Graph Structure Learning for Robust Graph Neural Networks

Wei Jin
Michigan State University
jinwei2@msu.edu

Yao Ma
Michigan State University
mayao4@msu.edu

Xiaorui Liu
Michigan State University
xiaorui@msu.edu

Xianfeng Tang
The Pennsylvania State University
tangxianfeng@outlook.com

Suhang Wang
The Pennsylvania State University
szw494@psu.edu

Jiliang Tang
Michigan State University
tangjili@msu.edu

ABSTRACT

Graph Neural Networks (GNNs) are powerful tools in representation learning for graphs. However, recent studies show that GNNs are vulnerable to carefully-crafted perturbations, called adversarial attacks. Adversarial attacks can easily fool GNNs in making predictions for downstream tasks. The vulnerability to adversarial attacks has raised increasing concerns for applying GNNs in safety-critical applications. Therefore, developing robust algorithms to defend adversarial attacks is of great significance. A natural idea to defend adversarial attacks is to clean the perturbed graph. It is evident that real-world graphs share some intrinsic properties. For example, many real-world graphs are low-rank and sparse, and the features of two adjacent nodes tend to be similar. In fact, we find that adversarial attacks are likely to violate these graph properties. Therefore, in this paper, we explore these properties to defend adversarial attacks on graphs. In particular, we propose a general framework Pro-GNN, which can jointly learn a structural graph and a robust graph neural network model from the perturbed graph guided by these properties. Extensive experiments on real-world graphs demonstrate that the proposed framework achieves significantly better performance compared with the state-of-the-art defense methods, even when the graph is heavily perturbed. We release the implementation of Pro-GNN to our DeepRobust repository for adversarial attacks and defenses [1]. The specific experimental settings to reproduce our results can be found in https://github.com/ChandlerBang/Pro-GNN.

1 INTRODUCTION

Graphs are ubiquitous data structures in numerous domains, such as chemistry (molecules), finance (trading networks) and social media

---

[1] https://github.com/DSE-MSU/DeepRobust

(the Facebook friend network). With their prevalence, it is particularly important to learn effective representations of graphs and then apply them to solve downstream tasks. Recent years have witnessed great success from Graph Neural Networks (GNNs) [15, 18, 22, 30] in representation learning of graphs. GNNs follow a message-passing scheme [14], where the node embedding is obtained by aggregating and transforming the embeddings of its neighbors. Due to the good performance, GNNs have been applied to various analytical tasks including node classification [18], link prediction [19], and recommender systems [35].

Although promising results have been achieved, recent studies have shown that GNNs are vulnerable to adversarial attacks [10, 17, 32, 39, 40]. In other words, the performance of GNNs can greatly degrade under an unnoticeable perturbation in graphs. The lack of robustness of these models can lead to severe consequences for critical applications pertaining to the safety and privacy. For example, in credit card fraud detection, fraudsters can create several transactions with only a few high-credit users to disguise themselves, thus escaping from the detection based on GNNs. Hence, developing robust GNN models to resist adversarial attacks is of significant importance. Modifying graph data can perturb either node features or graph structures. However, given the complexity of structural information, the majority of existing adversarial attacks on graph data have focused on modifying graph structure especially adding/deleting/rewiring edges [34]. Thus, in this work, we aim to defend against the most common setting of adversarial attacks on graph data, i.e., poisoning adversarial attacks on graph structure. Under this setting, the graph structure has already been perturbed by modifying edges before training GNNs while node features are not changed.

One perspective to design an effective defense algorithm is to clean the perturbed graph such as removing the adversarial edges and restoring the deleted edges [28, 38]. The key challenge from this perspective is what criteria we should follow to clean the perturbed graph. It is well known that real-world graphs often share certain properties. First, many real-world clean graphs are low-rank and sparse [37]. For instance, in a social network, most individuals are connected with only a small number of neighbors and there are only a few factors influencing the connections among users [13, 37]. Second, connected nodes in a clean graph are likely to share similar features or attributes (or feature smoothness) [25]. For example, in a citation network, two connected publications often share similar topics [18]. Figure 1 demonstrates these properties of clean and poisoned graphs. Specifically, we apply the state-of-the-art graph poisoning attack, *metattack* [40], to perturb the graph data and visualize the graph properties before and after *mettack*. As shown
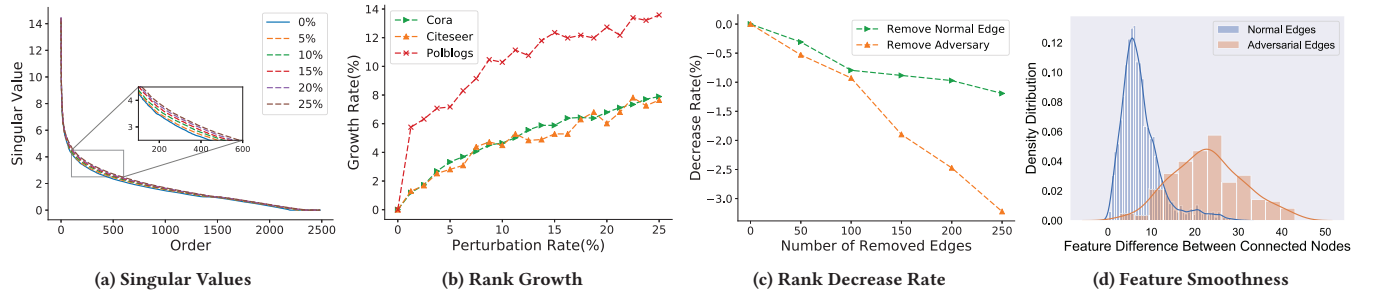
(a) Singular Values   **(b) Rank Growth**   **(c) Rank Decrease Rate**   **(d) Feature Smoothness**

Figure 1: An illustrative example on the property changes of the adjacency matrix by adversarial attacks

in Figure 1a, *metattack* enlarges the singular values of the adjacency matrix and Figure 1b illustrates that *metattack* quickly increases the rank of adjacency matrix. Moreover, when we remove the adversarial and normal edges from the perturbed graph respectively, we observe that removing adversarial edges reduces the rank faster than removing normal edges as demonstrated in Figure 1c. In addition, we depict the density distribution of feature difference of connected nodes of the attacked graph in Figure 1d. It is observed that *metattack* tends to connect nodes with large feature difference. Observations from Figure 1 indicate that adversarial attacks could violate these properties. Thus, these properties have the potential to serve as the guidance to clean the perturbed graph. However, work of exploring these properties to build robust graph neural networks is rather limited.

In this paper, we target on exploring graph properties of sparsity, low rank and feature smoothness to design robust graph neural networks. Note that there could be more properties to be explored and we would like to leave it as future work. In essence, we are faced with two challenges: (i) how to learn clean graph structure from poisoned graph data guided by these properties; and (ii) how to jointly learn parameters for robust graph neural network and the clean structure. To solve these two challenges, we propose a general framework Property GNN (Pro-GNN) to simultaneously learn the clean graph structure from perturbed graph and GNN parameters to defend against adversarial attacks. Extensive experiments on a variety of real-world graphs demonstrate that our proposed model can effectively defend against different types of adversarial attacks and outperforms the state-of-the-art defense methods.

The rest of the paper is organized as follows. In Section 2, we review some of the related work. In Section 3, we introduce notations and formally define the problem. We explain our proposed framework in Section 4 and report our experimental results in Section 5. Finally, we conclude the work with future directions in Section 6.

2 RELATED WORK

In line with the focus of our work, we briefly describe related work on GNNs, and adversarial attacks and defense for graph data.

