DEFENDING GRAPH NEURAL NETWORKS VIA TENSOR-BASED ROBUST GRAPH AGGREGATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Graph Neural Networks (GNNs) have achieved outstanding success in a wide variety of domains and applications. However, they are still vulnerable to unnoticeable perturbations of graphs specially designed by attackers, causing significant performance drops. Developing algorithms to defend GNNs with robust graphs vaccinating from adversarial attacks still remains a challenging issue. Existing methods treat every edges individually or regularize them by specific robust properties, which ignores the structural relationships among edges or correlations among different properties. In this paper, we propose a tensor-based framework for GNNs to learn robust graphs from adversarial graphs by aggregating predefined robust graphs to enhance the robustness of GNNs via tensor approximation. All the predefined robust graphs are linearly compressed into and recovered from a low-rank space, which aggregates the robust graphs and the structural information in a balanced manner. Extensive experiments on real-world graph datasets show that the proposed framework effectively mitigates the adverse effects of adversarial attacks and outperforms state-of-the-art defense methods.

1 INTRODUCTION

In real-world applications, many complex structures from various domains can be represented by graph data, such as social networks, chemistry (molecular), biology (protein-protein interaction networks), knowledge databases, and recommender systems. Graph Neural Networks (GNNs) have achieved outstanding success for graph data in a wide range of application areas (Kipf & Welling, 2017; Battaglia et al., 2018; Wu et al., 2021; Zhou et al., 2020). GNNs are equipped with a message-passing scheme (Gilmer et al., 2017), where the node representation is updated by transforming the aggregated representations propagated from its neighbor nodes via edges, and are optimized for performance on downstream tasks. In this fashion, GNNs can generate representations capturing both graph structural information and node feature information.

In this paper, we focus on the semi-supervised task of node classification (Kipf & Welling, 2017) by GNNs. GNNs have achieved promising performance in recent years, but have also shown to be vulnerable to adversarial attacks (Bojchevski & Günnemann, 2019; Dai et al., 2018; Jin et al., 2020a; Liu et al., 2019; Ma et al., 2019; Wu et al., 2019b; Zügner et al., 2018; Zügner & Günnemann, 2019b). Most of adversarial attack methods for GNNs focus on poisoning attacks, which deliberately rebuild the graph topology by changing an unnoticeable number of edges before the training process of GNNs. These unnoticeable perturbations can significantly degrade learned graph representations and confuse GNNs to catastrophically misclassify nodes. To defend the GNNs from adversarial edges, existing methods attempt to learn robust graphs, which are extracted from the perturbed graphs and perform solidly with GNNs. One way is to learn robust graph by reducing the attention weights of potentially malicious edges (Zhu et al., 2019; Tang et al., 2019; 2020; Zhang & Zitnik, 2020). Preprocessing the perturbed graphs to obtain robust graphs is another way to defend GNNs from adversarial edges (Wu et al., 2019b; Entezari et al., 2020). Most of the existing works only pay attention to a specific property of graph data (*e.g.*, low-rank property (Entezari et al., 2020; Al-Sayouri et al., 2020), feature similarities (Wu et al., 2019b)). Pro-GNN (Jin et al., 2020b) regularizes the learned graph to fit a set of the corresponding graph properties and learns the weights for each node instead of the whole graph. These methods show improvements in defending GNNs from poisoning attacks but ignore the connection and balance between different graph properties or different nodes. *The learned robust graph has every single edge with a weight close to the original graph and following the specific*

graph properties, however, when looking through the point of view from the whole graph, it is either ignoring the original structure of the graph or not able to remove most of the perturbed edges.

To learn better robust graph from adversarial graphs, we propose a Tensor-based framework for Graph Neural Networks (TGNN), which can aggregate common information from the predefined robust graphs and the adversarial graphs and thus enhance the robustness of GNNs with improved classification performance. TGNN approximates the graph tensor, which consists of the attacked graph and several predefined robust graphs focusing on specific graph properties (*e.g.*, low-rank, feature similarity properties), via low-rank approximation by applying tensor decomposition. By preserving the low-rank tensor representation, TGNN generates the robust graph and maintains the common core tensor space between the learned robust graph and predefined robust graphs. The learned robust graph is no longer calculated individually by each entry but aggregated with the common core tensors, *balancing not only the different graph properties but also varied structural graph information*. The learned robust graph is fed into GNNs and jointly leaned with GNN parameters. Extensive empirical results show that our proposed TGNN framework consistently improves the robustness of the learned graph and significantly boosts the performance of node classification task for GNNs against adversarial graphs.

2 Related Works

2.1 Graph Neural Networks and Adversarial Attacks on Graphs

GNNs have achieved great success in solving machine learning problems on graph data in recent years. Existing GNNs can be divided into two categories: spectral-based methods where graph spectral theory (Bruna et al., 2014; Defferrard et al., 2016; Kipf & Welling, 2017; Wu et al., 2019a; Chen et al., 2018) was applied to learn node representations; and spatial-based GNNs based on local-dependence assumption of graph-structured data that graph convolution is defined in the spatial domain as aggregating and transforming local information (Gilmer et al., 2017; Hamilton et al., 2017; Velickovic et al., 2018; Atwood & Towsley, 2016).

Extensive studies have demonstrated that deep learning models are vulnerable to adversarial attacks (Goodfellow et al., 2015; Kurakin et al., 2017a;b; Rakhsha et al., 2020). Attackers add unnoticeable perturbations to the input for altering the output of the deep learning model. GNNs are also fragile to adversarial attacks (Bojchevski & Günnemann, 2019; Dai et al., 2018; Jin et al., 2020a; Liu et al., 2019; Ma et al., 2019; Wu et al., 2019b; Zügner et al., 2018; Zügner & Günnemann, 2019b). It is more challenging to investigate adversarial attack on graphs due to the discrete graph and dependent nodes compared with independent samples or grid data. Nettack (Zügner et al., 2018) preserves degree distribution and constrains feature co-occurrence to generate unnoticeable perturbations to poison GNNs in training time. Reinforcement learning was employed in RL-S2V (Dai et al., 2018) to generate adversarial attacks in testing-time. Both of the above two methods can only degrade the performance of GNN on target nodes. To attack the graph globally, metattack (Zügner & Günnemann, 2019b) is proposed to generate non-targeted poisoning attacks based on meta-learning.

2.2 Defense on Graphs

The studies for improving the robustness of GNNs have received increasing attention recently (Tang et al., 2019; Wu et al., 2019b; Zhu et al., 2019; Zügner & Günnemann, 2019a; Bojchevski et al., 2020; Geisler et al., 2020). In RGCN (Zhu et al., 2019), Gaussian distribution is adopted in hidden layers to absorb the effects of adversarial attacks into the variances. A meta-optimization way is proposed in PA-GNN (Tang et al., 2019) to learn attention scores for robust GNNs with the help of supervision knowledge from clean graphs. Transfer learning (Tang et al., 2020) is leveraged to improve the robustness of GNNs against poisoning attack with additional graph data from the similar domain during training. It is observed in Wu et al. (2019b) that attackers tend to connect to nodes with different features occurrence, GNN-Jaccard is proposed to remove the links between dissimilar nodes to identify the manipulated edges. GNN-SVD (Entezari et al., 2020) is proposed to preprocess the graph with its low-rank approximations to remove noisy information through an SVD decomposition. Pro-GNN (Jin et al., 2020b) proposes three different regularizers which correspondingly regularize the graph to be sparse, low-rank and smooth the features of connected nodes. In Zhang & Zitnik (2020), GNNGuard is proposed to quantify the relationship between the graph and node features,

and assigns less weight to suspicious edges, which mitigates negative effects of the adversarial perturbations.

Different from the aforementioned defense methods, we aim to aggregates predefined robust adversarial graphs to form a powerful regularization, which can be extended to different GNNs straightforwardly to enhance the robustness of GNNs.

3 METHODOLOGY

Before presenting the proposed method, we first introduce the problem statement for node classification with GNN, including some basic notations and concepts. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{X})$ be a graph, where $\mathcal{V}$ represents the set of N nodes $\{v_1, v_2, \cdots, v_N\}$, $\mathcal{E}$ represents the set of M edges $\{(i_1, j_1), (i_2, j_2), \cdots, (i_M, j_M)\}$, and $\mathbf{X} \in \mathbb{R}^{N \times d}$ denotes the node feature matrix composed of the feature vector $\mathbf{x}_i$ of the node v_i. The edges $\mathcal{E}$ describe the graph by the pair-wise relations between nodes, which can also be represented by an adjacency matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$, one of the most popular representation for graph, where $\mathbf{A}_{ij} = 1 \iff (i, j) \in \mathcal{E}$ (otherwise $A_{ij} = 0$). In this work, we alternately use the concept of adjacency matrix and graph. Thus, a graph can also be denoted as $\mathcal{G} = (\mathbf{A}, \mathbf{X})$.

In this work, we focus on the common semi-supervised node classification setting for graph representation. Given a single (attributed) graph $\mathcal{G} = (\mathbf{A}, \mathbf{X})$, the part of nodes $\mathcal{V}_P = \{v_1, v_2, \cdots, v_P\} \subseteq \mathcal{V}$ are associated with the corresponding labels $\mathcal{Y}_P = \{y_1, y_2, \cdots, y_P\}$, where y_i is the class label (*e.g.*, the topic of a paper in a citation graph) of v_i, the goal of node classification in GNN is to learn a neural network representing a function $f_\theta : \mathcal{V}_P \to \mathcal{Y}_P$ which maps the nodes to the corresponding labels so that f_θ can be generalized to predict the labels of unlabeled nodes (*i.e.*, nodes not in $\mathcal{V}_P$), where θ is the parameters of f_θ. With $\mathcal{L}_{CE}(\cdot, \cdot)$ denoting the cross entropy loss, the objective function of node classification task in GNN can be formulated as:

$$\min_{\theta} \mathcal{L}_{cls}(\theta, \mathbf{A}, \mathbf{X}, \mathcal{Y}_P) = \frac{1}{P} \sum_{i=1}^{P} \mathcal{L}_{CE}\left(y_i, f_\theta(\mathbf{X}, \mathbf{A})_i\right), \tag{1}$$

where $f_\theta(\mathbf{X}, \mathbf{A})_i$ is the prediction of node v_i. Note that our proposed method can be extended to different GNNs straightforwardly, in this work, we focus on Graph Convolution Network (GCN) proposed in Kipf & Welling (2017) among numbers of different GNNs. Specifically, we adopt a two-layer GCN with layers $(f_{\theta_1}, f_{\theta_2})$ and the corresponding parameters $(\theta_1, \theta_2) = (\mathbf{W}_1, \mathbf{W}_2)$ can be denoted as:

$$f_\theta(\mathbf{X}, \mathbf{A}) = softmax(f_{\theta_2}(f_{\theta_1}(\mathbf{X}, \mathbf{A}), \mathbf{A})) = softmax(\hat{\mathbf{A}}\sigma(\hat{\mathbf{A}}\mathbf{X}\mathbf{W}_1)\mathbf{W_2}), \tag{2}$$

where σ is the activation function and $\hat{\mathbf{A}} = \hat{\mathbf{D}}^{-\frac{1}{2}}(\mathbf{A} + \mathbf{I})\hat{\mathbf{D}}^{-\frac{1}{2}}$ represents diagonal normalized $\mathbf{A} + \mathbf{I}$. Here, self-loop $\mathbf{I}$ is added to $\mathbf{A}$ and $\hat{\mathbf{D}}$ is the diagonal degree matrix of $\mathbf{A} + \mathbf{I}$ such that $\hat{\mathbf{D}}_{ii} = \mathbf{1} + \sum_j \mathbf{A}_{ij}$.

