H}^l \in \mathbb{R}^{N \times F'}$	node embedding matrix at layer l
F'	Node embedding dimensions
$\sigma(\cdot)$	Nonlinear activation function
$\mathbf{W}^l$	Learnable weights at layer l
$\hat{\mathcal{G}} = (\hat{\mathbf{A}}, \mathbf{X})$	Perturbed graph
$\hat{\mathbf{A}}$	Perturbed adjacency matrix
$\mathcal{V}_t$	Victim node set
Δ	Attack budget
$\mathcal{L}_{atk}$	Attack objective function
$\mathcal{L}_{train}$	Training objective function
$f_\theta(\cdot)$	GNN model

using the aggregated output and the old node embeddings:

$$\mathbf{h}_i^{l+1} = \mathsf{UPDATE}(\mathbf{a}_i^l, \mathbf{h}_i^l) \tag{2.6}$$

The update is a per-node operation.

Therefore, the node embedding update paradigm can be generalized as follows:

$$\mathbf{h}_i^{l+1} = \mathsf{UPDATE}(\mathsf{AGGREGATE}(\{\mathsf{MESSAGE}(\{\mathbf{h}_j^l | v_j \in \mathcal{N}(v_i)\}\}), \mathbf{h}_i^l) \tag{2.7}$$

Different GNNs instantiate this paradigm in different ways. For GCN [20], the node embedding update process for each node is as follows:

$$\mathbf{h}_i^{l+1} = \sigma\left(\sum_{v_j \in \bar{\mathcal{N}}(v_i)} \frac{1}{\sqrt{\bar{\mathbf{D}}_{ii}\bar{\mathbf{D}}_{jj}}} \mathbf{h}_j^l \mathbf{W}^l\right) \tag{2.8}$$

In (2.8), $\bar{\mathcal{N}}(v_i)$ is v_i's neighborhood with "self loop" where we consider v_i as its neighbor as well. Therefore, $\bar{D}_{ii} = d_i + 1$. $\sigma(\cdot)$ is the nonlinear activation function and $\mathbf{W}^l$ is the learnable weights at layer l. Adopting the adjacency matrix with self loop $\bar{\mathbf{A}}$, (2.8) has the equivalent matrix form:

$$\mathbf{H}^{l+1} = \sigma\left(\bar{\mathbf{D}}^{-\frac{1}{2}}\bar{\mathbf{A}}\mathbf{D}^{-\frac{1}{2}}\mathbf{H}^l\mathbf{W}^l\right) \tag{2.9}$$

2.3 Graph Adversarial Attacks

Similar to other machine learning methods, Graph Neural Networks are also susceptible to adversarial attacks [32, 36, 17]. What makes adversarial attacks on graphs unique and more challenging is that graph structure is discrete, and connecting nodes depend on each other. In this Thesis, we only discuss adversarial attacks on the graph structure, so by default, node features are not perturbed in any case. Thus, the node-level graph adversarial attacks can be formulated as the general form below.

Given a graph $\mathcal{G} = (\mathbf{A}, \mathbf{X})$ and a subset of nodes as victim nodes $\mathcal{V}_t \subseteq \mathcal{V}$, we use y_i' to denote the label of v_i, and y_i' can either be the ground truth label or the

predicted label. The goal of the attacker is to find a perturbed graph $\hat{\mathcal{G}} = (\hat{\mathbf{A}}, \mathbf{X})$ that minimizes the attack objective $\mathcal{L}_{atk}$:

$$\begin{aligned} &min\ \mathcal{L}_{atk}(f_\theta(\hat{\mathcal{G}})) = \sum_{v_i \in \mathcal{V}_t} \mathcal{L}_{atk}\left(f_{\theta^*}(\hat{\mathcal{G}})_i, y_i'\right) \\ &s.t., \theta^* = arg\ \min_{\theta} \mathcal{L}_{train}(f_\theta(\mathcal{G}')) \end{aligned} \tag{2.10}$$

where $\mathcal{L}_{atk}$ is the loss function for attack and can be set as $\mathcal{L}_{atk} = -\mathcal{L}_{train}$. $\mathcal{L}_{train}$ can be any training loss function such as cross entropy function. $\mathcal{G}'$ can either be $\mathcal{G}$ or $\hat{\mathcal{G}}$. Note that $\hat{\mathcal{G}}$ should be close to $\mathcal{G}$ since attackers usually constrain their behavior to be unnoticeable with a budget Δ, so that:

$$||\hat{\mathbf{A}} - \mathbf{A}||_0 \leq 2\Delta \tag{2.11}$$

where $||\cdot||_0$ is the $l0$ norm, and attack budget Δ is a small number or proportion of edges the attackers aim to perturb.

Based on different settings, existing graph adversarial attacks can be put into different categories:

- **Poisoning attack or evasion attack**: Based on "when to attack", graph adversarial attacks can be divided into poisoning attack and evasion attack. A poisoning attack happens before the applied GNN model is trained, so it tries to attack the GNN model by perturbing the training data. This is the case when $\mathcal{G}' = \hat{\mathcal{G}}$ in (2.10). An evasion attack, on the other hand, happens in the test phase. This means the applied GNN model is fixed, and the attacker is not able to change the applied GNN model's parameter. Evasion attack is performed when $\mathcal{G}' = \mathcal{G}$ in (2.10).

- **Untargeted global attack or targeted attack**: Based on "which set of nodes the attackers attack", graph adversarial attacks can be categorized into untargeted global attack or targeted local attack. When $\mathcal{V}_t = \mathcal{V}$ in (2.10), the attack is considered to be an untargeted global attack, and it aims to make the trained model have bad overall performance. When $\mathcal{V}_t \subsetneqq \mathcal{V}$ in (2.10), a

targeted attack is performed, and the attacker aims to let the applied GNN model misclassify a limited number of test nodes.

- **White-box attack, gray-box attack, or black-box attack**: Based on the attacker's knowledge of the victim model, graph adversarial attacks have three settings: white-box attack, gray-box attack, and black-box attack. If all the information about the victim model and training data is visible to the attacker (e.g, model parameters, clean adjacency matrix, node features, training node labels), the attack is the white-box attack. Black-box attack is the attack when the attacker does not have access to the applied GNN model's parameters or training node labels. If the attacker's knowledge lies within black-box attack and white-box attack, then it is considered to be the gray-box attack.

Based on the above taxonomy, some of the widely studied graph adversarial attacks are categorized in Table 2.2. PGD attack or Min-max attack [37] is a white-box, untargeted attack that can be applied to both poisoning and evasion attack settings, the core idea is the "projected gradient decent". It relaxes the adjacency matrix from discrete $\{0,1\}^{N\times N}$to $[0,1]^{N\times N}$ during the gradient-based optimization, and the resulting weighted change indicates the probability of manipulating an edge. To make sure that the attack budget holds in expectation $||\mathbb{E}[\hat{\mathbf{A}}] - \mathbf{A}||_0 \leq 2\Delta$. Metattack [45] is a gray-box untargeted attack that can be applied to a poisoning attack setting. It generates the poisoning edges or deletes existing edges based on the largest meta-gradient. DICE [45] is a gray-box untargeted attack that can be applied to a poisoning attack setting. It has the knowledge of all the node labels and randomly connects nodes with different labels and disconnects nodes with the same labels. Nettack [44] is a gray-box targeted attack that can be applied to both poisoning and evasion attack settings. It generates perturbations by preserving degree distribution and imposing constraints on feature co-occurrence. RL-S2V [6] is a black-box untargeted attack that can be applied to an evasion attack setting. It employs reinforcement learning to generate edge perturbations which result in the maximum attack utility.