2.1 Graph Neural Networks

Over the past few years, graph neural networks have achieved great success in solving machine learning problems on graph data. To learn effective representation of graph data, two main families of GNNs have been proposed, i.e., spectral methods and spatial methods. The first family learns node representation based on graph

spectral theory [6, 11, 18]. Bruna et al. [6] generalize the convolution operation from Euclidean data to non-Euclidean data by using the Fourier basis of a given graph. To simplify spectral GNNs, Defferrard et al. [11] propose ChebNet and utilize Chebyshev polynomials as the convolution filter. Kipf et al. [18] propose GCN and simplify ChebNet by using its first-order approximation. Further, Simple Graph Convolution (SGC) [31] reduces the graph convolution to a linear model but still achieves competitive performance. The second family of models define graph convolutions in the spatial domain as aggregating and transforming local information [14, 15, 30]. For instance, DCNN [2] treats graph convolutions as a diffusion process and assigns a certain transition probability for information transferred from one node to the adjacent node. Hamilton et al. [15] propose to learn aggregators by sampling and aggregating neighbor information. Veličković et al. [30] propose graph attention network (GAT) to learn different attention scores for neighbors when aggregating information. To further improve the training efficiency, FastGCN [8] interprets graph convolutions as integral transforms of embedding functions under probability measures and performs importance sampling to sample a fixed number of nodes for each layer. For a thorough review, we please refer the reader to recent surveys [3, 33, 36].

2.2 Adversarial Attacks and Defense for GNNs

Extensive studies have demonstrated that deep learning models are vulnerable to adversarial attacks. In other words, slight or unnoticeable perturbations to the input can fool a neural network to output a wrong prediction. GNNs also suffer this problem [5, 10, 17, 23, 24, 32, 39, 40]. Different from image data, the graph structure is discrete and the nodes are dependent of each other, thus making it far more challenging. The *nettack* [39] generates unnoticeable perturbations by preserving degree distribution and imposing constraints on feature co-occurrence. RL-S2V [10] employs reinforcement learning to generate adversarial attacks. However, both of the two methods are designed for targeted attack and can only degrade the performance of GNN on target nodes. To perturb the graph globally, *metattack* [40] is proposed to generate poisoning attacks based on meta-learning. Although increasing efforts have been devoted to developing adversarial attacks on graph data, the research about improving the robustness of GNNs has just started recently [28, 32, 38, 41]. One way to solve the problem is to learn a robust network by penalizing the attention scores of adversarial edges. RGCN [38] is to model Gaussian distributions as hidden layers to absorb the effects of adversarial attacks in the variances.

PA-GNN [28] leverages supervision knowledge from clean graphs and applies a meta-optimization way to learn attention scores for robust graph neural networks. However, it requires additional graph data from similar domain. The other way is to preprocess the perturbed graphs to get clean graphs and train GNNs on the clean ones. Wu et. al [32] have found that attackers tend to connect to nodes with different features and they propose to remove the links between dissimilar nodes. Entezari et al. [12] have observed that *nettack* results in changes in high-rank spectrum of the graph and propose to preprocess the graph with its low-rank approximations. However, due to the simplicity of two-stage preprocessing methods, they may fail to counteract complex global attacks.

Different from the aforementioned defense methods, we aim to explore important graph properties to recover the clean graph while learning the GNN parameters simultaneously, which enables the proposed model to extract intrinsic structure from perturbed graph under different attacks.

3 PROBLEM STATEMENT

Before we present the problem statement, we first introduce some notations and basic concepts. The Frobenius norm of a matrix $\mathbf{S}$ is defined by $||\mathbf{S}||_F^2 = \Sigma_{ij}\mathbf{S}_{ij}^2$. The ℓ_1 norm of a matrix $\mathbf{S}$ is given by $||\mathbf{S}||_1 = \Sigma_{ij}|\mathbf{S}_{ij}|$ and the nuclear norm of a matrix $\mathbf{S}$ is defined as $||\mathbf{S}||_* = \Sigma_{i=1}^{rank(\mathbf{S})}\sigma_i$, where σ_i is the i-th singular value of $\mathbf{S}$. $(\mathbf{S})_+$ denotes the element-wise positive part of matrix $\mathbf{S}$ where $\mathbf{S}_{ij} = \max\{\mathbf{S}_{ij}, 0\}$ and $sgn(\mathbf{S})$ indicates the sign matrix of $\mathbf{S}$ where $sgn(\mathbf{S})_{ij} = 1, 0$, or -1 if $\mathbf{S}_{ij}$ >0, =0, or <0, respectively. We use $\odot$ to denote Hadamard product of matrices. Finally, we use $tr(\mathbf{S})$ to indicate the trace of matrix $\mathbf{S}$, i.e., $tr(\mathbf{S}) = \sum_i \mathbf{S}_{ii}$.

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a graph, where $\mathcal{V}$ is the set of N nodes $\{v_1, v_2, ..., v_N\}$ and $\mathcal{E}$ is the set of edges. The edges describe the relations between nodes and can also be represented by an adjacency matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$ where $\mathbf{A}_{ij}$ denotes the relation between nodes v_i and v_j. Furthermore, we use $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N] \in \mathbb{R}^{N \times d}$ to denote the node feature matrix where $\mathbf{x}_i$ is the feature vector of the node v_i. Thus a graph can also be denoted as $\mathcal{G} = (\mathbf{A}, \mathbf{X})$. Following the common node classification setting, only a part of nodes $\mathcal{V}_L = \{v_1, v_2, ..., v_l\}$ are associated with corresponding labels $\mathcal{Y}_L = \{y_1, y_2, ..., y_l\}$ where y_i denotes the label of v_i.

Given a graph $\mathcal{G} = (\mathbf{A}, \mathbf{X})$ and the partial labels $\mathcal{Y}_L$, the goal of node classification for GNN is to learn a function $f_\theta : \mathcal{V}_L \rightarrow \mathcal{Y}_L$ that maps the nodes to the set of labels so that f_θ can predict labels of unlabeled nodes. The objective function can be formulated as

$$\min_\theta \mathcal{L}_{GNN}(\theta, \mathbf{A}, \mathbf{X}, \mathcal{Y}_L) = \sum_{v_i \in \mathcal{V}_L} \ell\left(f_\theta(\mathbf{X}, \mathbf{A})_i, y_i\right), \quad (1)$$

where θ is the parameters of f_θ, $f_\theta(\mathbf{X}, \mathbf{A})_i$ is the prediction of node v_i and $\ell(\cdot, \cdot)$ is to measure the difference between prediction and true label such as cross entropy. Though there exist a number of different GNN methods, in this work, we focus on Graph Convolution Network (GCN) in [18]. Note that it is straightforward to extend the proposed framework to other GNN models. Specifically, a two-layer GCN with $\theta = (\mathbf{W}_1, \mathbf{W}_2)$ implements f_θ as

$$f_\theta(\mathbf{X}, \mathbf{A}) = \text{softmax}\left(\hat{\mathbf{A}}\ \sigma\left(\hat{\mathbf{A}}\mathbf{X}\mathbf{W}_1\right)\mathbf{W}_2\right), \quad (2)$$
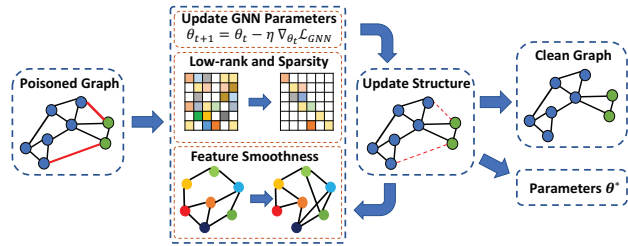
Figure 2: Overall framework of Pro-GNN. Dash lines indicate smaller weights.

where $\hat{\mathbf{A}} = \tilde{\mathbf{D}}^{-1/2}(\mathbf{A} + \mathbf{I})\tilde{\mathbf{D}}^{-1/2}$ and $\tilde{\mathbf{D}}$ is the diagonal matrix of $\mathbf{A} + \mathbf{I}$ with $\tilde{\mathbf{D}}_{ii} = 1 + \sum_j \mathbf{A}_{ij}$. σ is the activation function such as ReLU.