3.1 THE PROPOSED FRAMEWORK

Although existing GNNs achieve the state of the art performance in node classification task, they are still vulnerable to edge perturbations generated by adversarial attacks. We refer to the attacked graph with carefully designed edge perturbations degrading GNNs performance as *adversarial graphs* and the graphs which are robust for GNNs to perform accurate node classification as *robust graphs*. Learning robust graphs from adversarial graphs can vaccinate the GNNs from the poisoned graph data for stable performance in node classification task. With the statements at the beginning of this section, the goal of learning robust graphs can be formally stated as:

Given $\mathcal{G}$ poisoned by adversarial graph $\tilde{\mathbf{A}}$, the unperturbed feature $\mathbf{X}$ and partial node labels $\mathcal{Y}_P$, we aim to learn GNN parameters θ and robust graph $\bar{\mathbf{A}}$ replacing $\tilde{\mathbf{A}}$ to enhance robustness of GNNs and improve node classification performance for unlabeled nodes.

To generate robust graph, we propose a Tensor-based robust graph aggregation framework for Graph Neural Networks (TGNN) shown in Figure 1. TGNN aggregates K predefined robust adversarial

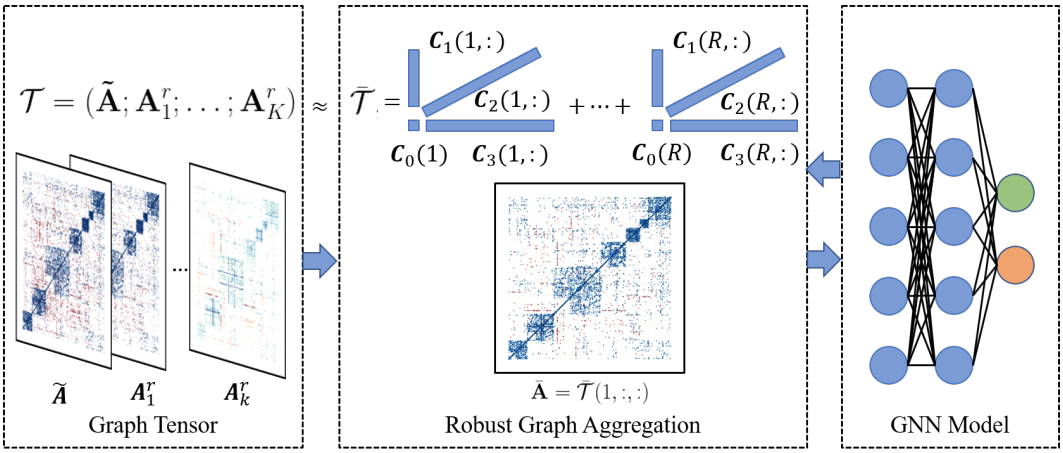


Figure 1: Illustration of our proposed TGNN framework. We first form the graph tensor $\mathcal{T}$ by combining the adversarial graph $\tilde{\mathbf{A}}$ (adjacency matrix is visualized that the blue points are edges in the original graph, while the red ones are perturbed edges) and K predefined robust graphs $\{\mathbf{A}_k^r|_{k=1}^K\}$, then learn robust graph by tensor-based robust graph aggregation. Finally, the learned robust graph $\bar{\mathbf{A}}$ will be fed into the GNN model and jointly learned with GNN model parameters.

graphs $\mathbf{A}^r = \{\mathbf{A}_k^r|_{k=1}^K\}$ to learn the robust graph $\bar{\mathbf{A}}$ from adversarial graph $\tilde{\mathbf{A}}$. In particular, TGNN first calculates K predefined robust graphs $\mathbf{A}^r$ and combines them with the adversarial graph $\tilde{\mathbf{A}}$ to form a tensor structure, then decomposes the graph tensor into several core tensors $\mathcal{C}$ and reconstructs the tensor via low-rank approximation. By preserving the low-rank tensor representation for the graphs, TGNN generates the robust graph $\bar{\mathbf{A}}$ and maintains the multi-linear relation between the learned robust graph $\bar{\mathbf{A}}$ and predefined robust graphs $\mathbf{A}^r$. Then the learned robust graph $\bar{\mathbf{A}}$ will be fed into the GNN. TGNN simultaneously updates both the GNN parameters θ and $\bar{\mathbf{A}}$ by updating learnable core tensors $\mathcal{C}$ during the training process. In this way, TGNN can achieve a more robust graph and better node classification performance. In the following sections, we will discuss how to aggregate robust graphs with TGNN, the predefined graphs, and the results of the generated robust graph of TGNN.

3.2 Tensor-Based Aggregation for Robust Graphs

A tensor, also called multi-dimensional array, is denoted by Euler script letter, *e.g.*, $\mathcal{T}$, where each dimension of the tensor is called a mode. In TGNN, we formulate a batch of adjacency matrices as a three-mode tensor. Specifically, given perturbed adjacency matrix $\tilde{\mathbf{A}}$ and K robust adjacency matrices $\mathbf{A}_1^r, \cdots, \mathbf{A}_K^r$, we stack these adjacency matrices as a graph aggregation tensor $\mathcal{T} \in \mathbb{R}^{(K+1)\times N\times N}$, where $\mathcal{T}(1,:,:) = \tilde{\mathbf{A}}$ and $\mathcal{T}(k+1,:,:) = \mathbf{A}_k^r$ for $1 \leq k \leq K$. To achieve a low-rank approximation of the graph aggregation tensor $\mathcal{T}$, we perform tensor approximation by decomposing $\mathcal{T}$ into several smaller core tensors following specific formulations (Harshman et al., 1970; Tucker, 1966; Oseledets, 2011; Zhao et al., 2016), and then apply multilinear contraction operations on these core tensors to reconstruct approximated tensor $\bar{\mathcal{T}}$. Low-rank approximation can discard high-rank noises in the graph to improve the performance of GNNs against adversarial graphs (Jin et al., 2020b; Entezari et al., 2020). Also, graphs in the real-world are often naturally low-rank because nodes within a community or the same class are apt to have similar neighbors or similar node features. Hence, low-rank approximation is a potential way to extract the robust graph from the adversarial graphs. Tensor decomposition is a powerful tool to compress the high-dimensional data into low-rank representation (Cichocki et al., 2015; Sidiropoulos et al., 2017). In this paper, we attempt to decompose the graph aggregation tensor $\mathcal{T}$ with Canonical Polyadic (CP) (Harshman et al., 1970), Tucker Tucker (1966) and Tensor-Train (Oseledets, 2011) formulations, respectively, the comparison results with three different decomposition methods are reported in Supplementary. To describe the process of learning robust graph, we use CP decomposition method as an example

which decomposes a three mode tensor $\mathcal{T} \in \mathbb{R}^{(K+1)\times N\times N}$ into:

$$\mathcal{T} \approx \bar{\mathcal{T}} = \Sigma_{t=1}^{R} \mathbf{C}_0(t)\mathbf{C}_1(t,:) \circ \mathbf{C}_2(t,:) \circ \mathbf{C}_3(t,:), \tag{3}$$

where $\mathbf{C}_1 \in \mathbb{R}^{R\times(K+1)}$, $\mathbf{C}_2 \in \mathbb{R}^{R\times N}$, $\mathbf{C}_3 \in \mathbb{R}^{R\times N}$ and $\mathbf{C}_0 \in \mathbb{R}^{R}$ are trainable core tensors (matrix/vectors, $\mathbf{C}_0$ is used as normalization vector). R represents the value of rank which is relatively small as compared to the node number N and '$\circ$' denotes outer product (*i.e.*, $\bar{\mathcal{T}}(i,j,k) = \sum_{t=1}^{R} \mathbf{C}_0(t)\mathbf{C}_1(t,i)\mathbf{C}_2(t,j)\mathbf{C}_3(t,k)$). After the tensor graph approximation, TGNN takes the first slice of the first-mode in the approximated graph aggregation tensor, *i.e.*, $\bar{\mathcal{T}}(1,:,:)$ as the learned robust graph $\bar{\mathbf{A}}$ to be the input for GNNs. With $\mathcal{C}$ representing the sets of all core tensors, the objective function for TGNN can be defined as:

$$\min_{\theta,\mathcal{C}} \mathcal{L} = \mathcal{L}_{cls}(\theta, \bar{\mathbf{A}}, \mathbf{X}, \mathcal{Y}_P) + \lambda||\bar{\mathcal{T}} - \mathcal{T}||_F^2, \tag{4}$$

where $\mathcal{L}_{cls}$ is the classification loss from Eqn. (1) and λ is a hyper-parameter to balance the tensor approximation loss and the classification loss.

The decomposed representation of $\mathcal{T}$ in Eqn. (3) maps the graphs to a shared core tensor space in which different robust graphs are linearly connected to form common structure patterns by compressing and fusing the structure representations of each graphs. The learned common structure patterns are beneficial to TGNN to learn a more robust graph and we will show this in the next section.

3.3 PREDEFINED ROBUST GRAPHS

Here we briefly introduce three simple methods to generate robust graphs from adversarial graphs which will be adopted as predefined robust graphs by our proposed robust graph aggregation method:

- **PRUNE:** Adversarial graphs tend to add edges to unconnected nodes with dissimilar features (Wu et al., 2019b; Jin et al., 2020b). It is natural to prune the graph to get robust graph by eliminating edges that connect nodes with less similarity of features. Specifically, the pruned adjacency matrix can be represented by:

$$\mathbf{A}^r_{PRUNE}(i,j) = \begin{cases} 1 & \text{if } \tilde{\mathbf{A}}(\mathrm{i},\mathrm{j}) = 1 \text{ and } \mathrm{s}(\mathbf{x}_\mathrm{i}, \mathbf{x}_\mathrm{j}) > \tau, \\ 0 & \text{otherwise.} \end{cases} \tag{5}$$

  Here $s(\cdot,\cdot)$ denotes similarity function that calculates the feature similarity of two nodes (can be Jaccard similarity for binary features or cosine similarity for real-valued features), and τ is a threshold parameter.

- **KNN:** Since the edges are perturbed and we are not able fully determine which edge is perturbed, we can completely ignore the edge information and construct the graph only based on the node features. Similar to **PRUNE**, we can build robust graph by calculating the k-nearest neighbors based on feature similarities between nodes:

$$\mathbf{A}^r_{KNN}(i,j) = \begin{cases} 1 & \text{if } \mathrm{j} \in \mathrm{top}(\mathrm{i},\mathrm{k}), \\ 0 & \text{otherwise.} \end{cases} \tag{6}$$

  Here $top(i,k)$ denotes set of the top k-th nodes that have large feature similarity to node i.

- **SVD:** Existing works (Jin et al., 2020b; Entezari et al., 2020) found that adversarial perturbations tend to add edges that connect nodes of different communities to fool GNNs, which enlarge the singular values of the adjacency matrix. Hence, with $\mathbf{u}_i$, σ_i, and $\mathbf{v}_i$ to be the i-th singular value/vectors of $\tilde{\mathbf{A}}$, we can generate robust graph by calculating low-rank approximations of the adjacency matrix:

$$\mathbf{A}^r_{SVD} = \Sigma_{i=1}^{r} \mathbf{u}_i \sigma_i \mathbf{v}_i. \tag{7}$$

All the aforementioned methods are preprocessing methods that cost little time to run but still efficient to generate robust graphs to defend against adversarial edges. However, these methods are designed for single graph property so that they may fail to counteract complex global attacks. In Figure 2, we visualize the adjacency matrix of different predefined robust graphs and those learned by TGNN. As we can see from the figure, $\mathbf{A}^r_{SVD}$ preserve the graph with more graph spectral information that some of the perturbed edges are reduced, but a large number of original edges are also weakened.