Considering the fact that poisoning attack is more challenging for transductive

Table 2.2: Categorization of some existing attack methods

Attack methods	Evasion or poisoning	Targeted or untargated	Attacker's knowledge
PGD, Min-max [37]	Can be both	Untargated	White-box
Nettack [44]	Can be both	Targated	Gray-box
Metattack [45]	Poisoning	Untargated	Gray-box
DICE [45]	Poisoning	Untargated	Gray-box
RL-S2V [6]	Evasion	Untargated	Black-box

learning, untargeted attack provides more possibilities for the attacker, and white-box attack allows for optimal knowledge of the attacker, we focus on PGD attack [37] in this Thesis. But our method can be used to defend against poisoning attacks in general.

2.4 Graph Adversarial Defenses

Plenty of efforts have been devoted to improving GNNs' adversarial robustness to tackle the attackers' behavior and prevent potential harmful effects. Existing methods can be categorized into 3 classes. Namely, graph purification, GNN architecture modification, and adversarial training [25].

- **Graph purification**: Some representative methods for graph purification are GCN-SVD [9], GCN-Jaccard [35], and Pro-GNN [18]. GCN-SVD [9] vaccinates GNNs by preprocessing the suspicious graph's adjacency matrix using the singular-value-decomposition-based low-rank approximation. This method is effective when the graph is heavily attacked, but it is harmful when facing a slightly perturbed graph since it will drop many useful edges and thus sacrifice the GNN's performance. Unlike GCN-SVD [9], which utilizes the low-rank property of the clean graph's adjacency matrix, GCN-Jaccard [35] prunes the graph based on the neighboring nodes' Jaccard similarity score. When two connected nodes are dissimilar based on the threshold, GCN-Jaccard will drop the edge between them. Though it leverages the information from node features, the potential information from graph topology is ignored. To exploit information from

graph structure and node features at the same time, based on the sparse and low-rank assumption for the adjacency matrix and feature smoothness assumption, Pro-GNN [18] adds regularizations to the objective function, aiming to learn the graph structure and robust node representations jointly. However, it's hard to tune the hyperparameters to regularize the objective function of Pro-GNN, and it has quadratic space and time complexity, which is not scalable for large graphs.

- **GNN architecture modification**: Some methods also specify the vulnerabilities in GNN architecture and modify the architecture accordingly. GNN-Guard [39] learns to assign higher weights to edges connecting similar nodes while pruning edges between unrelated nodes in the message-passing stage to mitigate potentially harmful effects of adversarial edges. RGCN [41] tries to absorb the effects of adversarial attacks in the aggregation stage by modeling latent node representations as gaussian distributions. It also applies attention mechanism to penalize nodes with high neighborhood feature variance in the message-passing stage.

- **Adversarial training**: Adversarial training-related methods generally model the defense as a min-max optimization problem, which inserts adversarial edges during training while minimizing the task-specific training loss [37, 7, 16]. However, this requires knowing the clean graph topology, or it will reinforce the poisoning attack if only the perturbed graph topology is observed, making it an impractical defense method.

The categorization of the above defense methods is summarized in Table 2.3.

Table 2.3: Categorization of some existing defense methods

Category	Defense methods
Graph purification	GCN-SVD [9], GCN-Jaccard [35], Pro-GNN [18]
GNN architecture modification	GNNGuard [39], RGCN [41]
Adversarial training	Xu et al. [37], Deng et al. [7], Jin et al. [16]

Chapter 3

Proposed Method

In this chapter, we introduce the proposed GalNN model in detail. In section 3.1, we will cover the adaptive aggregator selection. Then in section 3.2, we will explain the long-range dependency modeling. Followed by section 3.3, in which we will elaborate on how to conduct robust information fusion. Each section is organized with the motivation and design of the corresponding module in GalNN.

3.1 Adaptive Aggregator Selection

3.1.1 Motivation of Adaptive Aggregator Selection

The widely adopted weighted average aggregation can make GNNs vulnerable to graph structure adversarial attacks. Due to the popularity of GCN, most white-box and gray-box attackers tend to hijack GCN as the victim model to generate perturbations since this is the most cost-effective option to transfer attacks to other GNN models. Therefore, it's crucial to understand GCN's vulnerabilities to design countermeasures to improve the robustness of GNNs. Recall GCN's node embedding update process in (2.8):

$$\mathbf{h}_i^{l+1} = \sigma\left(\sum_{v_j \in \bar{\mathcal{N}}(v_i)} \frac{1}{\sqrt{\bar{\mathbf{D}}_{ii}\bar{\mathbf{D}}_{jj}}} \mathbf{h}_j^l \mathbf{W}^l\right) \tag{3.1}$$

It's obvious that the new centering node embedding is learned based on the weighted average of its neighboring nodes' old node embeddings. The weight $\frac{1}{\sqrt{\bar{\mathbf{D}}_{ii}\bar{\mathbf{D}}_{jj}}}$ is specified by the centering node's degree $\bar{\mathbf{D}}_{ii}$ and the neighboring node's degree $\bar{\mathbf{D}}_{jj}$. If $\bar{\mathbf{D}}_{jj}$ is large, centering node v_i will be less affected by v_j. However, if $\bar{\mathbf{D}}_{jj}$ is small, centering node v_i will be largely affected by v_j, indicating that sparse neighborhood is more likely to be compromised by an outlier. This has been verified in the empirical studies that attackers tend to manipulate edges among low-degree neighborhoods on homophily graphs [4]. What's more, every neighboring node has an influence on the output of the $mean(\cdot)$ aggregator for the centering node, indicating that even only one extreme outlier in a neighborhood would contribute to a harmful aggregation output for v_i. This harmful effect in the adversarial setting is illustrated in Figure 3-1. After the attacker connects node 3 with node 5, the output of the $mean(\cdot)$ aggregator is compromised since the output deviates from the original output by a large distance in the embedding space. This harmful effect can be more significant when attackers have a large attack budget. This is because attackers tend to conduct heterophilic attacks on homophily graphs, adding more edges connecting nodes with different labels or deleting edges between nodes with the same labels, where more outliers for the original neighborhoods are incorporated [42].

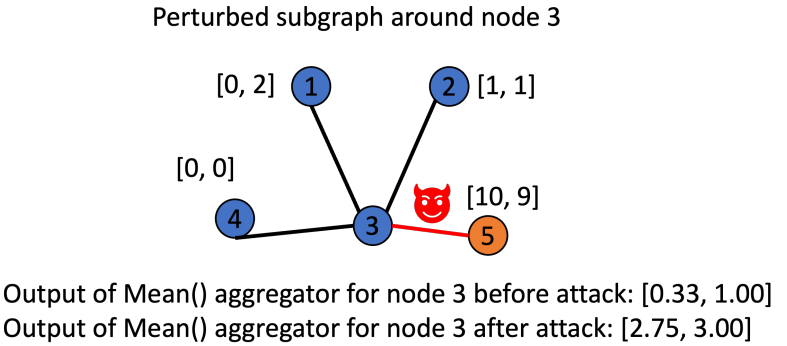

Figure 3-1: Illustration of the potential harmful effects of the $mean(\cdot)$ aggregator under attack. After the attacker connects node 3 with node 5, the output of the $mean(\cdot)$ aggregator is compromised.

Our proposed adaptive aggregator selection aims to protect sparse neighborhoods and mitigate the potentially harmful effects of the mean aggregator under attack on

homophily graphs.