With aforementioned notations and definitions, the problem we aim to study in this work can be formally stated as:

Given $\mathcal{G} = (\mathbf{A}, \mathbf{X})$ and partial node label $\mathcal{V}_L$ with $\mathbf{A}$ being poisoned by adversarial edges and feature matrix $\mathbf{X}$ unperturbed, simultaneously learn a clean graph structure with the graph adjacency matrix $\mathbf{S} \in \mathcal{S} = [0, 1]^{N \times N}$ and the GNN parameters θ to improve node classification performance for unlabeled nodes.

4 THE PROPOSED FRAMEWORK

Adversarial attacks generate carefully-crafted perturbation on graph data. We refer to the carefully-crafted perturbation as adversarial structure. Adversarial structure can cause the performance of GNNs to drop rapidly. Thus, to defend adversarial attacks, one natural strategy is to eliminate the crafted adversarial structure, while maintaining the intrinsic graph structure. In this work, we aim to achieve the goal by exploring graph structure properties of low rank, sparsity and feature smoothness. The illustration of the framework is shown in Figure 2, where edges in black are normal edges and edges in red are adversarial edges introduced by an attacker to reduce the node classification performance. To defend against the attacks, Pro-GNN iteratively reconstructs the clean graph by preserving the low rank, sparsity, and feature smoothness properties of a graph so as to reduce the negative effects of adversarial structure. Meanwhile, to make sure that the reconstructed graph can help node classification, Pro-GNN simultaneously updates the GNN parameters on the reconstructed graph by solving the optimization problem in an alternating schema. In the following subsections, we will give the details of the proposed framework.

4.1 Exploring Low rank and Sparsity Properties

Many real-world graphs are naturally low-rank and sparse as the entities usually tend to form communities and would only be connected with a small number of neighbors [37]. Adversarial attacks on GCNs tend to add adversarial edges that link nodes of different communities as this is more efficient to reduce node classification performance of GCN. Introducing links connecting nodes of different communities in a sparse graph can significantly increase the rank of the adjacency matrix and enlarge the singular values, thus damaging the low rank and sparsity properties of graphs, which is verified in Figure 1a and Figure 1b. Thus, to recover the clean graph structure from the noisy and perturbed graph, one potential way is to learn a clean adjacency matrix $\mathbf{S}$ close to the adjacency matrix of the poisoned graph by enforcing the new adjacency matrix

with the properties of low rank and sparsity. As demonstrated in Figure1c, the rank decreases much faster by removing adversarial edges than by removing normal edges. This implies that the low rank and sparsity constraint can remove the adversarial edges instead of normal edges. Given the adjacency matrix $\mathbf{A}$ of a poisoned graph, we can formulate the above process as a structure learning problem [16, 27]:

$$\underset{\mathbf{S} \in \mathcal{S}}{\arg\min} \, \mathcal{L}_0 = \|\mathbf{A} - \mathbf{S}\|_F^2 + R(\mathbf{S}), \ s.t., \mathbf{S} = \mathbf{S}^\top. \tag{3}$$

Since adversarial attacks target on performing unnoticeable perturbations to graphs, the first term $\|\mathbf{A} - \mathbf{S}\|_F^2$ ensures that the new adjacency matrix $\mathbf{S}$ should be close to $\mathbf{A}$. As we assume that the graph are undirected, the new adjacency matrix should be symmetric, i.e., $\mathbf{S} = \mathbf{S}^\top$. $R(\mathbf{S})$ denotes the constraints on $\mathbf{S}$ to enforce the properties of low rank and sparsity. According to [7, 20, 27], minimizing the ℓ_1 norm and the nuclear norm of a matrix can force the matrix to be sparse and low-rank, respectively. Hence, to ensure a sparse and low-rank graph, we want to minimize the ℓ_1 norm and the nuclear norm of $\mathbf{S}$. Eq. (3) can be rewritten as:

$$\underset{\mathbf{S} \in \mathcal{S}}{\arg\min} \, \mathcal{L}_0 = \|\mathbf{A} - \mathbf{S}\|_F^2 + \alpha \|\mathbf{S}\|_1 + \beta \|\mathbf{S}\|_*, \ s.t., \mathbf{S} = \mathbf{S}^\top, \tag{4}$$

where α and β are predefined parameters that control the contributions of the properties of sparsity and low rank, respectively. One important benefit to minimize the nuclear norm $\|\mathbf{S}\|_*$ is that we can reduce every singular value, thus alleviating the impact of enlarging singular values from adversarial attacks.

4.2 Exploring Feature Smoothness

It is evident that connected nodes in a graph are likely to share similar features. In fact, this observation has been made on graphs from numerous domains. For example, two connected users in a social graph are likely to share similar attributes [25], two linked web pages in the webpage graph tend to have similar contents [29] and two connected papers in the citation network usually have similar topics [18]. Meanwhile, recently it is demonstrated that adversarial attacks on graphs tend to connect nodes with distinct features [32]. Thus, we aim to ensure the feature smoothness in the learned graph. The feature smoothness can be captured by the following term $\mathcal{L}_s$:

$$\mathcal{L}_s = \frac{1}{2} \sum_{i,j=1}^{N} \mathbf{S}_{ij} (\mathbf{x}_i - \mathbf{x}_j)^2, \tag{5}$$

where $\mathbf{S}$ is the new adjacency matrix, $\mathbf{S}_{ij}$ indicates the connection of v_i and v_j in the learned graph and $(\mathbf{x}_i - \mathbf{x}_j)^2$ measures the feature difference between v_i and v_j. $\mathcal{L}_s$ can be rewritten as:

$$\mathcal{L}_s = tr(\mathbf{X}^\top \mathbf{L} \mathbf{X}), \tag{6}$$

where $\mathbf{L} = \mathbf{D} - \mathbf{S}$ is the graph Laplacian matrix of $\mathbf{S}$ and $\mathbf{D}$ is the diagonal matrix of $\mathbf{S}$. In this work, we use normalized Laplacian matrix $\hat{\mathbf{L}} = \mathbf{D}^{-1/2} \mathbf{L} \mathbf{D}^{-1/2}$ instead of $\mathbf{L}$ to make feature smoothness independent on the degrees of the graph nodes [1], i.e.,

$$\mathcal{L}_s = tr(\mathbf{X}^\top \hat{\mathbf{L}} \mathbf{X}) = \frac{1}{2} \sum_{i,j=1}^{N} \mathbf{S}_{ij} \left(\frac{\mathbf{x}_i}{\sqrt{d_i}} - \frac{\mathbf{x}_j}{\sqrt{d_j}} \right)^2, \tag{7}$$

where d_i denotes the degree of v_i in the learned graph. In the learned graph, if v_i and v_j are connected (i.e., $\mathbf{S}_{ij} \neq 0$), we expect that the feature difference $(\mathbf{x}_i - \mathbf{x}_j)^2$ should be small. In other words, if the features between two connected node are quite different, $\mathcal{L}_s$ would be very large. Therefore, the smaller $\mathcal{L}_s$ is, the smoother features $\mathbf{X}$ are on the graph $\mathbf{S}$. Thus, to fulfill the feature smoothness in the learned graph, we should minimize $\mathcal{L}_s$. Therefore, we can add the feature smoothness term to the objective function of Eq. (4) to penalize rapid changes in features between adjacent nodes as:

$$\underset{\mathbf{S} \in \mathcal{S}}{\arg\min} \, \mathcal{L} = \mathcal{L}_0 + \lambda \cdot \mathcal{L}_s = \mathcal{L}_0 + \lambda \, tr(\mathbf{X}^\top \hat{\mathbf{L}} \mathbf{X}), \ s.t., \ \mathbf{S} = \mathbf{S}^\top, \tag{8}$$

where λ is a predefined parameter to control the contribution from feature smoothness.