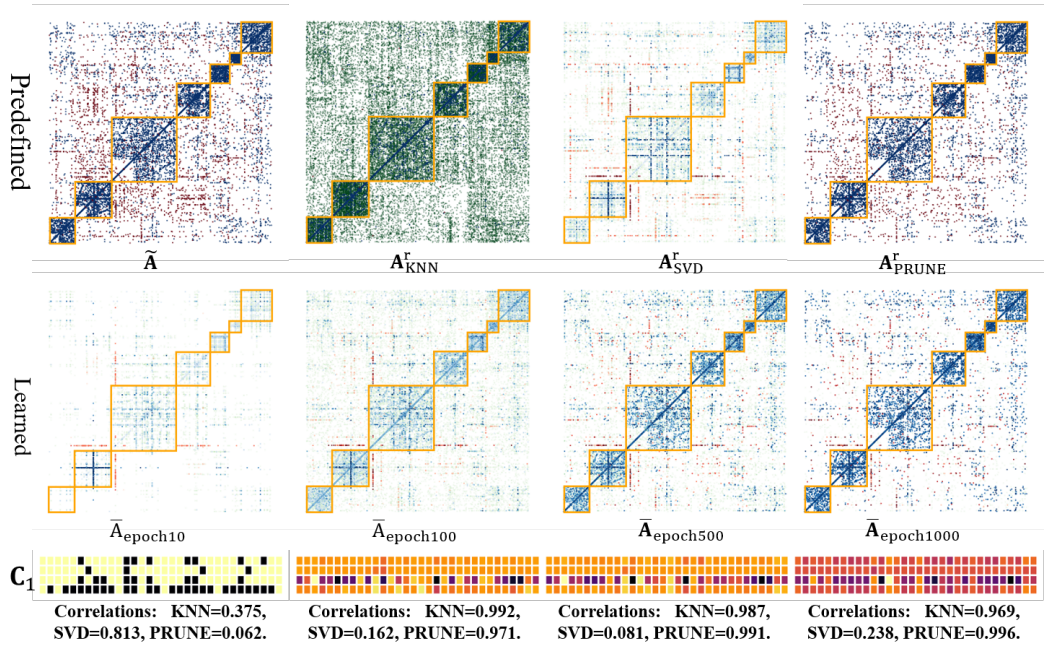


Figure 2: Visualization of the adversarial graph $\tilde{\mathbf{A}}$ under metattack (Zügner & Günnemann, 2019b) for Cora (McCallum et al., 2000) dataset, three predefined robust graphs $\mathbf{A}^r_{KNN}$, $\mathbf{A}^r_{SVD}$ and $\mathbf{A}^r_{PRUNE}$ and robust graph $\bar{\mathbf{A}}$ learned from TGNN after different epochs of training. Nodes are sorted by class labels. Blue dots represent edges that are in the original graph $\mathbf{A}$, red dots represent edges that are added by the adversarial graph (*i.e.*, $\tilde{\mathbf{A}} - \mathbf{A}$), and green dots are the edges not included in both $\tilde{\mathbf{A}}$ and $\mathbf{A}$. Class communities are boxed in orange. Core tensor $\mathbf{C}_1$ is visualized and the correlations between the learned robust graph $\bar{\mathbf{A}}$ and predefined robust graph $\mathbf{A}^r$ are shown.

$\mathbf{A}^r_{KNN}$ link the nodes with similar features, which contains many additional edges (green dots), and almost removes all the perturbed edges but also drops most of the original edges. For $\mathbf{A}^r_{PRUNE}$, it eliminates some perturbed edges with low feature similarity but also a small number of original edges. For the robust graph $\bar{\mathbf{A}}$ learned from TGNN, at the beginning of the training process, similar to $\mathbf{A}^r_{SVD}$, the low-rank presentation makes the learned graph $\bar{\mathbf{A}}$ pay more attention to spectral information, while gradually, it considers more about the feature similarities and reconstructs the input graph. Then during the training, the learned graph $\bar{\mathbf{A}}$ is fuzzier and more edges that are not in the original input graph are added. Then at the rest of the training process, $\bar{\mathbf{A}}$ is sparser and closer to the original graph. Compared with $\mathbf{A}^r_{PRUNE}$, $\bar{\mathbf{A}}$ drops more perturbed edges. In Figure 2, we also visualize the core tensor $\mathbf{C}_1$ defined in Eqn. (3) which reveals the relationship between the learned graph and the predefined graphs. Here we choose the rank as 32 so that $\mathbf{C}_1 \in \mathbb{R}^{32\times 4}$ and show the heatmap of values in $\mathbf{C}_1$. From the top to the bottom rows, the vectors are representing the weights for the learned graph $\bar{\mathbf{A}}$, $\mathbf{A}^r_{KNN}$, $\mathbf{A}^r_{SVD}$ and $\mathbf{A}^r_{PRUNE}$. Correlations are calculated by the cosine similarity between different weights, *e.g.*, the correlation between $\bar{\mathbf{A}}$ and $\mathbf{A}^r_{KNN}$ is calculated by the cosine similarity between $\mathbf{C}_1(:,1)$ and $\mathbf{C}_1(:,2)$. From the results, we can see that the correlation between $\bar{\mathbf{A}}$ and $\mathbf{A}^r_{SVD}$ is initially high in the training process, degrades in the middle stage and grows back slightly at the end of the training. For $\mathbf{A}^r_{KNN}$ and $\mathbf{A}^r_{PRUNE}$, the correlations are low at the beginning, then grow quickly during the training, and finally at a high level. These results also reveal how TGNN aggregates the predefined robust graphs and learn $\bar{\mathbf{A}}$.

Also, these predefined methods may suffer from the simple two-stage preprocessing without end-to-end training. Combined with our proposed tensor-based robust graph aggregation method, we can aggregate the properties of these robust graphs and jointly learn the robust graph and the GNN. In Section 4.3, we will discuss the performance of our proposed TGNN compared with these predefined robust graphs for node classification performance. In Appendix A.6, we visualize the graph for nettack and random attack. In Appendix A.5, we compare the graph visualization between TGNN and a state-of-the-art method Pro-GNN.

Table 1: Defense performance against non-target (metattack) attacks.

Datasets	Rate (%)	GCN	GAT	RGCN	GCN-Jaccard	GCN-SVD	Pro-GNN	GNNGuard	TGNN
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	82.38±1.23	**84.11±0.59**
	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	78.96±0.55	**82.84±0.35**
	10	70.39±1.28	75.61±0.59	72.22±0.38	75.16±0.76	71.47±0.83	79.03±0.59	72.17±2.01	**81.12±0.86**
	15	65.10±0.71	69.78±1.28	66.82±0.39	71.03±0.64	66.69±1.18	76.40±1.27	68.71±1.87	**80.28±0.74**
	20	59.56±2.72	59.94±0.92	59.27±0.37	65.71±0.89	58.94±1.13	73.32±1.56	58.48±1.59	**78.12±1.50**
	25	47.53±1.96	54.78±0.74	50.51±0.78	60.82±1.08	52.06±1.19	69.72±1.69	53.19±0.79	**74.14±2.00**
Citeseer	0	71.96±0.55	73.26±0.83	71.20±0.83	72.10±0.63	70.65±0.32	73.28±0.69	71.15±0.84	**74.01±0.60**
	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	70.78±0.68	**73.92±0.64**
	10	67.55±0.89	70.63±0.48	67.71±0.30	69.54±0.56	68.87±0.62	72.51±0.75	66.04±0.81	**73.75±0.79**
	15	64.52±1.11	69.02±1.09	65.69±0.37	65.95±0.94	63.26±0.96	72.03±1.11	64.29±1.29	**73.52±1.00**
	20	62.03±3.49	61.04±1.52	62.49±1.22	59.30±1.40	58.55±1.09	70.02±2.28	58.80±2.18	**73.38±1.25**
	25	56.94±2.09	61.85±1.12	55.35±0.66	59.89±1.47	57.18±1.87	68.95±2.78	56.07±2.65	**73.00±2.01**
Polblogs	0	**95.69±0.38**	95.35±0.20	95.22±0.14	95.40±0.15	95.31±0.18	93.20±0.64	95.56±0.21	95.20±0.47
	5	73.07±0.80	83.69±1.45	74.34±0.19	82.70±2.26	89.09±0.22	93.29±0.18	85.84±0.92	**93.71±0.53**
	10	70.72±1.13	76.32±0.85	71.04±0.34	77.26±2.02	81.24±0.49	89.42±1.09	80.65±0.86	**91.14±2.55**
	15	64.96±1.91	68.80±1.14	67.28±0.38	73.53±2.68	68.10±3.73	86.04±2.21	75.71±0.69	**86.80±1.69**
	20	51.27±1.23	51.50±1.63	59.89±0.34	70.14±2.81	57.33±3.15	79.56±5.68	70.32±1.95	**82.25±2.13**
	25	49.23±1.36	51.19±1.49	56.02±0.56	65.70±2.70	48.66±9.93	63.18±4.40	65.07±3.16	**73.72±2.70**
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	85.03±0.10	**87.80±0.11**
	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	82.19±0.23	**87.78±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	78.28±0.36	**87.62±0.12**
	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	74.63±0.24	**87.40±0.11**
	20	77.35±0.19	68.21±0.96	71.18±0.31	83.88±0.05	83.01±0.22	87.15±0.15	72.13±0.34	**87.34±0.17**
	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	69.45±0.40	**86.93±0.21**

4 EXPERIMENTS

4.1 EXPERIMENTAL SETTING

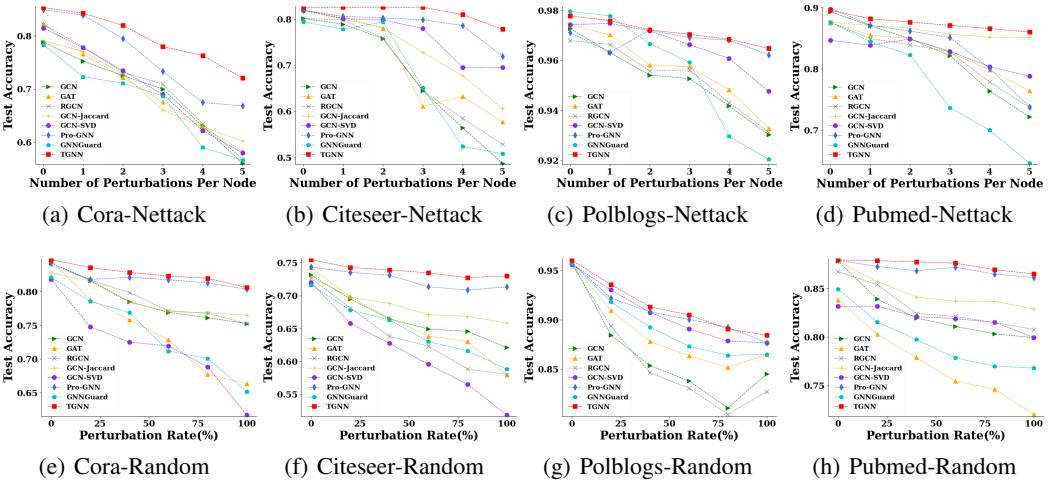
Datasets Following Zügner et al. (2018); Zügner & Günnemann (2019b); Jin et al. (2020b), we conduct experiment on four benchmark datasets,including three citation graphs, *i.e.*, Cora (McCallum et al., 2000), Citeseer (Giles et al., 1998; Sen et al., 2008) and Pubmed (Sen et al., 2008), and one blog graph Polblogs (Adamic & Glance, 2005). We summarize the statistics of the datasets in Appendix A.2.