3.1.2 Design of Adaptive Aggregator Selection

Adaptive aggregator selection focuses on the aggregation stage of GNN and has 2 phases: neighborhood profile learning and aggregator weight assigning. Each neighborhood profile is a vector, and each element in the vector serves as a weight for a corresponding aggregator. The sum of all the elements in a neighborhood profile equals to 1:

$$\mathbf{p}_i \doteq [w_1, w_2, ..., w_g] \in \mathbb{R}^{1\times G} \tag{3.2}$$

$$sum(\mathbf{p}_i) = 1 \tag{3.3}$$

where $\mathbf{p}_i$ is the profile for v_i. G is the number of aggregators used in the aggregator selection, and w_g is the learned weight that will be assigned to aggregator g.

Ideally, a desirable neighborhood profile should contain information from local graph topology and neighboring node features that can help differentiate perturbed neighborhoods from clean neighborhoods so that different weights can be assigned to different aggregators to absorb harmful effects from attackers. According to former studies, attackers change the degree distribution of sparse neighborhoods [4] and leveraging neighborhood node feature variance can help improve the robustness of GNNs [41]. Therefore, we learn each neighborhood's profile using MLP based on the centering node's degree and neighborhood node embedding variance:

$$\mathbf{p}_i = softmax(MLP(concat(d_i, var(\{\mathbf{h}_j | v_j \in \bar{\mathcal{N}}(v_i)\})))) \tag{3.4}$$

where $concat(\cdot)$ is the concatenation operation, $MLP(\cdot)$ is the MLP model, and $softmax(\cdot)$ is the softmax function.

We prepare 4 independent aggregators for aggregator selection, which are the $mean(\cdot)$ aggregator, $max(\cdot)$ aggregator, $min(\cdot)$ aggregator, and $median(\cdot)$ aggregator. Given a message matrix $\mathbf{M}_i \in \mathbb{R}^{|\bar{\mathcal{N}}(v_i)|\times F'}$ of v_i gained from the message passing stage, the aggregation outputs of each aggregator are as follows:

- **Mean aggregator**: Mean aggregator $mean(\cdot)$ calculates the arithmetic mean of $\mathbf{M}_i$ along each embedding dimension, so the output of it can be denoted as:

$$\mathbf{a}_i^{mean} = mean(\mathbf{M}_i) \in \mathbb{R}^{1\times F'} \tag{3.5}$$

- **Max aggregator**: Max aggregator $max(\cdot)$ calculates the maximum value of $\mathbf{M}_i$ along each embedding dimension, so the output of it can be denoted as:

$$\mathbf{a}_i^{max} = max(\mathbf{M}_i) \in \mathbb{R}^{1\times F'} \tag{3.6}$$

- **Min aggregator**: Min aggregator $min(\cdot)$ calculates the minimum value of $\mathbf{M}_i$ along each embedding dimension, so the output of it can be denoted as:

$$\mathbf{a}_i^{min} = min(\mathbf{M}_i) \in \mathbb{R}^{1\times F'} \tag{3.7}$$

- **Median aggregator**: Median aggregator $median(\cdot)$ calculates the median of $\mathbf{M}_i$ along each embedding dimension, so the output of it can be denoted as:

$$\mathbf{a}_i^{median} = median(\mathbf{M}_i) \in \mathbb{R}^{1\times F'} \tag{3.8}$$

Therefore, for each node v_i, the final aggregation output is:

$$\mathbf{a}_i = \mathbf{p}_i \begin{bmatrix} \mathbf{a}_i^{mean} \\ \mathbf{a}_i^{max} \\ \mathbf{a}_i^{min} \\ \mathbf{a}_i^{median} \end{bmatrix} \tag{3.9}$$

Thus, the final aggregated output is a weighted average of the 4 outputs from each aggregator. By doing so, we can achieve adaptive aggregator selection for node embedding learning.

3.2 Long-range Dependency Modeling

3.2.1 Motivation of Long-range Dependency Modeling

The observed graph topology is less trustworthy or less useful. In the real-world scenario, given a graph $\mathcal{G}' = (\mathbf{A}', \mathbf{X})$, we have no idea whether the observed graph is the original clean graph: $\mathcal{G}' = \mathcal{G}$, or in the worst case the observed graph is already under attack: $\mathcal{G}' = \hat{\mathcal{G}}$. Therefore, we may not trust the observed graph topology and apply the GNN model to it directly. Besides, even if the observed graph is the original clean graph: $\mathcal{G}' = \mathcal{G}$, GNN's performance may be hindered by the "homophily assumption" since not all the original graphs are homophily graphs [23, 22]. Various methods have been proposed to improve the performance of GNNs on heterophily graphs, and many of them focus on how to make GNNs use a more homophilic graph to learn node embeddings [40].

Ignoring the observed graph topology provides the best robustness guarantee. Though in most cases, the observed graph topology provides a satisfying regularization for node embedding learning, it can also be manipulated by the attackers to achieve their goals, making the observed graph topology a bad regularization for the original downstream tasks. Methods do not use the observed graph topology like MLP ignore the observed graph topology and only use the node features to learn the node embeddings. Hence, methods similar to MLP are immune to any adversarial graph structure attacks.

The most informative nodes for a given node might not be the closest neighbors; instead, they might be far away from the given node based on the observed graph topology. Theoretically, although the receptive field of GNNs can be enlarged by stacking GNN layers which potentially can help capture informative nodes far away from the centering node, stacking GNN layers can induce the over-smoothing issue of GNN [26]. They may also include more noisy nodes in the receptive field as well.

Inspired by the insights above, long-range dependency modeling aims to construct an auxiliary graph that can be more homophilic and learns a set of robust node

embeddings based on the node features.

3.2.2 Design of Long-range Dependency Modeling

The core idea of long-range dependency modeling is borrowed from semi-supervised learning on Riemannian manifolds, with the manifold assumption that data points on the same low-dimensional manifold should have the same label [33, 2]. We learn a set of node embeddings based on node features and project them to a low-dimensional underlying manifold, and then use the relative locality of node embeddings to calibrate the node embeddings and iteratively repeat the whole process until the downstream classifier indicates the convergence. To guarantee the adversarial robustness of long-range dependency modeling, we use MLP and 1d convolution to perform node embedding learning, manifold locality projection, and simulated graph convolution to achieve the long-range dependency modeling.

There are 4 stages in long-range dependency modeling, which are warming-up node embedding learning, 1-d manifold locality projection, simulated graph convolution, and embedding calibration.

- **Warming-up node embedding learning**: In this stage, we use an MLP to learn the warming-up node embeddings $\mathbf{Z}$ for the upcoming stages:

$$\mathbf{Z}^* = MLP(\mathbf{X}) \in \mathbb{R}^{N \times F'} \tag{3.10}$$

- **1-d manifold locality projection**: In this stage, we utilize the warming-up node embeddings to get each node's locality on the manifold. Here we choose the simplest manifold, 1-d manifold along the number axis, to facilitate the clustering process. The first step for 1-d manifold locality projection is embedding projection, where we use an MLP to project each warming-up node embedding to a scalar:

$$\mathbf{S} = MLP(\mathbf{Z}^*) \in \mathbb{R}^{N \times 1} \tag{3.11}$$

  The second step is to cluster the nodes based on the value of scalars. For

manifolds above 2-d, this usually requires calculating the pair-wise distance, selecting top-k least distant nodes, and considering them on the same manifold. However, for a 1-d manifold, clustering can be achieved by sorting the scalars since nodes with similar scalars would be clustered automatically and treat the nearest k nodes as the centering node's neighbors.

- **Simulated graph convolution**: Having the relocated warming-up node embeddings from 1-d manifold locality projection, we can simulate the graph convolution with a sliding 1-d convolution. This is because, after 1-d manifold locality projection, we actually get a knn graph along the 1-d manifold where closer nodes share similar information and may come from the same class. Based on this characteristic, 1-d convolution can be used to extract common patterns, and we call it simulated graph convolution and get another set of node embeddings $\mathbf{Z}'$:

$$\mathbf{Z}' = 1DConv(\mathbf{Z}^*_{clustered}) \in \mathbb{R}^{N \times F'} \tag{3.12}$$

  where $\mathbf{Z}^*_{clustered}$ denotes the clustered warm-up embeddings based on the value of corresponding scalars.