4.3 Objective Function of Pro-GNN

Intuitively, we can follow the preprocessing strategy [12, 32] to defend against adversarial attacks – we first learn a graph from the poisoned graph via Eq. (8) and then train a GNN model based on the learned graph. However, with such a two-stage strategy, the learned graph may be suboptimal for the GNN model on the given task. Thus, we propose a better strategy to jointly learn the graph structure and the GNN model for a specific downstream task. We empirically show that jointly learning GNN model and the adjacency matrix is better than two stage one in Sec 5.4.2. The final objective function of Pro-GNN is given as

$$\underset{\mathbf{S} \in \mathcal{S}, \theta}{\arg\min} \, \mathcal{L} = \mathcal{L}_0 + \lambda \mathcal{L}_s + \gamma \mathcal{L}_{GNN} \tag{9}$$

$$= \|\mathbf{A} - \mathbf{S}\|_F^2 + \alpha \|\mathbf{S}\|_1 + \beta \|\mathbf{S}\|_* + \gamma \mathcal{L}_{GNN}(\theta, \mathbf{S}, \mathbf{X}, \mathcal{Y}_L) + \lambda tr(\mathbf{X}^\top \hat{\mathbf{L}} \mathbf{X})$$

$$s.t. \quad \mathbf{S} = \mathbf{S}^\top,$$

where $\mathcal{L}_{GNN}$ is a loss function for the GNN model that is controlled by a predefined parameter γ. Another benefit of this formulation is that the information from $\mathcal{L}_{GNN}$ can also guide the graph learning process to defend against adversarial attacks since the goal of graph adversarial attacks is to maximize $\mathcal{L}_{GNN}$.

4.4 An Optimization Algorithm

Jointly optimizing θ and $\mathbf{S}$ in Eq.(9) is challenging. The constraints on $\mathbf{S}$ further exacerbate the difficulty. Thus, in this work, we use an alternating optimization schema to iteratively update θ and $\mathbf{S}$.

Update θ. To update θ, we fix $\mathbf{S}$ and remove terms that are irrelevant to θ, then the objective function in Eq.(9) reduces to:

$$\min_{\theta} \mathcal{L}_{GNN}(\theta, \mathbf{S}, \mathbf{X}, \mathcal{Y}_L) = \sum_{u \in \mathcal{V}_L} \ell \left(f_\theta(\mathbf{X}, \mathbf{S})_u, y_u \right), \tag{10}$$

which is a typical GNN optimization problem and we can learn θ via stochastic gradient descent.

Update $\mathbf{S}$. Similarly, to update $\mathbf{S}$, we fix θ and arrive at

$$\min_{\mathbf{S}} \mathcal{L}(\mathbf{S}, \mathbf{A}) + \alpha \|\mathbf{S}\|_1 + \beta \|\mathbf{S}\|_* \quad s.t., \quad \mathbf{S} = \mathbf{S}^\top, \mathbf{S} \in \mathcal{S}, \tag{11}$$

where $\mathcal{L}(\mathbf{S}, \mathbf{A})$ is defined as

$$\mathcal{L}(\mathbf{S}, \mathbf{A}) = \|\mathbf{A} - \mathbf{S}\|_F^2 + \mathcal{L}_{GNN}(\theta, \mathbf{S}, \mathbf{X}, Y) + \lambda tr(\mathbf{X}^\top \hat{\mathbf{L}} \mathbf{X}). \tag{12}$$

Note that both ℓ_1 norm and nuclear norm are non-differentiable. For optimization problem with only one non-diffiential regularizer

---

Algorithm 1: Pro-GNN

> **Data:** Adjacency matrix $\mathbf{A}$, Attribute matrix $\mathbf{X}$, Labels $\mathcal{Y}_L$, Hyper-parameters $\alpha, \beta, \gamma, \lambda, \tau$, Learning rate η, η'
>
> **Result:** Learned adjacency $\mathbf{S}$, GNN parameters θ

1 Initialize $\mathbf{S} \leftarrow \mathbf{A}$

2 Randomly initialize θ

3 **while** *Stopping condition is not met* **do**

4    $\mathbf{S} \leftarrow \mathbf{S} - \eta \nabla_S (\|\mathbf{S} - \mathbf{A}\|_F^2 + \gamma \mathcal{L}_{GNN} + \lambda \mathcal{L}_s)$

5    $\mathbf{S} \leftarrow \text{prox}_{\eta\beta\|.\|_*}(\mathbf{S})$

6    $\mathbf{S} \leftarrow \text{prox}_{\eta\alpha\|.\|_1}(\mathbf{S})$

7    $\mathbf{S} \leftarrow P_{\mathcal{S}}(\mathbf{S})$

8   **for** *i=1 to τ* **do**

9       $g \leftarrow \frac{\partial \mathcal{L}_{GNN}(\theta, \mathbf{S}, \mathbf{X}, \mathcal{Y}_L)}{\partial \theta}$

10      $\theta \leftarrow \theta - \eta' g$

11 **Return** $\mathbf{S}, \theta$

$R(S)$, we can use Forward-Backward splitting methods [9]. The idea is to alternate a gradient descent step and a proximal step as:

$$\mathbf{S}^{(k)} = \text{prox}_{\eta R}\left(\mathbf{S}^{(k-1)} - \eta \nabla_S \mathcal{L}(S, A)\right), \tag{13}$$

where η is the learning rate, prox_R is the proximal operator as:

$$\text{prox}_R(\mathbf{Z}) = \underset{\mathbf{S} \in \mathbb{R}^{N \times N}}{\arg\min} \frac{1}{2}\|\mathbf{S} - \mathbf{Z}\|_F^2 + R(\mathbf{S}). \tag{14}$$

In particular, the proximal operator of ℓ_1 norm and nuclear norm can be represented as [4, 27],

$$\text{prox}_{\alpha\|.\|_1}(\mathbf{Z}) = sgn(\mathbf{Z}) \odot (|\mathbf{Z}| - \alpha)_+, \tag{15}$$

$$\text{prox}_{\beta\|.\|_*}(\mathbf{Z}) = \mathbf{U}\, diag((\sigma_i - \beta)_+)_i \mathbf{V}^T, \tag{16}$$

where $\mathbf{Z} = \mathbf{U}\, diag(\sigma_1, ..., \sigma_n)\mathbf{V}^\top$ is the singular value decomposition of $\mathbf{Z}$. To optimize objective function with two non-differentiable regularizers, Richard et al. [26] introduce the Incremental Proximal Descent method based on the introduced proximal operators. By iterating the updating process in a cyclic manner, we can update $\mathbf{S}$ as follows,

$$\begin{cases} \mathbf{S}^{(k)} = \mathbf{S}^{(k-1)} - \eta \cdot \nabla_S \left(\mathcal{L}(\mathbf{S}, \mathbf{A})\right), \\ \mathbf{S}^{(k)} = \text{prox}_{\eta\beta\|\cdot\|_*}\left(\mathbf{S}^{(k)}\right), \\ \mathbf{S}^{(k)} = \text{prox}_{\eta\alpha\|\cdot\|_1}\left(\mathbf{S}^{(k)}\right). \end{cases} \tag{17}$$

After we learn a relaxed $\mathbf{S}$, we project $\mathbf{S}$ to satisfy the constraints. For the symmetric constraint, we let $\mathbf{S} = \frac{\mathbf{S} + \mathbf{S}^\top}{2}$. For the constraint $\mathbf{S}_{ij} \in [0, 1]$, we project $\mathbf{S}_{ij} < 0$ to 0 and $\mathbf{S}_{ij} > 1$ to 1. We denote these projection operations as $P_{\mathcal{S}}(\mathbf{S})$.