Baselines Following the adversarial attack repository DeepRobust (Li et al., 2020), we report the results of the state-of-the-art GNN and defense models for comparison. GCN (Kipf & Welling, 2017) and GAT (Velickovic et al., 2018) are adopted as baselines of GNNs, while RGCN (Zhu et al., 2019), GCN-Jaccard (Wu et al., 2019b), GCN-SVD (Entezari et al., 2020), Pro-GNN (Jin et al., 2020b), and GNNGuard (Zhang & Zitnik, 2020) are chosen as baselines of defense models. Please note that GCN-Jaccard and GCN-SVD are using the same approaches as **PRUNE** and **SVD** in Section 3.3. Specific description to these baselines can be found in Appendix A.3.

Parameter Settings For each graph, all nodes are randomly split into 10% of training nodes, 10% of validation nodes, and 80% of testing nodes. We report the average performance of ten runs for each experiment. The hyper-parameters of all the models are tuned by observing the loss and accuracy on validation set. We use Adam (Kingma & Ba, 2015) optimizer with learning rate as 0.01. λ is 100 in Eqn. (4). We use PyTorch (Paszke et al., 2019) (TensorLy (Kossaifi et al., 2019) for the tensor decomposition parts) to implement TGNN. We choose to use CP decomposition and the rank is selected based on cross validation. In Appendix A.7, we will discuss how different formats of tensor decomposition affect TGNN. All the results from the state-of-the-art methods are reported with the best parameter settings.

4.2 DEFENSE PERFORMANCE

To evaluate the effectiveness of TGNN, we conduct node classification experiments under three types of adversarial attacks, *i.e.*, non-targeted attack, targeted attack and random attack. Targeted attack aims to fool GNNs with generated attacks on specific nodes, and we adopt the state-of-the-art targeted attack nettack (Zügner et al., 2018) in this paper. Different from targeted attack, non-targeted attack can degrade the overall performance of GNNs on the whole graph and we choose metattack (Zügner



(a) Cora-Nettack (b) Citeseer-Nettack (c) Polblogs-Nettack (d) Pubmed-Nettack

(e) Cora-Random (f) Citeseer-Random (g) Polblogs-Random (h) Pubmed-Random

Figure 3: Results of different models under nettack and random attack.

& Günnemann, 2019b) for the non-targeted attack evaluation. Random attack randomly adds noise to the original graph by injecting fake edges into the graph.

Against Non-Targeted Adversarial Attacks: Firstly, we evaluate the node classification performance of different methods against non-targeted adversarial attack. We adopt metattack Zügner & Günnemann (2019b) and implement it with its default parameter settings in original paper. We apply the most destructive attack variant of metattack, Meta-Self, to Cora, Citeseer and Polblogs. For Pubmed, A-Meta-Self as the approximate version of 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%. The average accuracy with standard deviation is reported in Table 1, in which the best performance is highlighted in bold. From the table, we can see that TGNN achieves the highest node classification accuracy under different perturbation rates, which demonstrates that TGNN can generate robust graph against non-targeted adversarial attack. For instance, TGNN improves GCN over 20.7% at 5% perturbation rate on Polblogs. Under the 25% perturbation rate, vanilla GCN performs very poorly and TGNN improves GCN by 27% (*resp.,* 16%, 23% and 11%) on the dataset Cora (*resp.,* Citeseer, Polblogs, and Pubmed), which indicates TGNN outperforms other baselines by a larger margin even under large perturbation. In addition, TGNN may also perform better when the graph is not attacked (*i.e.*, perturbation rate is 0). That is because there may be a small amount of noise edges in the original graph which are mitigated by TGNN. For the other defense methods, although GCN-SVD also employs SVD to get low-rank approximation of the graph, the performance of GCN-SVD drops rapidly. Similarly, GCN-Jaccard does not perform as well as Pro-GNN under different perturbation rates. This is because simply pruning the perturbed graph by feature similarities once cannot extract the robust graph from the carefully-crafted adversarial noises. TGNN also outperforms Pro-GNN that applies regularization based on low-rank, sparse and feature similarity properties to learn robust graph, which shows the regularization is not able to perfectly balance different graph properties to enhance robustness. On the contrary, TGNN adopts tensor-based aggregation method to update the graph and GNNs simultaneously to learn better graph and more robust GNNs.

Against Targeted Adversarial Attack: For the targeted-attack method nettack (Zügner et al., 2018), we use the default parameter settings in the original implementation. Following Zhu et al. (2019), the number of perturbations added on every targeted node is varied from 1 to 5 with a step size of 1. Nodes with degree larger than 10 as target nodes are selected as the test set. 10% of target nodes are sampled on Pubmed dataset to save running time, while all the target nodes are used on other datasets. The node classification accuracy on target nodes is shown in Figure 3 (a)-(d), in which we can observe that TGNN achieves better performance than other baselines with the increase of the number of perturbation in most cases. For instance, at 5 perturbation per targeted node, TGNN improves vanilla GCN by 16% (*resp.,* 29%, 6%, 6%) on Cora (*resp.,* Citeseer, Polblogs, Pubmed) dataset. It demonstrates that TGNN can also resist the targeted adversarial attack.

Table 2: Classification performance for different settings of baselines and our proposed TGNN under metattack attack on Cora dataset. Note that there are two settings, "No $\mathbf{A}$"$= 62.12 \pm 1.55$ and "KNN"$= 71.94 \pm 1.08$, of which the results do not change with the perturbation rates.

Rate (%)	GCN	PRUNE	SVD	TGNN-P	TGNN-K	TGNN-S	TGNN-N	Ensemble	Two-Stage	TGNN-ALL
0	83.50±0.44	82.05±0.51	80.63±0.45	83.57±0.46	74.91±1.15	81.64±0.67	79.95±0.52	82.42±0.41	81.09±0.36	**84.11±0.59**
5	76.55±0.79	79.13±0.59	78.39±0.54	81.51±0.55	73.80±1.07	80.22±0.54	78.37±0.91	80.78±0.42	80.87±0.29	**82.84±0.35**
10	70.39±1.28	75.16±0.76	71.47±0.83	78.15±1.01	73.52±1.38	75.38±1.96	75.37±1.12	78.60±0.63	78.92±0.38	**81.12±0.86**
15	65.10±0.71	71.03±0.64	66.69±1.18	76.01±1.36	73.77±1.17	74.14±1.49	74.04±2.00	75.23±1.02	77.04±0.70	**80.28±0.74**
20	59.56±2.72	65.71±0.89	58.94±1.13	72.23±2.02	73.43±1.34	65.86±3.41	69.81±2.89	72.11±2.06	75.03±0.77	**78.12±1.50**
25	47.53±1.96	60.82±1.08	52.06±1.19	64.43±2.90	73.03±1.63	57.77±2.87	62.33±4.76	71.68±1.01	72.11±0.54	**74.14±2.00**

Against Random Attack: By varying the ratios of random noises from 0% to 100% with a step size of 20%, we compare the performance of TGNN with baselines and state-of-the-art methods. Node classification accuracy results are reported in Figure 3 (e)-(h). It can be observed that TGNN is able to resist random attack and outperforms all other baselines under random attack.

4.3 Ablation Studies

We design different ablated methods to demonstrate the effectiveness of TGNN defending against metattack attack and report classification performance on Cora (McCallum et al., 2000) dataset in Table 2, in which our full method is named as "TGNN-ALL". Please note that the setting "No $\mathbf{A}$" means running GCN without graph information, *i.e.*, remove $\hat{\mathbf{A}}$ in Eqn. (2). We list it here as a reference. Both "No $\mathbf{A}$" and "KNN" are not using the graph, which means their results are unchanged with the perturbation rate, so we omit the results and put them in the caption of Table 2. We also put the baseline results of "GCN" for the sake of comparison.

Variations of Predefined Robust Graphs: Firstly we report the results of three predefined adversarial graphs in Section 3.3, which are referred to as "PRUNE", "KNN", and "SVD" in Table 2. Then, we use TGNN to aggregate a predefined adversarial graph "PRUNE" (*resp.,* "KNN", "SVD"), which is referred to as "TGNN-P" (*resp.,* TGNN-K, TGNN-S). From the comparison result between "PRUNE" (*resp.,* "SVD") and "TGNN-P" (*resp.,* "TGNN-S"), we can see that each predefined adversarial graph with the help of our TGNN can consistently improve classification performance. We also remove all predefined adversarial graphs from our method, which is named as "TGNN-N" in Table 2. We find that the performance is comparable with predefined adversarial graphs "PRUNE" and "SVD", which demonstrates the rationality of TGNN.

Compared to Ensembles: To evaluate the effectiveness of the tensor decomposition in TGNN, we simply ensemble predefined adversarial graphs, which is referred to as "Ensemble", the result is better than "PRUNE" and "SVD", which indicates that assembling predefined adversarial graphs by learning a GNN for each one and summing them up is one way to improve the robustness of GNNs. However, the performance is worse than our full method "TGNN-ALL", which demonstrates the effectiveness of the designed aggregation strategy of TGNN.

End-to-End Training: Corroborate the superiority of end-to-end training strategy, we design a two-stage variant of TGNN where the robust graph is firstly obtained, and then using it to train a GNN, which is recorded as "Two-Stage" in Table 2. It can be seen that the performance is worse than our full method "TGNN-ALL", which demonstrates the advantage of the jointly learning strategy.

5 Conclusion

In this paper, we introduce TGNN, a framework for defending GNN against poisoning attacks. TGNN combines predefined robust graph structures and formulates them with the adversarial graph structure as a graph structure tensor, then applies low-rank tensor approximation to generate a more robust graph structure. The low-rank representation of the robust graph structures preserve the common patterns of robust structures, balance different graph properties, and enhance the robustness of the generated graph structure. Experiments on four datasets show that TGNN outperforms existing GNN defense algorithms, especially when the perturbation rate is high.

6 ETHICS STATEMENT

TGNN is a framework for defending GNN against poisoning attacks, combining predefined robust graphs and the adversarial graph as a graph tensor, then applying low-rank tensor approximation to generate a more robust graph. We have shown that TGNN outperforms existing GNN defense methods, especially when the perturbation rate is high. TGNN can be used in a wide range of applications integrated with GNNs, *e.g.*, recommendation system, drug discovery, and social/financial networks. Positive impacts of TGNN include helping users avoid potential losses caused by malicious attacks, predict false diagnoses for knowledge systems, and improve fairness and explainability of GNNs. One potentially negative impact is to design a more powerful attacking algorithm based on TGNN. TGNN aggregates predefined robust graph methods, we have shown that TGNN can always improve the classification accuracy based on the provided predefined methods. Hence, attackers may develop more powerful attacking algorithms based on TGNN (*i.e.*, adaptive attacks).

In this paper, all the datasets are downloaded from public academic resources (please refer to Appendix A.2).

7 REPRODUCIBILITY STATEMENT

We provide the source code in the supplementary materials to reproduce the results. We also provide extensive analyses for the settings and hyperparameters of TGNN, which can be found in Appendix.

REFERENCES

Lada A Adamic and Natalie Glance. The political blogosphere and the 2004 us election: divided they blog. In *Proceedings of the 3rd international workshop on Link discovery*, 2005.