- **embedding calibration**: After getting $\mathbf{Z}'$ we can conduct embedding calibration on the warming-up node embeddings and get the final node embeddings for downstream tasks by:

$$\mathbf{Z} = concat(\mathbf{Z}^*, \mathbf{S}^T\mathbf{Z}') \in \mathbb{R}^{N \times 2F'} \tag{3.13}$$

After long-range dependency modeling, we will have an auxiliary k-nn graph $\mathcal{G}^*$ and a set of robust node embedding $\mathbf{Z}$ that is ready for robust information fusion.

3.3 Robust Information Fusion

3.3.1 Motivation of Robust Information Fusion

Fusing the information from aggregator selection and long-range dependency modeling may help to take advantage of the strength of both modules and at the same time avoid their weaknesses of them. Based on the homophily assumption, adaptive aggregator selection can handle poisoning attacks on homophily graphs by effectively capturing the local topology dependency. However, it may not fit into the heterophily pattern well. On the other hand, long-range dependency modeling ignores graph topology and captures node feature dependency well, and thus can handle heterophily patterns well. Still, it does not leverage the informative local topology under the homophily setting, so that the performance may be less satisfying on homophily graphs.

3.3.2 Design of Robust Information Fusion

According to the inputs and outputs of the adaptive aggregator selection and long-range dependency modeling, we have 3 schemes to conduct robust information fusion, which are neighborhood fusion, embedding fusion, and neighborhood plus embedding fusion.

- **Neighborhood fusion**: Given an observed graph $\mathcal{G}' = (\mathbf{A}', \mathbf{X})$, long-range dependency modeling can offer an auxiliary k-nn graph $\mathcal{G}^* = (\mathbf{A}^*, \mathbf{X})$. Neighborhood fusion combines these two graphs:

$$\mathcal{G}_{new} = \mathcal{G}' + \mathcal{G}^* = (\mathbf{A}' + \mathbf{A}^*, \mathbf{X}) \tag{3.14}$$

  and then make the new graph $\mathcal{G}_{new}$ as the input for adaptive aggregator selection to learn the node embeddings for the downstream tasks.

- **Embedding fusion**: Using the observed graph $\mathcal{G}' = (\mathbf{A}', \mathbf{X})$ as the input for both modules, long-range dependency modeling can provide a set of node

embeddings $\mathbf{Z}$, and adaptive aggregator selection can provide another set of node embeddings $\mathbf{H}$, we can initialize two learnable weights $w_1 + w_2 = 1$ to fuse these two sets of node embeddings and get the final node embeddings for the downstream tasks:

$$\mathbf{H}_{new} = w_1\mathbf{H} + w_2\mathbf{Z} \tag{3.15}$$

where a bigger w_1 indicates that local topology is more important than information from long-range dependencies, and vice versa.

- **Neighborhood fusion plus embedding fusion**: For this fusion scheme, the input of long-range dependency modeling is the observed graph $\mathcal{G}'$, while the input for adaptive aggregator selection is $\mathcal{G}_{new}$ defined in the neighborhood fusion. After getting the two sets of node embeddings $\mathbf{Z}$ and $\mathbf{H}$, embedding fusion is performed in the same manner as above.

GalNN's variants are illustrated in Figure 3-2. If the observed graph is only passed to the long-range dependency modeling module, GalNN is instantiated as GalNN-LDM. If the observed graph is only passed to the adaptive aggregator selection module, GalNN is instantiated as GalNN-AAS. If the observed graph is passed to the adaptive aggregator selection module with the neighborhood fusion module activated, GalNN is instantiated as GalNN-NF. If the observed graph is passed to both the long-range dependency modeling module and the adaptive aggregator selection module without activating the neighborhood fusion module, GalNN is instantiated as GalNN-EF. If all the modules are activated, GalNN is instantiated as GalNN-both.

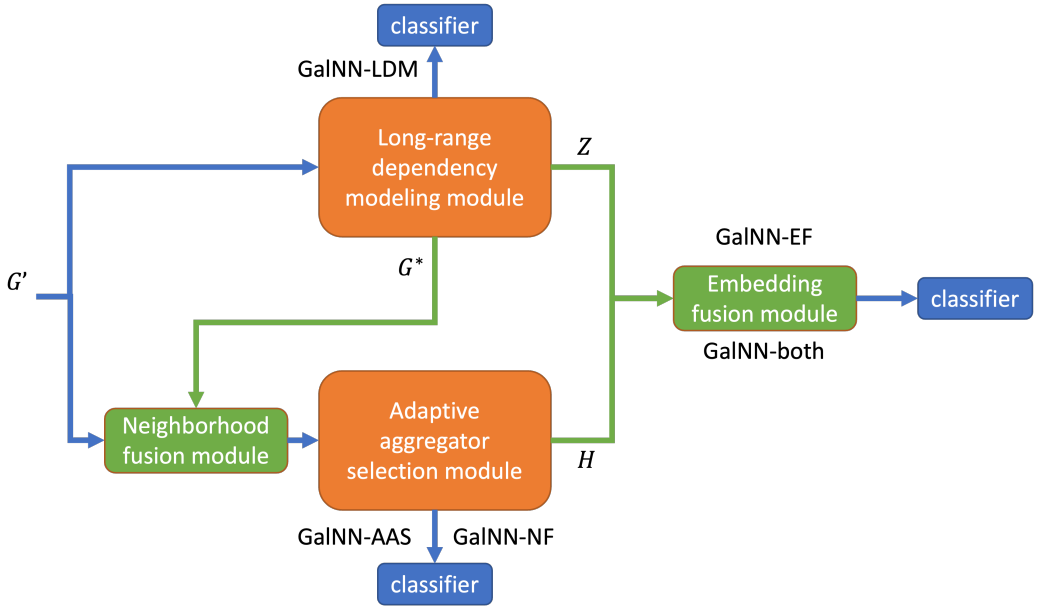

Figure 3-2: Illustration of GalNN's variants. If the observed graph is only passed to the long-range dependency modeling module, GalNN is instantiated as GalNN-LDM. If the observed graph is only passed to the adaptive aggregator selection module, GalNN is instantiated as GalNN-AAS. If the observed graph is passed to the adaptive aggregator selection module with neighborhood fusion module activated, GalNN is instantiated as GalNN-NF. If the observed graph is passed to both the long-range dependency modeling module and the adaptive aggregator selection module without activating the neighborhood fusion module, GalNN is instantiated as GalNN-EF. If all the modules are activated, GalNN is instantiated as GalNN-both.

Chapter 4

Experiments

In this section, we investigate the effectiveness and robustness against the poisoning attack of our proposed GalNN on node classification tasks. We conduct experiments on 9 real-world datasets across the whole homophily score spectrum. We compare GalNN's variants against a set of state-of-the-art baselines to illustrate its advantage. In addition, we also did an ablation analysis to study the importance of each component in GalNN.

4.1 Experimental Setup

Our code for GalNN is implemented with PyTorch [27], PyTorch Geometric [10], and DeepRobust [17]. We use PGD attack in DeepRobust library with the default setting and hijack GCN model to launch poisoning attacks on graph topology.

4.1.1 Datasets

All the datasets we use are summarized in Table 4.1. These datasets are obtained from PyTorch Geometric library. For homophily datasets Cora, Citeseer, and Pubmed, we randomly select 20 nodes from each class as training nodes, 20 nodes from each class as validation nodes and select 1000 nodes from the rest of the nodes as test nodes. For the rest heterophily datasets, we randomly split the train/validation/test set

with ratio 48%/32%/20%, which is the same split ratio as what's been adopted in Geom-GCN [28].