Training Algorithm. With these updating and projection rules, the optimization algorithm is shown in Algorithm 1. In line 1, we first initialize the estimated graph $\mathbf{S}$ as the poisoned graph $\mathbf{A}$. In line 2, we randomly initialize the GNN parameters. From lines 3 to 10, we update $\mathbf{S}$ and the GNN parameters θ alternatively and iteratively. Specifically, we train the GNN parameters in each iteration while training the graph reconstruction model every τ iterations.

Table 1: Dataset Statistics. Following [12, 39, 40], we only consider the largest connected component (LCC).

	N_{LCC}	E_{LCC}	Classes	Features
Cora	2,485	5,069	7	1,433
Citeseer	2,110	3,668	6	3,703
Polblogs	1,222	16,714	2	/
Pubmed	19,717	44,338	3	500

5 EXPERIMENTS

In this section, we evaluate the effectiveness of Pro-GNN against different graph adversarial attacks. In particular, we aim to answer the following questions:

- **RQ1** How does Pro-GNN perform compared to the state-of-the-art defense methods under different adversarial attacks?
- **RQ2** Does the learned graph work as expected?
- **RQ3** How do different properties affect performance of Pro-GNN.

Before presenting our experimental results and observations, we first introduce the experimental settings.

5.1 Experimental settings

5.1.1 Datasets. Following [39, 40], we validate the proposed approach on four benchmark datasets, including three citation graphs, i.e., Cora, Citeseer and Pubmed, and one blog graph, i.e., Polblogs. The statistics of the datasets are shown in Table 1. Note that in the Polblogs graph, node features are not available. In this case, we set the attribute matrix to $N \times N$ identity matrix.

5.1.2 Baselines. To evaluate the effectiveness of Pro-GNN, we compare it with the state-of-the-art GNN and defense models by using the adversarial attack repository DeepRobust [21]:

- **GCN** [18]: while there exist a number of different Graph Convolutional Networks (GCN) models, we focus on the most representative one [18].
- **GAT** [30]: Graph Attention Netowork (GAT) is composed of attention layers which can learn different weights to different nodes in the neighborhood. It is often used as a baseline to defend against adversarial attacks.
- **RGCN** [38]: RGCN models node representations as gaussian distributions to absorb effects of adversarial attacks. It also employs attention mechanism to penalize nodes with high variance.
- **GCN-Jaccard** [32]: Since attackers tend to connect nodes with dissimilar features or different labels, GCN-Jaccard preprocesses the network by eliminating edges that connect nodes with jaccard similarity of features smaller than threshold τ. Note that this method only works when node features are available.
- **GCN-SVD** [12]: This is another preprocessing method to resist adversarial attacks. It is noted that *nettack* is a high-rank attack, thus GCN-SVD proposes to vaccinate GCN with the low-rank approximation of the perturbed graph. Note that it originally targets at defending against *nettack*, however, it is straightforward to extend it to non-targeted and random attacks.

In addition to representative baselines, we also include one variant of the proposed framework, Pro-GNN-fs, which is the variant by eliminating the feature smoothness term (or setting $\lambda = 0$).

Table 2: Node classification performance (Accuracy±Std) under non-targeted attack (*metattack*).

Dataset	Ptb Rate (%)	GCN	GAT	RGCN	GCN-Jaccard[2]	GCN-SVD	Pro-GNN-fs	Pro-GNN[3]
Cora	0	83.50±0.44	**83.97±0.65**	83.09±0.44	82.05±0.51	80.63±0.45	83.42±0.52	82.98±0.23
	5	76.55±0.79	80.44±0.74	77.42±0.39	79.13±0.59	78.39±0.54	**82.78±0.39**	82.27±0.45
	10	70.39±1.28	75.61±0.59	72.22±0.38	75.16±0.76	71.47±0.83	77.91±0.86	**79.03±0.59**
	15	65.10±0.71	69.78±1.28	66.82±0.39	71.03±0.64	66.69±1.18	76.01±1.12	**76.40±1.27**
	20	59.56±2.72	59.94±0.92	59.27±0.37	65.71±0.89	58.94±1.13	68.78±5.84	**73.32±1.56**
	25	47.53±1.96	54.78±0.74	50.51±0.78	60.82±1.08	52.06±1.19	56.54±2.58	**69.72±1.69**
Citeseer	0	71.96±0.55	73.26±0.83	71.20±0.83	72.10±0.63	70.65±0.32	73.26±0.38	**73.28±0.69**
	5	70.88±0.62	72.89±0.83	70.50±0.43	70.51±0.97	68.84±0.72	**73.09±0.34**	72.93±0.57
	10	67.55±0.89	70.63±0.48	67.71±0.30	69.54±0.56	68.87±0.62	72.43±0.52	**72.51±0.75**
	15	64.52±1.11	69.02±1.09	65.69±0.37	65.95±0.94	63.26±0.96	70.82±0.87	**72.03±1.11**
	20	62.03±3.49	61.04±1.52	62.49±1.22	59.30±1.40	58.55±1.09	66.19±2.38	**70.02±2.28**
	25	56.94±2.09	61.85±1.12	55.35±0.66	59.89±1.47	57.18±1.87	66.40±2.57	**68.95±2.78**
Polblogs	0	**95.69±0.38**	95.35±0.20	95.22±0.14	-	95.31±0.18	93.20±0.64	-
	5	73.07±0.80	83.69±1.45	74.34±0.19	-	89.09±0.22	**93.29±0.18**	-
	10	70.72±1.13	76.32±0.85	71.04±0.34	-	81.24±0.49	**89.42±1.09**	-
	15	64.96±1.91	68.80±1.14	67.28±0.38	-	68.10±3.73	**86.04±2.21**	-
	20	51.27±1.23	51.50±1.63	59.89±0.34	-	57.33±3.15	**79.56±5.68**	-
	25	49.23±1.36	51.19±1.49	56.02±0.56	-	48.66±9.93	**63.18±4.40**	-
Pubmed	0	87.19±0.09	83.73±0.40	86.16±0.18	87.06±0.06	83.44±0.21	**87.33±0.18**	87.26±0.23
	5	83.09±0.13	78.00±0.44	81.08±0.20	86.39±0.06	83.41±0.15	**87.25±0.09**	87.23±0.13
	10	81.21±0.09	74.93±0.38	77.51±0.27	85.70±0.07	83.27±0.21	**87.25±0.09**	87.21±0.13
	15	78.66±0.12	71.13±0.51	73.91±0.25	84.76±0.08	83.10±0.18	**87.20±0.09**	87.20±0.15
	20	77.35±0.19	68.21±0.96	71.18±0.31	83.88±0.05	83.01±0.22	87.09±0.10	**87.15±0.15**
	25	75.50±0.17	65.41±0.77	67.95±0.15	83.66±0.06	82.72±0.18	86.71±0.09	**86.76±0.19**

[1][2] JaccardGCN and Pro-GNN cannot be directly applied to datasets where node features are not available.

5.1.3 Parameter Settings. For each graph, we randomly choose 10% of nodes for training, 10% of nodes for validation and the remaining 80% of nodes for testing. For each experiment, we report the average performance of 10 runs. The hyper-parameters of all the models are tuned based on the loss and accuracy on validation set. For GCN and GAT, we adopt the default parameter setting in the author's implementation. For RGCN, the number of hidden units are tuned from {16, 32, 64, 128}. For GCN-Jaccard, the threshold of similarity for removing dissimilar edges is chosen from {0.01, 0.02, 0.03, 0.04, 0.05, 0.1}. For GCN-SVD , the reduced rank of the perturbed graph is tuned from {5, 10, 15, 50, 100, 200}.