Saba A. Al-Sayouri, Ekta Gujral, Danai Koutra, Evangelos E. Papalexakis, and Sarah S. Lam. t-PINE: tensor-based predictable and interpretable node embeddings. *Soc. Netw. Anal. Min.*, 10(1):46, 2020.

James Atwood and Don Towsley. Diffusion-convolutional neural networks. In *NeurIPS*, 2016.

Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinícius Flores Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Çaglar Gülçehre, H. Francis Song, Andrew J. Ballard, Justin Gilmer, George E. Dahl, Ashish Vaswani, Kelsey R. Allen, Charles Nash, Victoria Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Pushmeet Kohli, Matthew Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. Relational inductive biases, deep learning, and graph networks. *CoRR*, 2018.

Srinadh Bhojanapalli and Sujay Sanghavi. A new sampling technique for tensors. *CoRR*, abs/1502.05023, 2015.

Aleksandar Bojchevski and Stephan Günnemann. Adversarial attacks on node embeddings via graph poisoning. In *ICML*, 2019.

Aleksandar Bojchevski, Johannes Klicpera, and Stephan Günnemann. Efficient robustness certificates for discrete data: Sparsity-aware randomized smoothing for graphs, images and more. In *ICML*, 2020.

Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. In *ICLR*, 2014.

J Douglas Carroll and Jih-Jie Chang. Analysis of individual differences in multidimensional scaling via an n-way generalization of "eckart-young" decomposition. *Psychometrika*, 35(3):283–319, 1970.

Jie Chen, Tengfei Ma, and Cao Xiao. FastGCN: Fast learning with graph convolutional networks via importance sampling. In *ICLR*, 2018.

Dehua Cheng, Richard Peng, Yan Liu, and Ioakeim Perros. SPALS: fast alternating least squares via implicit leverage scores sampling. In *NeurIPS*, pp. 721–729, 2016.

Andrzej Cichocki, Danilo P. Mandic, Lieven De Lathauwer, Guoxu Zhou, Qibin Zhao, Cesar F. Caiafa, and Anh Huy Phan. Tensor decompositions for signal processing applications: From two-way to multiway component analysis. *IEEE Signal Process. Mag.*, 32(2):145–163, 2015.

Pierre Comon, Xavier Luciani, and André LF De Almeida. Tensor decompositions, alternating least squares and other tales. *Journal of Chemometrics: A Journal of the Chemometrics Society*, 23 (7-8):393–405, 2009.

Hanjun Dai, Hui Li, Tian Tian, Xin Huang, Lin Wang, Jun Zhu, and Le Song. Adversarial attack on graph structured data. In *ICML*, 2018.

Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *NeurIPS*, 2016.

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 *WSDM*, 2020.

Simon Geisler, Daniel Zügner, and Stephan Günnemann. Reliable graph neural networks via robust aggregation. In *NeurIPS*, 2020.

C Lee Giles, Kurt Bollacker, and Steve Lawrence CiteSeer. An automatic citation indexing system. In *Digital Libraries*, 1998.

Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In *ICML*, 2017.

Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *ICLR*, 2015.

Nathan Halko, Per-Gunnar Martinsson, and Joel A. Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Rev.*, 53(2): 217–288, 2011.

William L. Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NeurIPS*, 2017.

Richard A Harshman et al. Foundations of the parafac procedure: Models and conditions for an "explanatory" multimodal factor analysis. *UCLA Working Papers in phonetics*, 1970.

Wei Jin, Yaxin Li, Han Xu, Yiqi Wang, and Jiliang Tang. Adversarial attacks and defenses on graphs: A review and empirical study. *CoRR*, abs/2003.00653, 2020a.

Wei Jin, Yao Ma, Xiaorui Liu, Xianfeng Tang, Suhang Wang, and Jiliang Tang. Graph structure learning for robust graph neural networks. In *KDD*, 2020b.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.

Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. Predict then propagate: Graph neural networks meet personalized pagerank. In *ICLR*, 2019.

Jean Kossaifi, Yannis Panagakis, Anima Anandkumar, and Maja Pantic. TensorLy: Tensor learning in python. *J. Mach. Learn. Res.*, 20:26:1–26:6, 2019.

Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. Adversarial examples in the physical world. In *ICLR (Workshop)*, 2017a.

Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. Adversarial machine learning at scale. In *ICLR*, 2017b.

Alexandre Lacoste, Alexandra Luccioni, Victor Schmidt, and Thomas Dandres. Quantifying the carbon emissions of machine learning. *arXiv preprint arXiv:1910.09700*, 2019.

Brett W. Larsen and Tamara G. Kolda. Practical leverage-based sampling for low-rank tensor decomposition. *CoRR*, abs/2006.16438, 2020.

Richard B. Lehoucq, Danny C. Sorensen, and Chao Yang. *ARPACK users' guide - solution of large-scale eigenvalue problems with implicitly restarted Arnoldi methods*. Software, environments, tools. SIAM, 1998.

Yaxin Li, Wei Jin, Han Xu, and Jiliang Tang. DeepRobust: A PyTorch library for adversarial attacks and defenses. *CoRR*, abs/2005.06149, 2020.

Xuanqing Liu, Si Si, Jerry Zhu, Yang Li, and Cho-Jui Hsieh. A unified framework for data poisoning attack to graph-based semi-supervised learning. In *NeurIPS*, 2019.

Heng-Zhao Ma and Jianzhong Li. A true O(Nlog N) algorithm for the All-k-Nearest-Neighbors problem. In *COCOA*, volume 11949 of *Lecture Notes in Computer Science*, pp. 362–374, 2019.

Yao Ma, Suhang Wang, Lingfei Wu, and Jiliang Tang. Attacking graph convolutional networks via rewiring. *CoRR*, abs/1906.03750, 2019.

Andrew Kachites McCallum, Kamal Nigam, Jason Rennie, and Kristie Seymore. Automating the construction of internet portals with machine learning. *Information Retrieval*, 3(2):127–163, 2000.

Ivan V Oseledets. Tensor-train decomposition. *SIAM Journal on Scientific Computing*, 2011.

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. *NeurIPS*, 2019.

Amin Rakhsha, Goran Radanovic, Rati Devidze, Xiaojin Zhu, and Adish Singla. Policy teaching via environment poisoning: Training-time adversarial attacks against reinforcement learning. In *ICML*, 2020.

Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008.

Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868*, 2018.

Nicholas D. Sidiropoulos, Lieven De Lathauwer, Xiao Fu, Kejun Huang, Evangelos E. Papalexakis, and Christos Faloutsos. Tensor decomposition for signal processing and machine learning. *IEEE Trans. Signal Process.*, 65(13):3551–3582, 2017.

Xianfeng Tang, Yandong Li, Yiwei Sun, Huaxiu Yao, Prasenjit Mitra, and Suhang Wang. Robust graph neural network against poisoning attacks via transfer learning. *CoRR*, abs/1908.07558, 2019.

Xianfeng Tang, Yandong Li, Yiwei Sun, Huaxiu Yao, Prasenjit Mitra, and Suhang Wang. Transferring robustness for graph neural network against poisoning attacks. In *WSDM*, 2020.

Ledyard R Tucker. Some mathematical notes on three-mode factor analysis. *Psychometrika*, 1966.

Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *ICLR*, 2018.

Felix Wu, Amauri H. Souza Jr., Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Q. Weinberger. Simplifying graph convolutional networks. In *ICML*, 2019a.

Huijun Wu, Chen Wang, Yuriy Tyshetskiy, Andrew Docherty, Kai Lu, and Liming Zhu. Adversarial examples for graph data: Deep insights into attack and defense. In *IJCAI*, 2019b.

Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *IEEE Trans. Neural Networks Learn. Syst.*, 32 (1):4–24, 2021.

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. In *IJCAI*, 2019.

Xiang Zhang and Marinka Zitnik. Gnnguard: Defending graph neural networks against adversarial attacks. In *NeurIPS*, 2020.

Qibin Zhao, Guoxu Zhou, Shengli Xie, Liqing Zhang, and Andrzej Cichocki. Tensor ring decomposition. *arXiv preprint arXiv:1606.05535*, 2016.

Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI Open*, 1:57–81, 2020.

Dingyuan Zhu, Ziwei Zhang, Peng Cui, and Wenwu Zhu. Robust graph convolutional networks against adversarial attacks. In *KDD*, 2019.

Daniel Zügner and Stephan Günnemann. Certifiable robustness and robust training for graph convolutional networks. In *KDD*, 2019a.

Daniel Zügner and Stephan Günnemann. Adversarial attacks on graph neural networks via meta learning. In *ICLR*, 2019b.

Daniel Zügner, Amir Akbarnejad, and Stephan Günnemann. Adversarial attacks on neural networks for graph data. In *KDD*, 2018.

A APPENDIX

In this section, we provide more details of our experiments settings, visualization of TGNN compared with state-of-the-art methods, visualization of TGNN under other types of attacks, analyses of tensor decomposition formations and ranks, and hyperparameter effects. We also provide the code and data in the supplementary materials for reproducing the results we reported.

A.1 FURTHER IMPLEMENTATION DETAILS

We use PyTorch Paszke et al. (2019) (BSD Style License) to implement TGNN and TensorLy Kossaifi et al. (2019) (BSD 3-Clause License) for the tensor decomposition parts. For each graph, all nodes are randomly split into 10% of training nodes, 10% of validation nodes, and 80% of testing nodes. The best learned robust graph and GNN parameters are chosen based on the validation set for each graph. All the results are averaged based on 10 runs. We are using two different Adam Kingma & Ba (2015) optimizers for learning GNNs and the tensor cores in Eqn. (3, 4). The learning rate is 0.01. The weight decay for learning tensor cores and GNNs are 0.3 and 0. ReLU activation is applied after the first layer of the GCN. We train TGNN with 1000 epochs for each graph.

The goal of TGNN is to learn low-rank aggregation of the robust graphs, and the decomposition of TGNN is learned end-to-end. The diagnose is objective-oriented, which means to be based on the validation performance. We are focusing on the semi-supervised task of node classification that for each graph, so all the models are selected based on the best validation performance (accuracy). We found that the results for validation sets and the results for test sets show similar trends for different hyper-parameters.

The experiments are run on NVidia Tesla V100 GPU, Intel(R) Xeon(R) Silver 4116 CPU 2.10GHz, 192G RAM. It takes 83 seconds to train the full-fledged TGNN (GCN takes 39 seconds on the same device) which is estimated to emit 3.92g CO2 (estimators were conducted in ML CO2 impact calculator Lacoste et al. (2019)).

We analyze different hyperparameters and settings for TGNN on Cora dataset in Section A.7, A.8 and A.9. For the full implementation details, please refer to the code provided with supplementary material.

A.2 DATASETS STATISTICS

Following Zügner et al. (2018); Zügner & Günnemann (2019b); Jin et al. (2020b), we conduct experiments on four public benchmark datasets, including three citation graphs, *i.e.*, Cora McCallum et al. (2000), Citeseer Giles et al. (1998); Sen et al. (2008) and Pubmed Sen et al. (2008), and one blog graph Polblogs Adamic & Glance (2005). We were unable to find the license for the datasets we used. All the data (including perturbed graphs) was downloaded from DeepRobust Li et al. (2020) (an open library for attack and defense methods on images and graphs, with MIT License) and attached in the supplementary material. Note that we set the attribute matrix to $N \times N$ identity matrix and use the adjacency matrix to compute node similarities in the Polblogs graph since node features in this dataset are not available. Further detailed statistics of the four chosen datasets are summarized In Table 3. For predefined methods **PRUNE** and **KNN**, we use jaccard similarity for Cora, Citeseer and Polblogs, and cosine similarity for Pubmed.