Table 4.1: Statistics of the node classification datasets

Dataset	Homophily score	Nodes	Edges	Features	Classes
Cora	0.81	2708	5429	1433	7
Citeseer	0.74	3327	4732	3703	6
Pubmed	0.80	19717	44338	500	3
Cornell	0.30	183	295	1703	5
Texas	0.11	183	309	1703	5
Wisconsin	0.21	251	499	1703	5
Chameleon	0.23	2277	36101	2325	4
Squirrel	0.22	5201	198353	2089	5
Actor	0.22	7600	26659	932	5

4.1.2 Baselines

We compare our model with five baselines containing defense methods from the graph purification category and the GNN architecture modification category. Note that we don't compare GalNN's performance with adversarial training since adversarial training requires the knowledge of clean graphs, which is not practical in real-world scenarios. We adopt the same hyperparameter setting as the authors of those baseline methods.

- **MLP**: The multi-layer perceptron model for semi-supervised learning, which only takes the node features as input to learn the node embeddings. Note that MLP is immune to graph structure attacks.

- **GCN**: Standard graph convolutional network proposed by Kipf and Welling. This is the most widely used GNNs in the real world, so attackers always hijack GCN to generate perturbations and compromise other GNN models.

- **GCN-SVD**: Graph purification defense method that only leverages the information from the adjacency matrix. It tries to recover and approximate the clean graph using low-rank approximation on the adjacency matrix then use the approximated adjacency matrix in the training process.

- **GCN-Jaccard**: Graph purification defense method that only leverages the information from the node features. It tries to identify and prune the edges based on pair-wise Jaccard similarity, therefore, edge connecting nodes with low Jaccard similarity score will be dropped before the training of GCN.

- **RGCN**: GNN architecture modification defense method that tries to absorb the effects of adversarial attacks in the aggregation stage by modeling latent node representations as gaussian distributions. It also applies attention mechanism to penalize nodes with high neighborhood feature variance in the message-passing stage.

4.1.3 Implementation Details

For GalNN's variants that activate the adaptive aggregator selection module, the number of hidden units is set as 16, and the MLP for learning the neighborhood is instantiated as one linear layer. For GalNN's variants that activate the long-range dependency modeling, all the MLPs involved are instantiated as two-layer MLPs, and the embedding size of warming-up node embeddings is 16. The window size of 1-d convolution used in simulated graph convolution is 5, so each node on the auxiliary graph will have 5 neighbors. For GalNN's variants that activate the embedding fusion module, we initialize the weight for $\mathbf{H}$ as 0.8 and the weight for $\mathbf{Z}$ as 0.2.

During training, we apply Adam optimizer [19] with a learning rate of 1e-2 and weight decay 5e-4. We train the model for at most 3000 epochs with early stopping based on validation loss. We use PGD attack in DeepRobust library with the default setting and hijack GCN model to launch poisoning attacks on graph topology.

4.2 Performance on Clean Graphs

To investigate the performance of GalNN on clean graphs, we report the node classification mean accuracy over 5 random seeds on all the 9 datasets and compare the performance with all the 5 baseline methods. The performance is reported in Ta-

ble 4.2, and we can see that GalNN's variants generally perform better compared to the baselines. For 8 out of 9 datasets, GalNN's variants can achieve the state of the art classification accuracy.

Table 4.2: Node classification accuracy on clean graphs. The best performance on each dataset is in bold font, and the second-best performance is underlined.

Methods \ Datasets	Cora	Citeseer	Pubmed	Cornell	Texas	Wisconsin	Squirrel	Chameleon	Actor
MLP	0.5440	0.4940	0.4890	**0.7568**	<u>0.7838</u>	<u>0.7647</u>	0.3324	0.4561	0.3336
GCN	0.7930	0.6560	<u>0.7870</u>	0.4054	0.6757	0.5098	0.4736	<u>0.6535</u>	0.3066
GCN-SVD	0.6930	0.5430	0.7810	0.6216	0.6757	0.6275	0.3506	0.4671	0.2993
RGCN	0.8030	0.6320	OOM^1	0.3514	0.6216	0.5394	0.3833	0.5943	0.3033
GCN-Jaccard	0.7600	0.6640	**0.7900**	0.3784	0.5405	0.5098	$divbyzero^2$	$divbyzero^2$	0.3263
GalNN-AAS	0.7990	**0.6930**	0.7830	0.4324	0.7027	0.5294	<u>0.5274</u>	**0.6557**	0.3033
GalNN-LDM	0.4490	0.4610	0.6490	<u>0.7027</u>	<u>0.7838</u>	**0.7843**	0.3468	0.4956	**0.3750**
GalNN-EF	**0.8110**	0.6840	0.7720	0.4865	**0.8378**	0.7451	0.5264	0.6228	<u>0.3632</u>
GalNN-NF	<u>0.8060</u>	0.6760	**0.7900**	0.4054	0.6757	0.4706	**0.5408**	0.6491	0.2803
GalNN-both	<u>0.8060</u>	<u>0.6850</u>	0.7670	0.6216	0.6757	**0.7843**	0.5082	0.5899	**0.3750**

1: OOM denotes out of memory.
2: The error of division by zero.

4.3 Performance on Graphs under Poisoning Attack

We launch PGD attack with 20% of edges attack budget on 4 representative datasets: Cora (large homophily graph), Wisconsin (small heterophily graph), Chameleon (large heterophily graph), and Actor (large heterophily graph but different from Chameleon and Squirrel). The performance of GalNN's variants and baselines is reported in Table 4.3. We can see that for all the datasets, GalNN's variants have overall significantly better performance, especially for GalNN-EF, which can consistently achieve top-2 performance. Compared to the performance on clean graphs, GalNN's variants achieve the most negligible performance degradation, which means GalNN is the most robust defense model in the benchmarking. We also launch DICE attack with 20% of edges attack budget on the above 4 datasets. The performance of GalNN's variants and baselines is reported in Table 4.4. Compared to baselines, GalNN's variants can achieve top-2 performance in most cases. The results listed in the below two tables indicate GalNN's effectiveness of defending against both supervised poisoning attacker (PGD) and unsupervised poisoning attacker (DICE).

Table 4.3: Node classification accuracy on perturbed graphs attacked by PGD attack with the attack budget of 20% of edges. The best performance on each dataset is in bold font, and the second-best performance is underlined.

Methods \ Datasets	Cora	Wisconsin	Chameleon	Actor
MLP	0.5440	<u>0.7647</u>	0.4561	0.3336
GCN	0.5810	0.5098	0.3794	0.2500
GCN-SVD	0.6910	0.7255	0.4123	0.2961
RGCN	0.6950	0.4314	0.3618	0.2579
GCN-Jaccard	0.6900	0.5686	*divbyzero*[1]	0.2697
GalNN-AAS	**0.7850**	0.5294	0.3728	0.2546
GalNN-LDM	0.4490	**0.7843**	<u>0.4956</u>	**0.3750**
GalNN-EF	<u>0.7590</u>	<u>0.7647</u>	**0.5197**	<u>0.3461</u>
GalNN-NF	0.7490	0.4902	0.3750	0.2645
GalNN-both	0.7530	0.7451	0.4737	0.3428

1: The error of division by zero.

Table 4.4: Node classification accuracy on perturbed graphs attacked by DICE attack with the attack budget of 20% of edges. The best performance on each dataset is in bold font, and the second-best performance is underlined.