5.2 Defense Performance

To answer the first question, we evaluate the node classification performance of Pro-GNN against three types of attacks, i.e., non-targeted attack, targeted attack and random attack:

- **Targeted Attack:** Targeted attack generates attacks on specific nodes and aims to fool GNNs on these target nodes. We adopt *nettack* [39] for the targeted attack method, which is the state-of-the-art targeted attack on graph data.
- **Non-targeted Attack:** Different from targeted attack, the goal of non-targeted attack is to degrade the overall performance of GNNs on the whole graph. We adopt one representative non-targeted attack, *metattack* [40] .
- **Random Attack:** It randomly injects fake edges into the graph. It can also be viewed as adding random noise to the clean graph.

We first use the attack method to poison the graph. We then train Pro-GNN and baselines on the poisoned graph and evaluate the node classification performance achieved by these methods.

5.2.1 Against Non-targeted Adversarial Attacks. We first evaluate the node classification accuracy of different methods against non-targeted adversarial attack. Specifically, we adopt *metattack* and keep all the default parameter settings in the authors' original implementation. The *metattack* has several variants. For Cora, Citeseer and Polblogs datasets, we apply Meta-Self since it is the most destructive attack variant; while for Pubmed, the approximate version of Meta-Self, A-Meta-Self is applied to save memory and time. We vary the perturbation rate, i.e., the ratio of changed edges, from 0 to 25% with a step of 5%. As mentioned before, all the experiments are conducted 10 times and we report the average accuracy with standard deviation in Table 2. The best performance is highlighted in bold. From the table, we make the following observations:

- Our method consistently outperforms other methods under different perturbation rates. For instance, on Polblogs dataset our model improves GCN over 20% at 5% perturbation rate. Even under large perturbation, our method outperforms other baselines by a larger margin. Specifically, under the 25% perturbation rate on the three datasets, vanilla GCN performs very poorly and our model improves GCN by 22%, 12% and 14%, respectively.
- Although GCN-SVD also employs SVD to get low-rank approximation of the graph, the performance of GCN-SVD drops rapidly. This is because GCN-SVD is designed for targeted attack, it cannot adapt well to the non-targeted adversarial attack. Similarly,
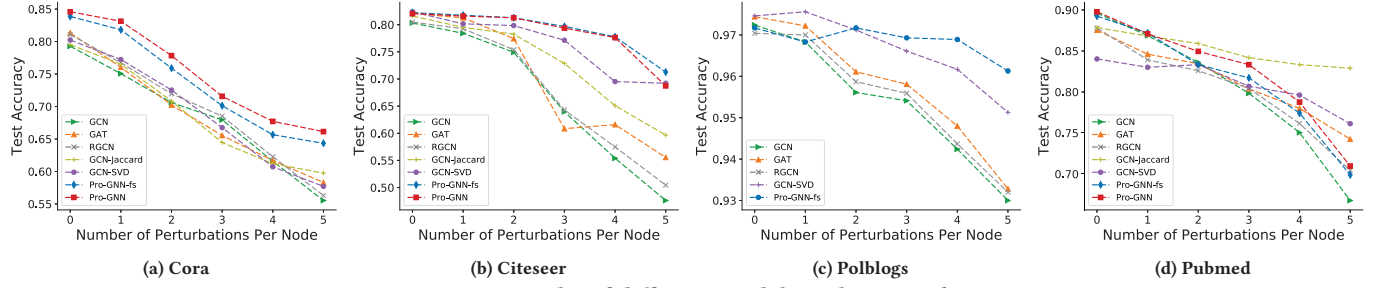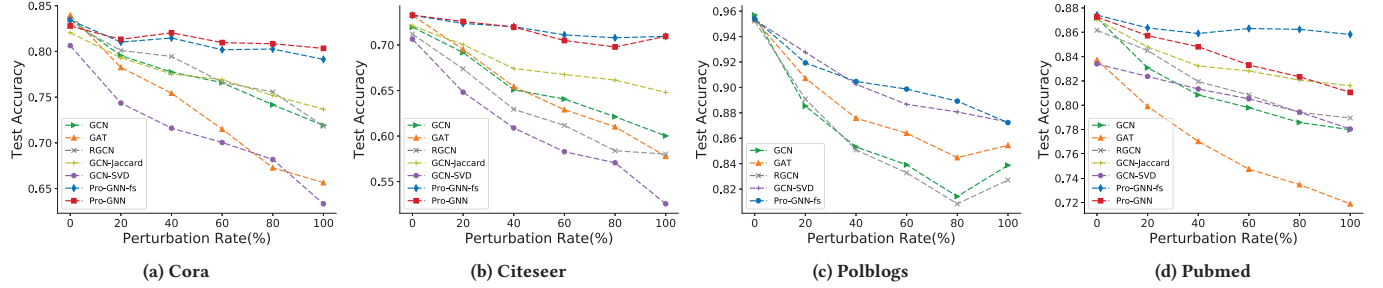
(a) Cora **(b) Citeseer** **(c) Polblogs** **(d) Pubmed**

Figure 3: Results of different models under *nettack*

(a) Cora **(b) Citeseer** **(c) Polblogs** **(d) Pubmed**

Figure 4: Results of different models under random attack

GCN-Jaccard does not perform as well as Pro-GNN under different perturbation rates. This is because simply preprocessing the perturbed graph once cannot recover the complex intrinsic graph structure from the carefully-crafted adversarial noises. On the contrary, simultaneously updating the graph structure and GNN parameters with the low rank, sparsity and feature smoothness constraints helps recover better graph structure and learn robust GNN parameters.

- Pro-GNN achieves higher accuracy than Pro-GNN-fs especially when the perturbation rate is large, which demonstrates the effectiveness of feature smoothing in removing adversarial edges.

5.2.2 Against Targeted Adversarial Attack. In this experiment, *nettack* is adopted as the targeted-attack method and we use the default parameter settings in the authors' original implementation. Following [38], we vary the number of perturbations made on every targeted node from 1 to 5 with a step size of 1. The nodes in test set with degree larger than 10 are set as target nodes. For Pubmed dataset, we only sample 10% of them to reduce the running time of *nettack* while in other datasets we use all the target nodes. The node classification accuracy on target nodes is shown in Figure 3. From the figure, we can observe that when the number of perturbation increases, the performance of our method is better than other methods on the attacked target nodes in most cases. For instance, on Citeseer dataset at 5 perturbation per targeted node, our model improves vanilla GCN by 23% and outperforms other defense methods by 11%. It demonstrates that our method can also resist the targeted adversarial attack.

5.2.3 Against Random Attack. In this subsection, we evaluate how Pro-GNN behaves under different ratios of random noises from 0% to 100% with a step size of 20%. The results are reported in Figure 4. The figure shows that Pro-GNN consistently outperforms all other baselines and successfully resists random attack. Together with

observations from Sections 5.2.1 and 5.2.2, we can conclude that Pro-GNN is able to defend various types of adversarial attacks. This is a desired property in practice since attackers can adopt any kinds of attacks to fool the system.

5.3 Importance of Graph Structure Learning

In the previous subsection, we have demonstrated the effectiveness of the proposed framework. In this section, we aim to understand the graph we learned and answer the second question.
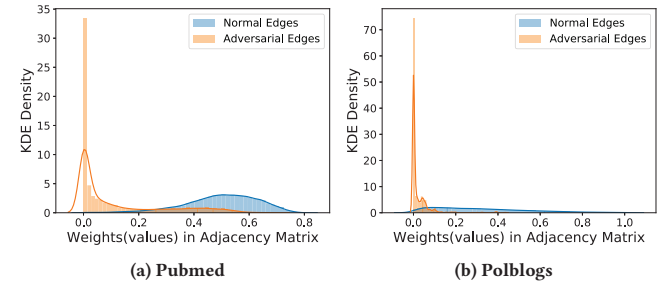

(a) Pubmed **(b) Polblogs**

Figure 5: Weight density distributions of normal and adversarial edges on the learned graph.