Table 3: Dataset Statistics. Following Entezari et al. (2020); Zügner et al. (2018); Zügner & Günnemann (2019b), statistics for the largest connected component are demonstrated.

	Nodes	Edges	Classes	Features	Feature Type
Cora	2485	5069	7	1433	Binary
Citeseer	2110	3668	6	3703	Binary
Polblogs	1222	16714	2	-	-
Pubmed	19717	44338	3	500	Continuous

A.3 BASELINE SETTINGS

- **GCN (Kipf & Welling, 2017):** Among a number of different Graph Convolutional Networks (GCN) models, we choose the most popular one Kipf & Welling (2017) as a baseline and the default parameter setting in the official implementation is adopted.
- **GAT (Velickovic et al., 2018):** Graph Attention Network (GAT) leverages attention layers to learn different weights for different nodes in the neighborhood. GAT is often used as a baseline Jin et al. (2020b); Zhang & Zitnik (2020) to defend against adversarial attacks. Similar to GCN, the default parameter setting in the official implementation is adopted for GAT.
- **RGCN (Zhu et al., 2019):** For RGCN models, Gaussian distributions are applied to model node representations for mitigating effects of adversarial attacks. The nodes with high variance are penalized by the attention mechanism. For RGCN, the number of hidden units are tuned from $\{16, 32, 64, 128\}$ and the best choice is selected for each dataset.
- **GCN-Jaccard (Wu et al., 2019b):** GCN-Jaccard is a preprocessing method that eliminates edges connecting nodes with feature similarities less than a given threshold. This is the same approach to **PRUNE** described in Section 3.3. Jaccard similarity is used for datasets with binary feature type, while for continuous features cosine similarity is adopted. For Polblogs dataset where adjacency matrix is used as node feature, we are also using jaccard similarity. The threshold of similarity for pruning dissimilar edges are chosen from $\{0.01, 0.02, 0.03, 0.04, 0.05, 0.1\}$.
- **GCN-SVD (Entezari et al., 2020):** GCN-SVD preprocesses the network by vaccinating GCN with the low-rank approximation (implemented by SVD decomposition) of the perturbed graph. This is the exact approach to **SVD** described in Section 3.3. The rank of GCN-SVD is selected from $\{5, 10, 15, 50, 100, 200\}$.
- **Pro-GNN (Jin et al., 2020b):** Pro-GNN is a method that regularizes learned robust graph with three different graph properties: sparse, low-rank and feature smoothing. There are two settings (one with feature smoothing and one without) reported in the original paper. In Table 1, the numbers of these two settings are merged with the best results.
- **GNNGuard (Zhang & Zitnik, 2020):** GNN-Guard quantifies the relationship between the graph and node features by assigning less weights to suspicious edges to absorb malicious edges. We re-run the officially released code of GNN-Guard under ours setting and the results are reported in Table 1.

A.4 COMPLEXITY ANALYSES

Following the notation in Section 3, we use N/M to represent the number of nodes/edges. We will report the complexity of all the mentioned attack methods and defend methods. The time and space complexity are the same for most reported methods.

Let's first look at the complexity of attack methods.

- Random: $O(M \times \text{Perturbation Rate})$
- Nettack: $O(|\text{Target Nodes}| \times N)$
- Metattack: $O(N^2)$

Metattack (Zügner & Günnemann, 2019b) and Nettack (Zügner et al., 2018) are considering both deleting existing edges and wiring the node pairs in the graph. Metattack is a global poisoning attack method and is much more powerful than Nettack or Random attack.

- Defend methods in $O(M)$ complexity: RGCN, Jaccard, GNNGuard
- Defend methods in $O(N^2)$ complexity: SVD, KNN, Pro-GNN, TGNN

The exact time/space complexity of TGNN is $O((NK + N^2)Rd)/O((N + K)R + N^2)$, where R is the rank, and K is the number of predefined graphs. The number of feature/embedding dimensionality d, rank R, and the number of predefined graphs K are relatively low-dimensional compared to N or M. The complexity of TGNN is in the same order as that of Metattack.

Please note we are doing truncated SVD for the **SVD** method, which only needs the top-k singular values/vectors for the adjacency matrix. The name **SVD** is only used to differ from **PRUNE** and

KNN. We are doing Truncated SVD via scipy.sparse.linalg.svds. We adopt the default option from scipy, which uses ARPACK (Lehoucq et al., 1998). Computing SVD of a matrix is done by eigendecomposition of a square symmetric matrix. ARPACK is an iterative algorithm used to quickly find a few eigenvalues/eigenvectors of large sparse matrices. The complexity of calculating truncated SVD by ARPACK is $O(kN^2)$. There is another work (Halko et al., 2011) pointing out that with randomized techniques, finding the k dominant components of the SVD is of $O(log(k)N^2)$. Since k is relatively minor to N, the predefined SVD algorithm has the complexity of the order $O(N^2)$.

It also should be noted that the complexity of **KNN** (All-k-Nearest-Neighbors Problem) is proved to be of the order $O(NlogN)$ (Ma & Li, 2019). Here we store the dense representation of the KNN map, so the space complexity of **KNN** is $O(N^2)$.

For the proposed TGNN, we first apply random initialization to the core tensors, then apply the alternating least squares (ALS) approach for 100 iterations. After that, the core tensors are learned end-to-end with the whole model. Since how to learn tensor decomposition is not the main contribution of this submission, and we are using the code from TensorLy (Kossaifi et al., 2019) to produce the core tensors, we omit the detail of tensor decompositions. For CP decomposition as an example, the ALS approach chooses one core tensor and fixes the other tensors to solve the chosen core tensor. Since each core tensor represents a specific mode, the process of solving the chosen core tensor can be solved by matrix least square approximation. The approach is proposed in Harshman et al. (1970) and Carroll & Chang (1970), while detailed numerical complexity analyses can be found in Comon et al. (2009). Overall, each iteration for ALS and end-to-end core tensor training is of the order $O(N^2)$ complexity.

Intuitively, when the graph is severely perturbed, it is necessary to check each node pair from the global view in the graph to verify whether the edge is removed or added. Methods with $O(M)$ complexity only consider the edges that appear in the perturbed graph. They are not considering rewiring the edges deleted by the attack methods, which implies they are not able to defend against attacks that remove existing edges. As shown in Table 1 and Figure 3, TGNN outperforms all the methods with $O(M)$ complexity. Hence, we argue that $O(N^2)$ is a more reasonable complexity level than $O(M)$ to effectively defend the poisoning attack methods. When compared to the methods with the same level of $O(N^2)$ computational complexity, Pro-GNN and SVD, TGNN performs substantially better. TGNN takes 83s to be trained on Cora while GCN takes 39s , GNNGuard takes 94s and Pro-GNN takes 890s on the same machine. TGNN does not consume too much extra time to run.

A.5 Comparison between TGNN and Pro-GNN

Like TGNN, Pro-GNN Jin et al. (2020b) is also a method that defends adversarial attacks combining different graph properties (sparse, low-rank and feature smoothing). Different from TGNN that balances different graph properties with tensor-based aggregation, Pro-GNN regularizes learned robust graph with objective functions directly satisfying these graph properties, which may cause imbalance among the properties and the original graph information. In Figure 4, we visualize the learned adjacency matrix from TGNN and Pro-GNN. The input graph is the same as that in Figure 2, which is from Cora dataset and attacked by metattack with 25% perturbation rate.

As we can see in the figure, when the attack is severe, Pro-GNN is apt to learn robust graph by overfitting strong sparse and low-rank property with many original edges removed, which is harmful because the original edges are useful for GNNs to provide robust classification performance. The learned robust graph from TGNN covers most of the original edges but is also sparse and with low-rank structure. *For the point of view from each edge entry, the edge weight satisfies every graph properties. But looking from the whole graph, the original graph structure is destroyed.* Combined with the quantitative results in Table 1, TGNN balances different graph properties and the original graph information better, and outperforms Pro-GNN when the perturbation is severe.

A.6 Visualization of TGNN against Nettack and Random Attack

In Figure 2, we visualize the learned graph from TGNN and predefined methods to show how TGNN aggregate and learn robust graphs. Here we also visualize the results for the other two types of

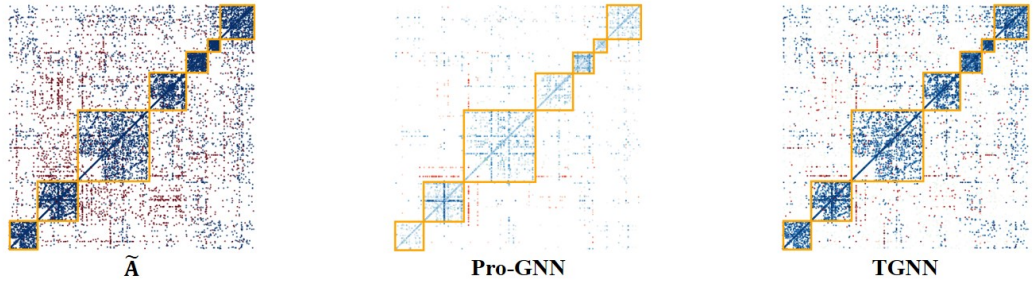


Figure 4: Visualization comparison between Pro-GNN Jin et al. (2020b) and TGNN on Cora dataset against metattack with 25% perturbation rate. The setting is the same as in Figure 2

adversarial attacks: nettack Zügner et al. (2018) and random attack in Figure 5 and Figure 6. Similar to Figure 2, the graphs in Figure 5 and Figure 6 are also perturbed in the most severe situation.

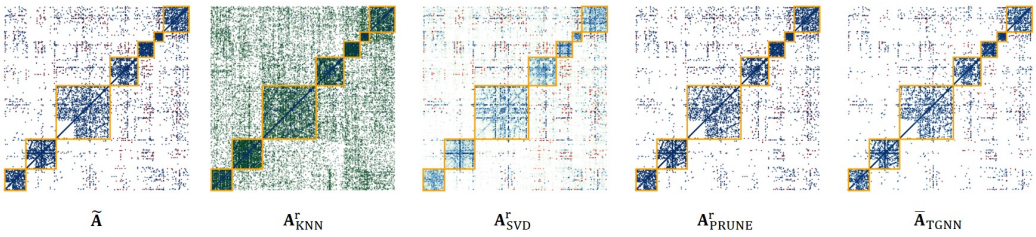


Figure 5: Visualization of TGNN and predefined methods against nettack with 5 perturbations per node on Cora dataset.

In Figure 5, we can see that nettack perturbs the graph with fewer edges compared with metattack, which is due to the targeted attacking mechanism. $\mathbf{A}^r_{SVD}$ and $\mathbf{A}^r_{KNN}$ ignore too much information from the original graph. TGNN generates robust graph similar to the one from $\mathbf{A}^r_{PRUNE}$ while still pays attention to spectral structures.

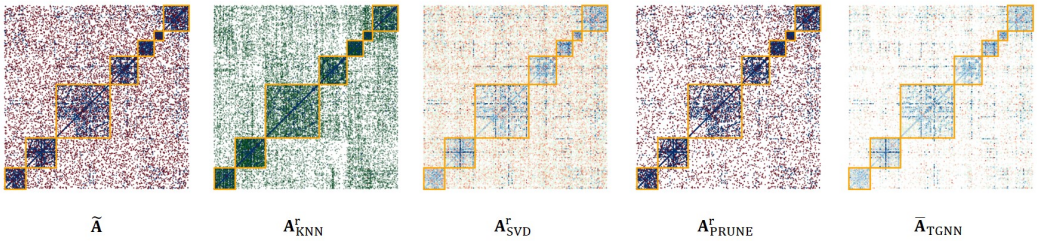


Figure 6: Visualization of TGNN and predefined methods against random attack with 100% perturbation rate on Cora dataset.