Methods \ Datasets	Cora	Wisconsin	Chameleon	Actor
MLP	0.5440	<u>0.7843</u>	0.4561	0.3336
GCN	0.7570	0.5098	**0.6053**	0.2803
GCN-SVD	0.6280	0.6471	0.4627	0.2993
RGCN	0.7450	0.5686	0.5636	0.2645
GCN-Jaccard	0.7340	0.5686	*divbyzero*[1]	0.3007
GalNN-AAS	0.7550	0.5490	0.5921	0.2829
GalNN-LDM	0.5060	**0.8039**	0.4978	0.3375
GalNN-EF	<u>0.7650</u>	0.7451	<u>0.5943</u>	<u>0.3507</u>
GalNN-NF	0.7540	0.5294	0.5833	0.2783
GalNN-both	**0.7670**	0.7451	0.5570	**0.3526**

1: The error of division by zero.

Due to GalNN-EF's solid performance listed above, we also performed the robustness performance benchmarking with the increase of attack budget (from 0 to 25%) under PGD attack on the 4 datasets from the whole homophily spectrum, the results are shown in Figure 4-1, Figure 4-2, Figure 4-3, and Figure 4-4. We can observe that GalNN-EF outperforms other baselines under any attack budget. On the datasets where MLP performs the best, GalNN-EF has relatively the same performance. Therefore, GalNN-EF provides a robust node embedding learning strategy accross the whole homophily spectrum.

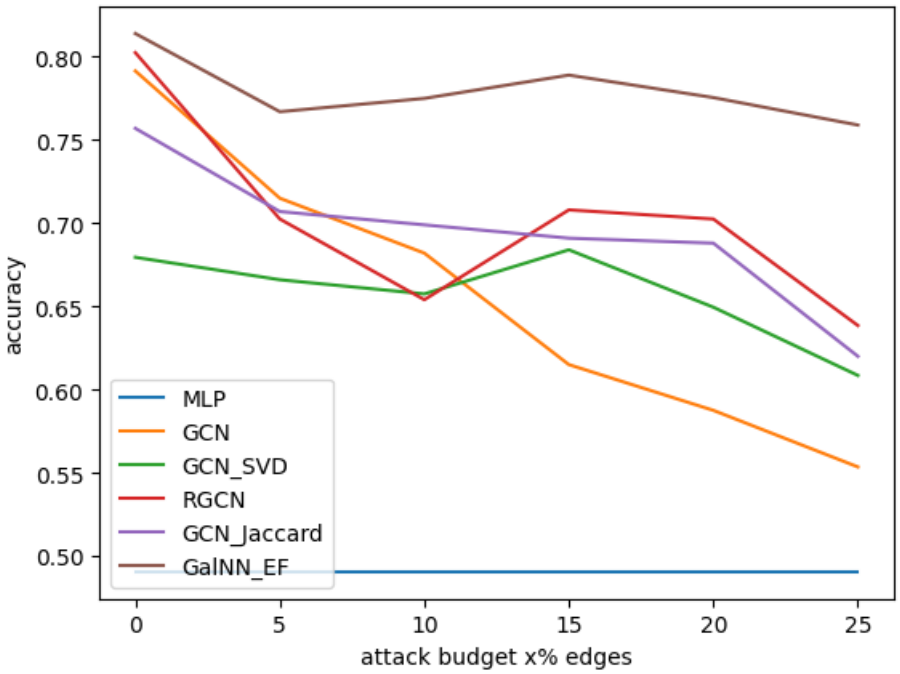

Figure 4-1: Robustness benchmarking under PGD attack on cora with the increase of attack budget.

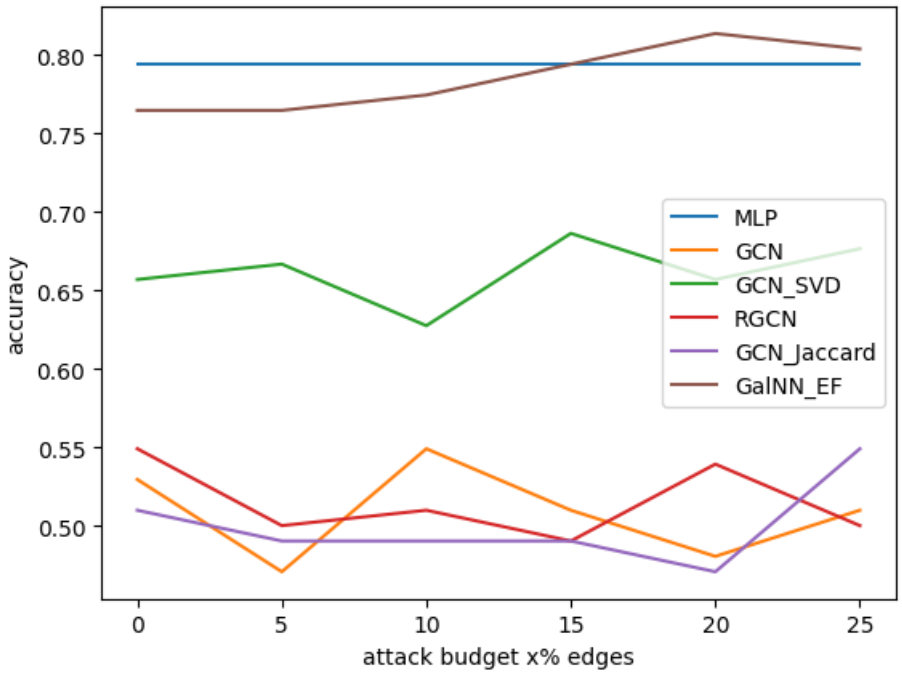

Figure 4-2: Robustness benchmarking under PGD attack on wisconsin with the increase of attack budget.

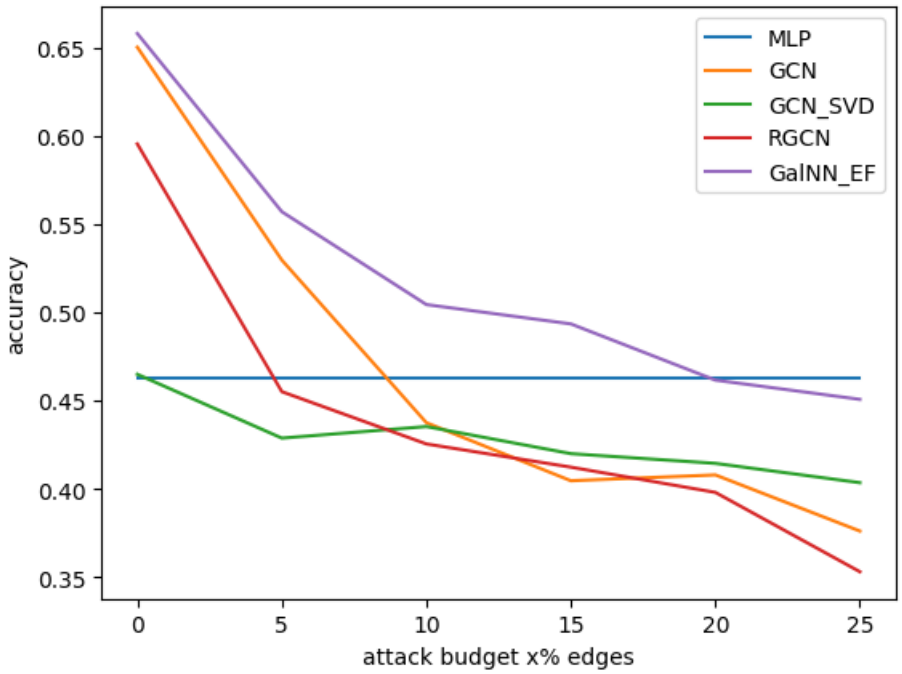

Figure 4-3: Robustness benchmarking under PGD attack on chameleon with the increase of attack budget.