5.3.1 Normal Edges Against Adversarial Edges. Based on the fact that adversary tends to add edges over delete edges [32, 40], if the model tends to learn a clean graph structure, the impact of the adversarial edges should be mitigated from the poisoned graph. Thus, we investigate the weights of normal and adversarial edges in the learned adjacency matrix $\mathbf{S}$. We visualize the weight density distribution of normal and perturbed edges of $\mathbf{S}$ in Figure 5. Due to the limit of space, we only show results on Pubmed and Polblogs under *metattack*. As we can see in the figure, in both datasets, the weights of adversarial edges are much smaller than those of normal edges, which shows that Pro-GNN can alleviate the effect of adversarial edges and thus learn robust GNN parameters.

Table 3: Node classification accuracy given the graph under 25% perturbation by _metattack_.

	GCN	GCN-NoGraph	Pro-GNN
Cora	47.53±1.96	62.12±1.55	**69.72±1.69**
Citeseer	56.94±2.09	63.75±3.23	**68.95±2.78**
Polblogs	49.23±1.36	51.79±0.62	**63.18±4.40**
Pubmed	75.50±0.17	84.14±0.11	**86.86±0.19**

5.3.2 Performance on Heavily Poisoned Graph. In this subsection, we study the performance when the graph is heavily poisoned. In particular, we poison the graph with 25% perturbation by _metattack_. If a graph is heavily poisoned, the performance of GCN will degrade a lot. One straightforward solution is to remove the poisoned graph structure. Specifically, when removing the graph structure, the adjacency matrix will be all zeros and GCN normalizes the zero matrix into identity matrix and then makes prediction totally by node features. Under this circumstance, GCN actually becomes a feed-forward neural network. We denote it as GCN-NoGraph. We report the performance of GCN, GCN-NoGraph and Pro-GNN when the graph is heavily poisoned in Table 3.

From the table, we first observe that when the graph structure is heavily poisoned, by removing the graph structure, GCN-NoGraph outperforms GCN. This observation suggests the necessity to defend poisoning attacks on graphs because the poisoned graph structure are useless or even hurt the prediction performance. We also note that Pro-GNN obtains much better results than GCN-NoGraph. This observation suggests that Pro-GNN can learn useful graph structural information even when the graph is heavily poisoned.

5.4 Ablation Study

To get a better understanding of how different components help our model defend against adversarial attacks, we conduct ablation studies and answer the third question in this subsection.

5.4.1 Regularizers. There are four key predefined parameters, i.e., α, β, γ and λ, which control the contributions for sparsity, low rank, GNN loss and feature smoothness, respectively. To understand the impact of each component, we vary the values of one parameter and set other parameters to zero, and then check how the performance changes. Correspondingly, four model variants are created: Pro-GNN-α, Pro-GNN-β, Pro-GNN-γ and Pro-GNN-λ. For example, Pro-GNN-α denotes that we vary the values of α while setting β, γ and λ to zero. We only report results on Cora and Citeseer, since similar patterns are observed in other cases, shown in Figure 6.

From the figure we can see Pro-GNN-α does not boost the model's performance too much with small perturbations. But when the perturbation becomes large, Pro-GNN-α outperforms vanilla GCN because it can learn a graph structure better than a heavily poisoned adjacency graph as shown in Section 5.3.2. Also, Pro-GNN-β and Pro-GNN-λ perform much better than vanilla GCN. It is worth noting that, Pro-GNN-β outperforms all other variants except Pro-GNN, indicating that nuclear norm is of great significance in reducing the impact of adversarial attacks. It is in line with our observation that adversarial attacks increase the rank of the graph and enlarge the singular values. Another observation from the figure is that, Pro-GNN-γ works better under small perturbation and when the perturbation rate increases, its performance

(a) Cora **(b) Citeseer**
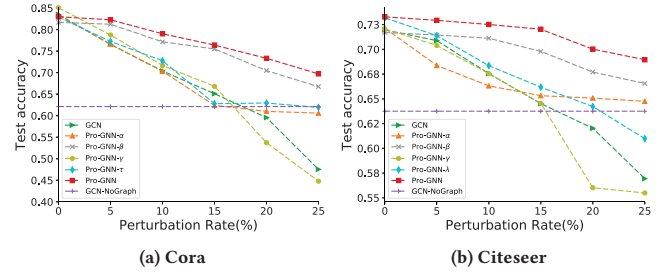
Figure 6: Classification performance of Pro-GNN variants.

degrades. From the above observations, different components play different roles in defending adversarial attacks. By incorporating these components, Pro-GNN can explore the graph properties and thus consistently outperform state-of-the-art baselines.

5.4.2 Two-Stage vs One-Stage. To study the contribution of jointly learning structure and GNN parameters, we conduct experiments with the variant Pro-GNN-two under _metattack_. Pro-GNN-two is the two stage variant of Pro-GNN where we first obtain the clean graph and then train a GNN model based on it. We only show the results on Cora in Table 4 due to the page limitation. We can observe from the results that although Pro-GNN-two can achieve good performance under large perturbation, it fails to defend the attacks when the perturbation rate is relatively low. The results demonstrate that jointly learning structure and GNN parameters can actually help defend attacks.

5.5 Parameter Analysis

In this subsection, we explore the sensitivity of hyper-parameters α, β, γ and λ for Pro-GNN. In the experiments, we alter the value of α, β, γ and λ to see how they affect the performance of our model. More specifically, we vary α from 0.00025 to 0.064 in a log scale of base 2, β from 0 to 5, γ from 0.0625 to 16 in a log scale of base 2 and λ from 1.25 to 320 in a log scale of base 2. We only report the results on Cora dataset with the perturbation rate of 10% by _metattack_ since similar observations are made in other settings.

The performance change of Pro-GNN is illustrated in Figure 7. As we can see, the accuracy of Pro-GNN can be boosted when choosing appropriate values for all the hyper-parameters. Different from γ, appropriate values of α and λ can boost the performance but large values will greatly hurt the performance. This is because focusing on sparsity and feature smoothness will result in inaccurate estimation on the graph structure. For example, if we set α and λ to $+\infty$, we will get a trivial solution of the new adjacency matrix, i.e, $\mathbf{S} = 0$. It is worth noting that, appropriate value of β can greatly increase the model's performance (more than 10%) compared with the variant without β, while too large or too small value of β will hurt the performance. This is also consistent with our observation in Section 5.4.1 that the low rank property plays an important role in defending adversarial attacks.

6 CONCLUSION

Graph neural networks can be easily fooled by graph adversarial attacks. To defend against different types of graph adversarial attacks, we introduced a novel defense approach Pro-GNN that learns the graph structure and the GNN parameters simultaneously.