While for random attack, more adversarial edges are added so that $\mathbf{A}^r_{PRUNE}$ is not able to remove most of the perturbed edges. In this situation, TGNN generates robust graph which is more likely to the one from $\mathbf{A}^r_{SVD}$, while still keeps the low-rank structure of the perturbed graph, but with less weights for the perturbed edges.

A.7 Analyses for Tensor Decomposition Methods and Ranks

In this section, we compare different tensor decomposition methods for TGNN and discuss how different choices of tensor decomposition methods and ranks affect TGNN performance. In Figure 7, we illustrate the formation of different tensor decomposition method.

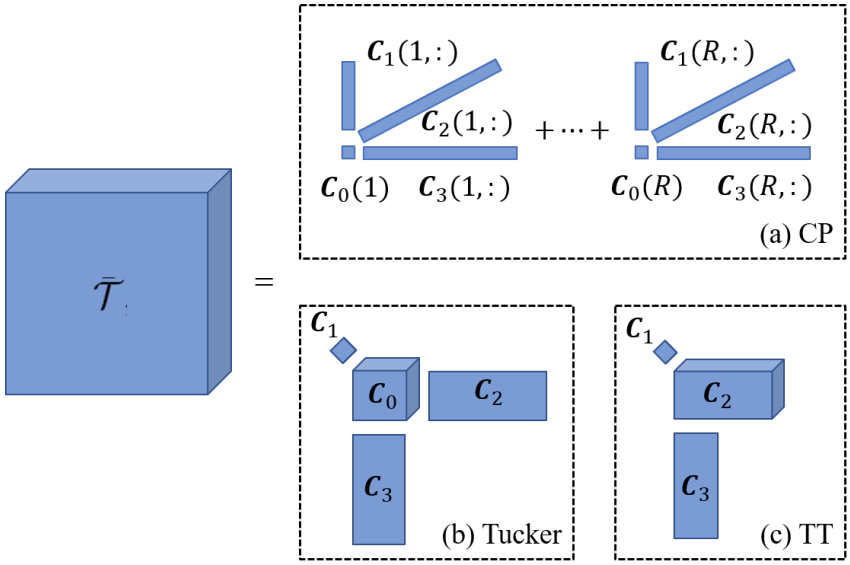


Figure 7: Illustration of different methods of tensor decomposition.

CP Decomposition (Harshman et al., 1970; Carroll & Chang, 1970): The formation of CP decomposition can be found in Eqn. (3). Here we rewrite the equation into element-wise formation:

$$\bar{\mathcal{T}}(i,j,k) = \sum_{r=1}^{R} \mathbf{C}_0(r)\mathbf{C}_1(r,i)\mathbf{C}_2(r,j)\mathbf{C}_3(r,k), \tag{8}$$

where $\mathbf{C}_1 \in \mathbb{R}^{R\times(K+1)}$, $\mathbf{C}_2 \in \mathbb{R}^{R\times N}$, $\mathbf{C}_3 \in \mathbb{R}^{R\times N}$ and $\mathbf{C}_0 \in \mathbb{R}^{R}$. There are $R \times (2N + K + 2)$ parameters in total. We select rank R from $\{8, 16, 32, 64, 128\}$ and plot the corresponding classification performance in Figure 8. All the results are on Cora dataset with metattack perturbations, which are under the same setting of Table 2 "TGNN-ALL".

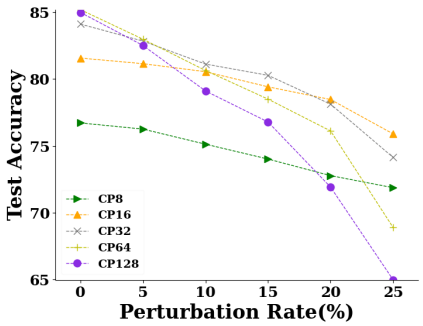


Figure 8: Node classification performance of TGNN with CP decomposition.

We can see from the figure that when R is too small (*i.e.*, $R = 8$), the low-rank property is too strong that the learned graph is not able to obtain enough information for GNNs for accurate classification. For $R \geq 16$, when R goes larger, performance for graphs with low perturbation rates is better, which means the low-rank property is not strong and the learned robust graph can reconstruct the original

graph well. But this may degrade the robustness of GNNs when the graph is heavily perturbed. Among all the choices of rank, we find that $R = 32$ is the best considering all the perturbation rates.

Tucker Decomposition (Tucker, 1966): Similar to CP decomposition, Tucker decomposition can be formulated as:

$$\bar{\mathcal{T}}(i,j,k) = \sum_{t_1,t_2,t_3} \mathcal{C}_0(t_1,t_2,t_3)\mathbf{C}_1(i,t_1)\mathbf{C}_2(j,t_2)\mathbf{C}_3(k,t_3), \tag{9}$$

where $\mathbf{C}_1 \in \mathbb{R}^{(K+1)\times(K+1)}$, $\mathbf{C}_2 \in \mathbb{R}^{R\times N}$, $\mathbf{C}_3 \in \mathbb{R}^{R\times N}$ and $\mathcal{C}_0 \in \mathbb{R}^{(K+1)\times R\times R}$. There are $R \times (R \times (K+1) + 2N) + (K+1) \times (K+1)$ parameters in total.

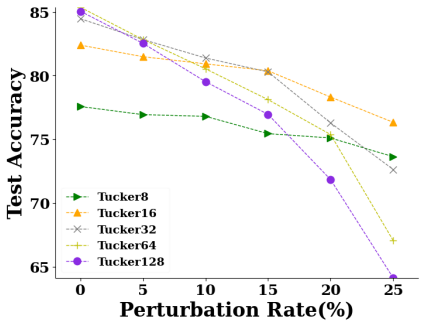


Figure 9: Node classification performance of TGNN with Tucker decomposition.

In Figure 9, we also plot the classification performance for different ranks of Tucker decomposition. Similar to CP decomposition, $R = 32$ is also the best choice considering all the perturbation rates.

Tensor-Train (TT) (Oseledets, 2011) Decomposition: TT decomposition can be formulated as:

$$\bar{\mathcal{T}}(i,j,k) = \sum_{t_1,t_2} \mathbf{C}_1(i,t_1)\mathcal{C}_2(t_1,j,t_2)\mathbf{C}_3(t_2,k), \tag{10}$$

where $\mathbf{C}_1 \in \mathbb{R}^{(K+1)\times(K+1)}$, $\mathcal{C}_2 \in \mathbb{R}^{(K+1)\times N\times R}$, $\mathbf{C}_3 \in \mathbb{R}^{R\times N}$. There are $R \times (N \times (K+1) + N) + (K+1) \times (K+1)$ parameters in total.

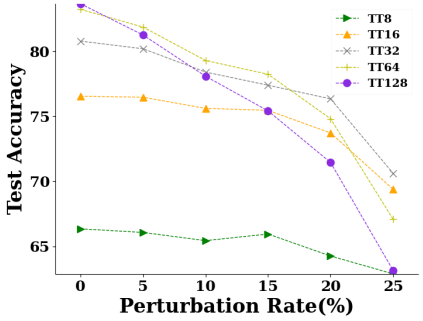


Figure 10: Node classification performance of TGNN with Tensor-Train decomposition.

In Figure 10, we plot the classification performance for different ranks of TT decomposition that $R = 32$ is the best choice considering all the perturbation rates. Please note there is another way to apply TT decomposition by swapping the first two modes of $\bar{\mathcal{T}}$. We omit this formation in this paper because it is very similar to the formation of Tucker decomposition.

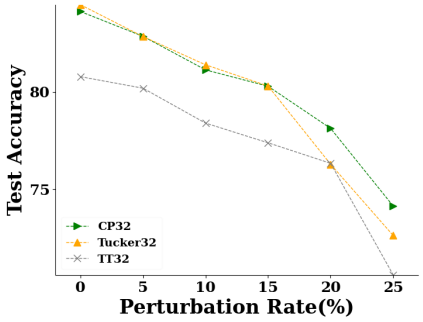


Figure 11: Comparison performance of TGNN with different decomposition methods.

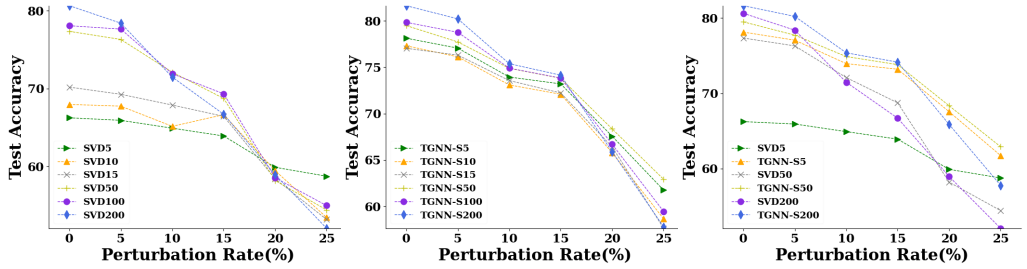
Comparing Different Tensor Decomposition Methods: Combining all the tensor decomposition methods, we find that CP decomposition , which costs the less amount of parameters under the same rank, is the best one for TGNN. Here we plot the results of CP, Tucker and TT decomposition when $R = 32$.

Please note it may be unfair to compare different decomposition methods with the same rank because the meanings of rank for these decomposition methods are different. We show the results to provide evidence for why we are using CP decomposition for our implementation.

A.8 Analyses for Hyperparameters in Predefined Methods

In this section, we analyze how hyperparameters affect the predefined methods and TGNN performance.

SVD: To investigate the impact of rank r in Eqn.(7) for SVD, we select rank r from $\{5, 10, 15, 50, 100, 200\}$ to conduct experiments with the setting of "SVD" and "TGNN-S" in Table 2. The results can be found in Figure 12.

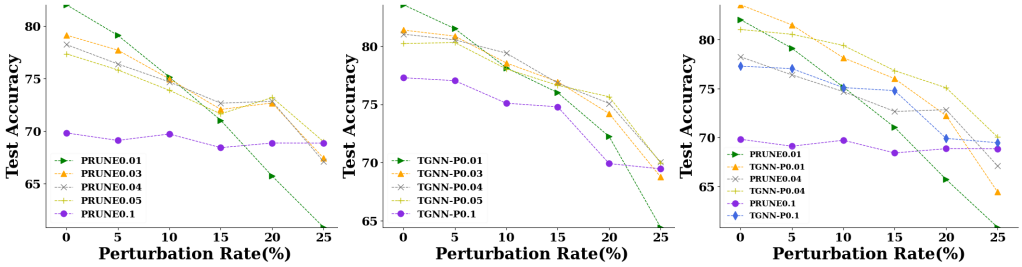


(a) SVD (b) TGNN-S (c) Comparison

Figure 12: Node classification performance of predefined method "SVD" and ablated method "TGNN-S" under different ranks of SVD.

For the predefined method "SVD" shown in Figure 12 (a) , the results are not promising when rank r is too small, which means the low-rank property is too strong for the method to carry enough information of the graph. When r grows, performance for the graphs with low perturbation rates becomes better, while the performance of the graphs with high perturbation rates is poorer. Similarly, in Figure 12 (b), "TGNN-S" also has this phenomenon but always performs better than "SVD" with the same ranks. We also plot the comparison results of "SVD" and "TGNN-S" with rank $\{5, 50, 200\}$ in Figure 12 (c) in which the gap between these two methods demonstrates the effectiveness of TGNN.