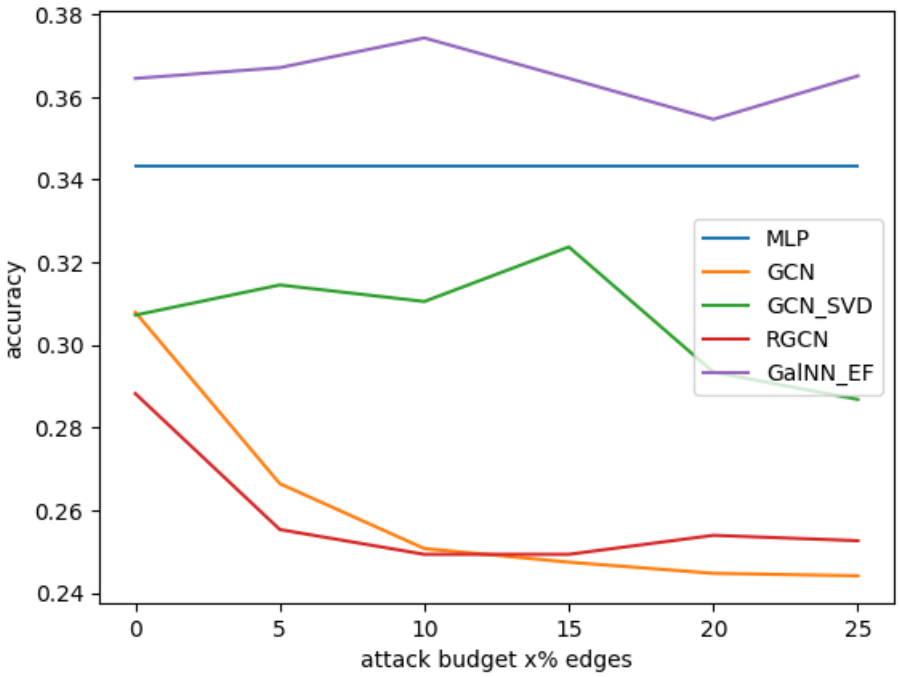

Figure 4-4: Robustness benchmarking under PGD attack on actor with the increase of attack budget.

4.4 Ablation Analysis

In this section, we try to open up the black box and study the contribution of each component to the superior performance of GalNN.

4.4.1 Contribution of Adaptive Aggregator Selection

From Table 4.2 and Table 4.3 we can observe that GalNN-AAS works extremely well on homophily graph attacked by a strong attacker PGD, with the performance degradation less than 2% on Cora even if the graph is heavily perturbed with 20% of edges. This supremacy comes from adaptive aggregator selection since the learned neighborhood profiles can help identify perturbed neighborhoods and assign different weights accordingly. We use t-SNE to project the learned neighborhood profiles to 2D space. Below in Figure 4-5 is the visualization of the clustering of neighborhood profiles on Cora perturbed by PGD attack with 20% of edges.

Figure 4-5: Visualization of the clustering of neighborhood profiles on Cora perturbed by PGD attack with 20% of edges. Each point denotes a neighborhood.

4.4.2 Contribution of Long-range Dependency Modeling

From Table 4.2 and Table 4.3 we can observe that GalNN-LDM has good performance on many heterophily graphs. This shows the effectiveness of long-range dependency modeling. By calculating homophily scores of the constructed $\mathcal{G}^*$ and comparing it with the original homophily scores, we draw the conclusion that long-range dependency helps to construct more homophilic graphs and therefore help with better node embedding learning on heterophily graphs. The homophily score comparison is summarized in Table 4.5.

Table 4.5: Homophily score comparison between the original heterophily graphs and constructed graph by long-range dependency modeling.

Graphs \ Datasets	Cornell	Texas	Wisconsin	Squirrel	Chameleon	Actor
original graph	0.30	0.11	0.21	0.22	0.23	0.22
constructed graph	0.68	0.72	0.65	0.32	0.42	0.31

4.4.3 Contribution of Embedding Fusion

As GalNN-EF performs well in defending both homophily graphs and heterophily graphs, we further investigate the weights GalNN-EF learns. The results in Table 4.6 indicate that GalNN-EF assigns more weight to embeddings from adaptive aggregator selection when the original graph is a homophily graph and more weight to long-range dependency modeling when the original graph is a heterophily graph.

Table 4.6: Weghts assigned to different node embeddings by GalNN-EF on perturbed graphs attacked by PGD attack with the attack budegt of 20% of edges.

Weights \ Datasets	Cora	Wisconsin	Chameleon	Actor
weight for AAS	0.8019	0.3757	0.4046	0.1717
weight for LDM	0.1981	0.6246	0.5954	0.8283

Chapter 5

Conclusion and Future Work

In this paper, we present GalNN, a robust Graph Neural Network against poisoning adversarial graph structure attacks. It integrates adaptive aggregator selection and long-range dependency modeling to improve the robustness of GNN across the whole homophily score spectrum. The adaptive aggregator selection module utilizes the observed graph topology to capture underlying local node dependencies. The learned neighborhood profiles for adaptive aggregator selection can help differentiate perturbed neighborhoods from clean ones and assign weights for different aggregators accordingly. Long-range dependency modeling provides robustness guarantees and helps construct more homophilic graphs to leverage when the original graph is heterophilic. Finally, we use robust information fusion with 3 options to take advantage of both modules. Extensive experiments on a wide range of real-world datasets show the effectiveness of GalNN.

Based on the limitations indicated by our experiment results, we treat the following directions as future work:

- **More in-depth understanding towards heterophily patterns**. Currently, most GNN's performance is upper-bounded by the homophily assumption. When it comes to heterophily, the only description is "a graph with low homophily score". However, unlike high homophily scores, similar low homophily scores may not indicate similar heterophily patterns. As is shown in Figure 2-1, graph

in (b) and graph in (c) have the same homophily score, but they have different heterophily patterns. This issue poses the difficulty for GNNs to generalize on heterophily graphs. Besides, in-depth understanding of heterophily patterns can also help understand the attacks on originally heterophily graphs and help design better defense in a direct way, instead of trying to transform a heterophily graph into a more homophilic one.

- **More flexible long-range dependency modeling methods based on observed graph**. Our long-range dependency modeling method disregards the observed graph topology. Doing so loses too much information about the original graph and thus harms the model's learning capacity. As attackers only have a limited budget to perform perturbations, we believe the most useful information still exists on the observed graph. Therefore, simply dropping the whole graph topology is sub-optimal.

- **Better aggregator ensemble in the adaptive aggregator selection**. We only prepared 4 aggregators (mean, max, min, and median) for adaptive aggregator selection. There is no theoretical guarantee that the ensemble of these 4 aggregators is the best option. Therefore, there may be better ensembles to explore.

Bibliography

[1] Naveed Akhtar and Ajmal Mian. Threat of adversarial attacks on deep learning in computer vision: A survey. *Ieee Access*, 6:14410–14430, 2018.

[2] Mikhail Belkin and Partha Niyogi. Semi-supervised learning on riemannian manifolds. *Machine learning*, 56(1):209–239, 2004.

[3] Anirban Chakraborty, Manaar Alam, Vishal Dey, Anupam Chattopadhyay, and Debdeep Mukhopadhyay. Adversarial attacks and defences: A survey. *arXiv preprint arXiv:1810.00069*, 2018.

[4] Liang Chen, Jintang Li, Qibiao Peng, Yang Liu, Zibin Zheng, and Carl Yang. Understanding structural vulnerability in graph convolutional networks. *arXiv preprint arXiv:2108.06280*, 2021.