Table 4: Classification performance of Pro-GNN-two and Pro-GNN on Cora dataset

Ptb Rate (%)	0	5	10	15	20	25
Pro-GNN-two	73.31±0.71	73.70±1.02	73.69±0.81	75.38±1.10	73.22±1.08	**70.57±0.61**
Pro-GNN	**82.98±0.23**	**82.27±0.45**	**79.03±0.59**	**76.40±1.27**	**73.32±1.56**	69.72±1.69
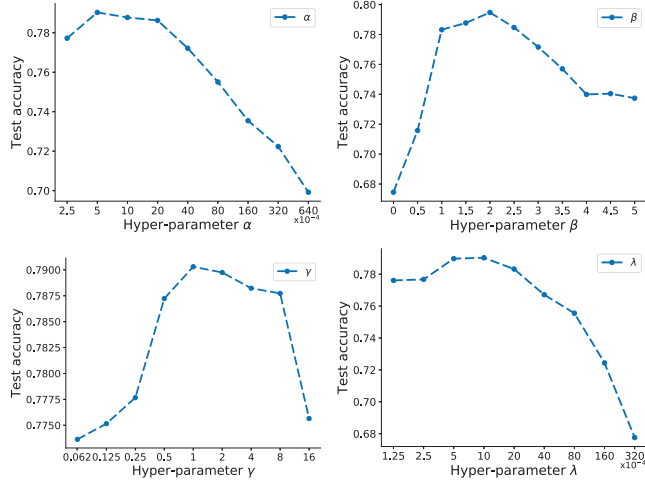
Figure 7: Results of parameter analysis on Cora dataset

Our experiments show that our model consistently outperforms state-of-the-art baselines and improves the overall robustness under various adversarial attacks. In the future, we aim to explore more properties to further improve the robustness of GNNs.

7 ACKNOWLEDGEMENTS

REFERENCES

[1] Rie K Ando and Tong Zhang. 2007. Learning on graph with Laplacian regularization. In *NeurIPS*.
[2] James Atwood and Don Towsley. 2016. Diffusion-convolutional neural networks. In *NeurIPS*.
[3] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. 2018. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261* (2018).
[4] Amir Beck and Marc Teboulle. 2009. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM journal on imaging sciences* (2009).
[5] Aleksandar Bojchevski and Stephan Günnemann. 2019. Adversarial Attacks on Node Embeddings via Graph Poisoning. In *ICML*.
[6] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. 2013. Spectral Networks and Locally Connected Networks on Graphs. arXiv:cs.LG/1312.6203
[7] Emmanuel J Candès and Benjamin Recht. 2009. Exact matrix completion via convex optimization. *Foundations of Computational mathematics* 9, 6 (2009), 717.
[8] Jie Chen, Tengfei Ma, and Cao Xiao. 2018. Fastgcn: fast learning with graph convolutional networks via importance sampling. In *ICLR*.
[9] Patrick L Combettes and Jean-Christophe Pesquet. 2011. Proximal splitting methods in signal processing. In *Fixed-point algorithms for inverse problems in science and engineering*. Springer, 185–212.
[10] Hanjun Dai, Hui Li, Tian Tian, Xin Huang, Lin Wang, Jun Zhu, and Le Song. 2018. Adversarial attack on graph structured data. In *ICML*.
[11] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *NeurIPS*.
[12] Negin Entezari, Saba A Al-Sayouri, Amirali Darvishzadeh, and Evangelos E Papalexakis. 2020. All You Need Is Low (Rank) Defending Against Adversarial Attacks on Graphs. In *WSDM*.

[13] Santo Fortunato. 2010. Community detection in graphs. *Physics reports* (2010).
[14] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. 2017. Neural message passing for quantum chemistry. 1263–1272.
[15] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *NeurIPS*.
[16] Bo Jiang, Ziyan Zhang, Doudou Lin, Jin Tang, and Bin Luo. 2019. Semi-supervised learning with graph learning-convolutional networks. In *CVPR*.
[17] Wei Jin, Yaxin Li, Han Xu, Yiqi Wang, and Jiliang Tang. 2020. Adversarial Attacks and Defenses on Graphs: A Review and Empirical Study. arXiv:cs.LG/2003.00653
[18] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).
[19] Thomas N Kipf and Max Welling. 2016. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308* (2016).
[20] Vladimir Koltchinskii, Karim Lounici, Alexandre B Tsybakov, et al. 2011. Nuclear-norm penalization and optimal rates for noisy low-rank matrix completion. *The Annals of Statistics* 39, 5 (2011), 2302–2329.
[21] Yaxin Li, Wei Jin, Han Xu, and Jiliang Tang. 2020. DeepRobust: A PyTorch Library for Adversarial Attacks and Defenses. arXiv:cs.LG/2005.06149
[22] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. 2015. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493* (2015).
[23] Xuanqing Liu, Si Si, Jerry Zhu, Yang Li, and Cho-Jui Hsieh. 2019. A Unified Framework for Data Poisoning Attack to Graph-based Semi-supervised Learning. In *NeurIPS*.
[24] Yao Ma, Suhang Wang, Lingfei Wu, and Jiliang Tang. 2019. Attacking Graph Convolutional Networks via Rewiring. *arXiv preprint arXiv:1906.03750* (2019).
[25] Miller McPherson, Lynn Smith-Lovin, and James M Cook. 2001. Birds of a feather: Homophily in social networks. *Annual review of sociology* 27, 1 (2001), 415–444.
[26] Hugo Raguet, Jalal Fadili, and Gabriel Peyré. 2013. A generalized forward-backward splitting. *SIAM Journal on Imaging Sciences* 6, 3 (2013), 1199–1226.
[27] Emile Richard, Pierre-André Savalle, and Nicolas Vayatis. 2012. Estimation of simultaneously sparse and low rank matrices. In *ICML*.
[28] Xianfeng Tang, Yandong Li, Yiwei Sun, Huaxiu Yao, Prasenjit Mitra, and Suhang Wang. 2019. Robust Graph Neural Network Against Poisoning Attacks via Transfer Learning. *arXiv preprint arXiv:1908.07558* (2019).
[29] Liwen Vaughan, Margaret EI Kipp, and Yijun Gao. 2007. Why are websites co-linked? The case of Canadian universities. *Scientometrics* 72, 1 (2007), 81–92.
[30] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2018. Graph attention networks. *ICLR*.
[31] Felix Wu, Tianyi Zhang, Amauri Holanda de Souza Jr, Christopher Fifty, Tao Yu, and Kilian Q Weinberger. 2019. Simplifying graph convolutional networks. *arXiv preprint arXiv:1902.07153* (2019).
[32] Huijun Wu, Chen Wang, Yuriy Tyshetskiy, Andrew Docherty, Kai Lu, and Liming Zhu. 2019. Adversarial examples for graph data: deep insights into attack and defense. In *IJCAI*.
[33] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S Yu. 2019. A comprehensive survey on graph neural networks. *arXiv preprint arXiv:1901.00596* (2019).
[34] Han Xu, Yao Ma, Haochen Liu, Debayan Deb, Hui Liu, Jiliang Tang, and Anil Jain. 2019. Adversarial attacks and defenses in images, graphs and text: A review. *arXiv preprint arXiv:1909.08072* (2019).
[35] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. 2018. Graph convolutional neural networks for web-scale recommender systems. In *KDD*.
[36] Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, and Maosong Sun. 2018. Graph neural networks: A review of methods and applications. *arXiv preprint arXiv:1812.08434* (2018).
[37] Ke Zhou, Hongyuan Zha, and Le Song. 2013. Learning social infectivity in sparse low-rank networks using multi-dimensional hawkes processes. In *Artificial Intelligence and Statistics*. 641–649.
[38] Dingyuan Zhu, Ziwei Zhang, Peng Cui, and Wenwu Zhu. 2019. Robust graph convolutional networks against adversarial attacks. In *KDD*.
[39] Daniel Zügner, Amir Akbarnejad, and Stephan Günnemann. 2018. Adversarial attacks on neural networks for graph data. In *KDD*. ACM.
[40] Daniel Zügner and Stephan Günnemann. 2019. Adversarial attacks on graph neural networks via meta learning. *arXiv preprint arXiv:1902.08412* (2019).
[41] Daniel Zügner and Stephan Günnemann. 2019. Certifiable robustness and robust training for graph convolutional networks. In *KDD*.