[5] Edward Choi, Mohammad Taha Bahadori, Le Song, Walter F Stewart, and Jimeng Sun. Gram: graph-based attention model for healthcare representation learning. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 787–795, 2017.

[6] Hanjun Dai, Hui Li, Tian Tian, Xin Huang, Lin Wang, Jun Zhu, and Le Song. Adversarial attack on graph structured data. In *International conference on machine learning*, pages 1115–1124. PMLR, 2018.

[7] Zhijie Deng, Yinpeng Dong, and Jun Zhu. Batch virtual adversarial training for graph convolutional networks. *arXiv preprint arXiv:1902.09192*, 2019.

[8] Guimin Dong, Mingyue Tang, Zhiyuan Wang, Jiechao Gao, Sikun Guo, Lihua Cai, Robert Gutierrez, Bradford Campbell, Laura E Barnes, and Mehdi Boukhechba. Graph neural networks in iot: A survey. *arXiv preprint arXiv:2203.15935*, 2022.

[9] Negin Entezari, Saba A Al-Sayouri, Amirali Darvishzadeh, and Evangelos E Papalexakis. All you need is low (rank) defending against adversarial attacks on graphs. In *Proceedings of the 13th International Conference on Web Search and Data Mining*, pages 169–177, 2020.

[10] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*, 2019.

[11] Samuel G Finlayson, John D Bowers, Joichi Ito, Jonathan L Zittrain, Andrew L Beam, and Isaac S Kohane. Adversarial attacks on medical machine learning. *Science*, 363(6433):1287–1289, 2019.

[12] Victor Fung, Jiaxin Zhang, Eric Juarez, and Bobby G Sumpter. Benchmarking graph neural networks for materials chemistry. *npj Computational Materials*, 7(1):1–8, 2021.

[13] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

[14] William L Hamilton. Graph representation learning. *Synthesis Lectures on Artifical Intelligence and Machine Learning*, 14(3):1–159, 2020.

[15] Dasaem Jeong, Taegyun Kwon, Yoojin Kim, and Juhan Nam. Graph neural network for music score data and modeling expressive piano performance. In *International Conference on Machine Learning*, pages 3060–3070. PMLR, 2019.

[16] Hongwei Jin and Xinhua Zhang. Latent adversarial training of graph convolution networks. In *ICML workshop on learning and reasoning with graph-structured representations*, volume 2, 2019.

[17] Wei Jin, Yaxin Li, Han Xu, Yiqi Wang, Shuiwang Ji, Charu Aggarwal, and Jiliang Tang. Adversarial attacks and defenses on graphs: A review, a tool and empirical studies. *arXiv preprint arXiv:2003.00653*, 2003.

[18] Wei Jin, Yao Ma, Xiaorui Liu, Xianfeng Tang, Suhang Wang, and Jiliang Tang. Graph structure learning for robust graph neural networks. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 66–74, 2020.

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

[20] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[21] Zhiwei Liu, Yingtong Dou, Philip S Yu, Yutong Deng, and Hao Peng. Alleviating the inconsistency problem of applying graph neural network to fraud detection. In *Proceedings of the 43rd international ACM SIGIR conference on research and development in information retrieval*, pages 1569–1572, 2020.

[22] Sitao Luan, Chenqing Hua, Qincheng Lu, Jiaqi Zhu, Mingde Zhao, Shuyuan Zhang, Xiao-Wen Chang, and Doina Precup. Is heterophily a real nightmare for graph neural networks to do node classification? *arXiv preprint arXiv:2109.05641*, 2021.

[23] Yao Ma, Xiaorui Liu, Neil Shah, and Jiliang Tang. Is homophily a necessity for graph neural networks? *arXiv preprint arXiv:2106.06134*, 2021.

[24] Diego Marcheggiani and Ivan Titov. Encoding sentences with graph convolutional networks for semantic role labeling. *arXiv preprint arXiv:1703.04826*, 2017.

[25] Felix Mujkanovic, Simon Geisler, Stephan Günnemann, and Aleksandar Bojchevski. Are defenses for graph neural networks robust? *Advances in Neural Information Processing Systems 35 (NeurIPS 2022)*, 2022.

[26] Kenta Oono and Taiji Suzuki. Graph neural networks exponentially lose expressive power for node classification. *arXiv preprint arXiv:1905.10947*, 2019.

[27] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[28] Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. Geom-gcn: Geometric graph convolutional networks. *arXiv preprint arXiv:2002.05287*, 2020.

[29] Jiezhong Qiu, Jian Tang, Hao Ma, Yuxiao Dong, Kuansan Wang, and Jie Tang. Deepinf: Social influence prediction with deep learning. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 2110–2119, 2018.

[30] Lei Shi, Yifan Zhang, Jian Cheng, and Hanqing Lu. Skeleton-based action recognition with directed graph neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7912–7921, 2019.

[31] Jonathan Shlomi, Peter Battaglia, and Jean-Roch Vlimant. Graph neural networks in particle physics. *Machine Learning: Science and Technology*, 2(2):021001, 2020.

[32] Lichao Sun, Yingtong Dou, Carl Yang, Ji Wang, Philip S Yu, Lifang He, and Bo Li. Adversarial attack and defense on graph data: A survey. *arXiv preprint arXiv:1812.10528*, 2018.

[33] Jesper E Van Engelen and Holger H Hoos. A survey on semi-supervised learning. *Machine Learning*, 109(2):373–440, 2020.

[34] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *stat*, 1050:20, 2017.

[35] Huijun Wu, Chen Wang, Yuriy Tyshetskiy, Andrew Docherty, Kai Lu, and Liming Zhu. Adversarial examples on graph data: Deep insights into attack and defense. *arXiv preprint arXiv:1903.01610*, 2019.

[36] Han Xu, Yao Ma, Hao-Chen Liu, Debayan Deb, Hui Liu, Ji-Liang Tang, and Anil K Jain. Adversarial attacks and defenses in images, graphs and text: A review. *International Journal of Automation and Computing*, 17(2):151–178, 2020.

[37] Kaidi Xu, Hongge Chen, Sijia Liu, Pin-Yu Chen, Tsui-Wei Weng, Mingyi Hong, and Xue Lin. Topology attack and defense for graph neural networks: An optimization perspective. *arXiv preprint arXiv:1906.04214*, 2019.

[38] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 974–983, 2018.

[39] Xiang Zhang and Marinka Zitnik. Gnnguard: Defending graph neural networks against adversarial attacks. *Advances in neural information processing systems*, 33:9263–9275, 2020.

[40] Xin Zheng, Yixin Liu, Shirui Pan, Miao Zhang, Di Jin, and Philip S Yu. Graph neural networks for graphs with heterophily: A survey. *arXiv preprint arXiv:2202.07082*, 2022.

[41] Dingyuan Zhu, Ziwei Zhang, Peng Cui, and Wenwu Zhu. Robust graph convolutional networks against adversarial attacks. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 1399–1407, 2019.

[42] Jiong Zhu, Junchen Jin, Donald Loveland, Michael T Schaub, and Danai Koutra. How does heterophily impact the robustness of graph neural networks? theoretical connections and practical implications. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 2637–2647, 2022.

[43] Jiong Zhu, Yujun Yan, Lingxiao Zhao, Mark Heimann, Leman Akoglu, and Danai Koutra. Beyond homophily in graph neural networks: Current limitations and effective designs. *Advances in Neural Information Processing Systems*, 33:7793–7804, 2020.

[44] Daniel Zügner, Amir Akbarnejad, and Stephan Günnemann. Adversarial attacks on neural networks for graph data. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 2847–2856, 2018.

[45] Daniel Zügner and Stephan Günnemann. Adversarial attacks on graph neural networks via meta learning. *arXiv preprint arXiv:1902.08412*, 2019.