PRUNE: We tune the threshold τ in Eqn.(5) from $\{0.01, 0.02, 0.03, 0.04, 0.05, 0.1\}$ for "PRUNE" and "TGNN-P" in Table 2, the corresponding results are summarized in Figure 13. Please note the results for $\tau = 0.01$ and $\tau = 0.02$ are too close due to the node feature similarity distribution (*i.e.*, few node similarities are in the range of $[0.01, 0.02]$), so we omit the results for $\tau = 0.02$ in this figure.



(a) PRUNE (b) TGNN-P (c) Comparison

Figure 13: Node classification performance of predefined method "PRUNE" and ablated method "TGNN-P" under different thresholds τ of PRUNE.

For the predefined method "PRUNE" in Figure 13 (a) , when τ grows, more edges (can be both perturbed edges and the original edges) are pruned, resulting in the poorer performance of the graphs with low perturbation rates, while the graphs with high perturbation rates achieve better results. Comparing Figure 13 (a) and Figure 13 (b), it can be seen that "TGNN-P" always performs better than "PRUNE" under the same perturbation rate. From the comparison results of "PRUNE" and "TGNN-P" with thresholds $\{0.01, 0.04, 0.1\}$ in Figure 13 (c), the gap between these two methods also demonstrates the effectiveness of aggregation strategy in TGNN.

KNN: The k in Eqn.(6) is chosen from $\{8, 16, 32, 64, 128\}$. We plot the results for "KNN" and "TGNN-K" in Table 2 and the results can be found in Figure 14. Since the results for "KNN" are not changing with perturbation rates, we plot them with horizontal lines in this figure. The results for "KNN" and "TGNN-K" with the same k are in the same color.

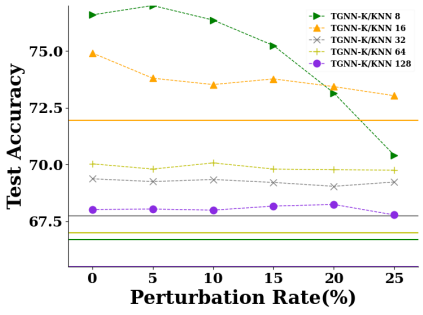


Figure 14: Node classification performance of predefined method "KNN" and ablated method "TGNN-K" under different top-k of KNN.

From the figure, we can see that "TGNN-K" always outperform predefined method "KNN".

A.9 Analyses for Other Hyperparameters

In this section, we analyze the other hyperparameters used in the proposed TGNN method, including λ in Eqn. (4) and the weight decay for learning tensor cores.

λ in Eqn. (4): The λ in Eqn. (4) is a hyperparameter balancing the classification loss and the low-rank approximation loss. Here we choose different λ from $\{1, 10, 100, 1000, 10000\}$ to verify

the performance of TGNN. The results are still from Cora dataset attacked with metattack and depicted in Figure 15. Please note the results for $\lambda = 1000$ and $\lambda = 10000$ are very close hence we omit the results for $\lambda = 10000$.

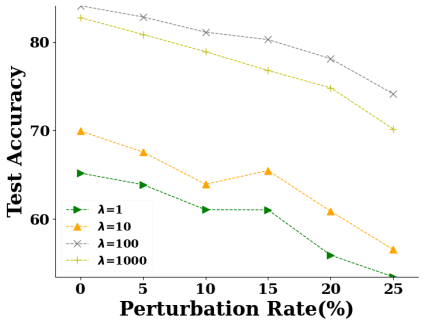


Figure 15: Node classification performance of TGNN tuned with different λ.

From the figure, we can see that $\lambda = 100$ is the best choice for TGNN to perform robustly against adversarial attacks.

Weight Decay for Learning Tensor Cores: In our TGNN, optimizer with weight decay 0.3 is adopted to learn the core tensors. To investigate the effects of weight decay, we choose it from $\{0, 0.01, 0.03, 0.1, 0.3, 1\}$ to tune TGNN and report the results in Figure 16.

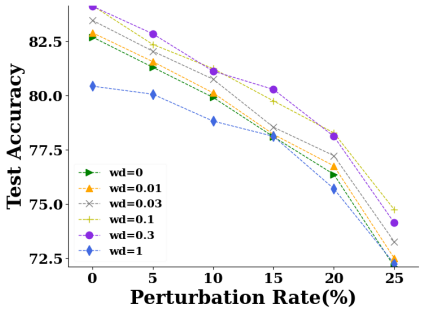


Figure 16: Node classification performance of TGNN optimized with different weight decay.

From the results, we can see that weight decay has minor effects on the performance of TGNN.

A.10 Limitations

One of the limitations for TGNN is scalability. All the existing methods use the same choice of datasets or others with similar scales. There are no existing defense methods evaluated on datasets on a scale of about or above 100000 nodes. It is also tough to evaluate attack methods like Metattack and Nettack on that scale because attack methods can wire any of the $N^2 - M$ edges not included in the graph, where N denotes the number of nodes and M denotes the number of edges. We argue that $O(N^2)$ is more reasonable than $O(M)$ for defense methods when considering to rewire the potential $N^2 - M$ deleted edges by the attacks. Our method has the same scalability as two competing methods (GCN-SVD and Pro-GNN), and is reasonable compared to the other three methods (GNNGuard, GCN-Jaccard and RGCN), while our method can beat all competing methods in terms of robustness to different attacks.

Here, we provide a potential solution for TGNN to handle large-scale graphs. The core idea is not to calculate the whole adjacency matrix explicitly. Sampling based algorithms have been used previously for scalable tensor decomposition (Larsen & Kolda, 2020; Cheng et al., 2016; Bhojanapalli & Sanghavi, 2015). In Cheng et al. (2016), statistical leverage scores (*i.e.*, sampling weights or

importance) are adaptively estimated by exploiting the tensor/matrix product structure of the designed tensors in least square problems. We can utilize this technique to reduce the complexity of the tensor approximation process of TGNN to $O(NR^3)$ for each iteration by reducing the size of least square problems in ALS. As we have analyzed in Section A.4, ignoring **SVD** and without explicitly constructing the whole predefined matrices, the complexity for **KNN** and **PRUNE** are $O(NlogN)$ and $O(M)$ respectively. We can also approximate the normalized graph $\hat{\mathbf{A}}$ directly to avoid estimating the diagonal degree matrix. Hence, the complexity of tensor approximation (calculating the core tensors) can be reduced to the order $O(M + NlogN)$. With the calculated core tensors, we can replace the calculation of each GCN layer in Eqn 2 by:

$$f_{\theta_k}(\mathbf{X}, \mathbf{C}_0, \mathbf{C}_1, \mathbf{C}_2, \mathbf{C}_3) = \sum_{r=1}^{R} \mathbf{C}_0(r)\mathbf{C}_1(r,1)\mathbf{C}_2(r,:)^T \left(\mathbf{C}_3(r,:)\mathbf{X}\mathbf{W}_k\right). \tag{11}$$

Since $\mathbf{C}_1 \in \mathbb{R}^{R\times(K+1)}$, $\mathbf{C}_2 \in \mathbb{R}^{R\times N}$, $\mathbf{C}_3 \in \mathbb{R}^{R\times N}$ and $\mathbf{C}_0 \in \mathbb{R}^R$, the computational complexity of each GCN layer in the above equation is of the order $O(NRD)$. Overall, ignoring the minor terms R and D, the complexity for this whole alternative algorithm is $O(M + NlogN)$.

Attack/defense algorithms for GNNs with high scalability would be interesting, and we will consider designing them and learn how they perform with TGNN as future works.

A.11 Transferability of the Learned Robust Graph $\bar{\mathbf{A}}$

In this section, we evaluate the transferability of the learned robust graph $\bar{\mathbf{A}}$ from TGNN. A robust graph is transferable if it can consistently defend different GNN models effectively. Since TGNN is trained with GCN, we evaluate the performance of $\bar{\mathbf{A}}$ transferring to GAT and another state-of-the-art GNN: APPNP (Klicpera et al., 2019). Evaluation is based on Metattack on Cora dataset and the results are in Figure 17.

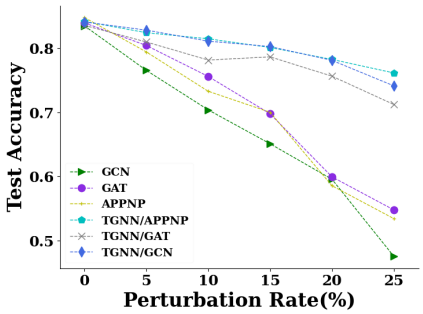


Figure 17: Defense transferability evaluation for the learned robust graph $\bar{\mathbf{A}}$ from TGNN.

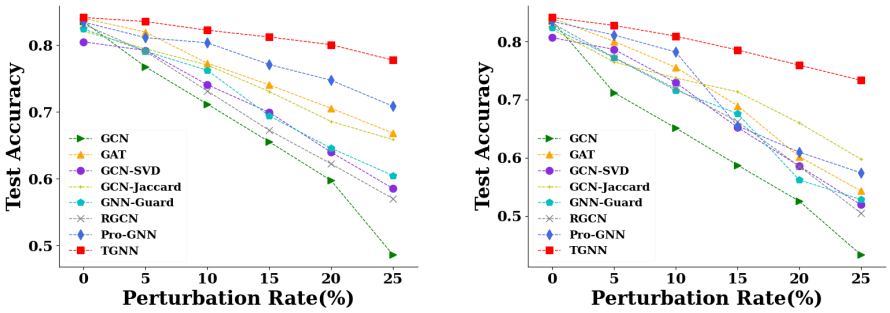
From Figure 17, we can see that APPNP also suffers from adversarial attacks (accuracy on clean graph: 84.80%, under 25% perturbation rate: 53.42%). Suppose we replace the perturbed graph $\tilde{\mathbf{A}}$ with the robust graph $\bar{\mathbf{A}}$ learned from TGNN by GCN for GAT and APPNP. In that case, we can significantly improve the classification performance for GAT and APPNP (under 25% perturbation rate, GAT: from 54.78% to 71.23%, APPNP: from 53.42% to 76.16%), which proves that the learned robust graph from TGNN can be transferred to different GNNs.

A.12 Results against Attacks with Features

In this section, we conduct experiments to verify whether TGNN is able to defend GNNs against adversarial attacks perturbing both structure and feature. We apply Metattack on Cora datasets changing both adjacency matrix and feature maps and evaluate state-of-the-art methods and the proposed TGNN. The feature matrix in Cora dataset is binary, so the attacks for feature can be applied in the same way as for structure. There are two different settings for this combined attacks: the first one is called **Combined**, following Zügner & Günnemann (2019b) that the attacks for feature matrix $\mathbf{X}$ and adjacency matrix $\mathbf{A}$ share the same budget (*a.k.a.*, maximum number of perturbations)

Δ, *i.e.*, $||\mathbf{A} - \tilde{\mathbf{A}}||_0 + ||\mathbf{X} - \tilde{\mathbf{X}}||_0 \leq \Delta$; the second setting is called **Separate** that the attacks for $\mathbf{X}$ and $\mathbf{A}$ have their own budget that $||\mathbf{A} - \tilde{\mathbf{A}}||_0 \leq \Delta_A$ and $||\mathbf{X} - \tilde{\mathbf{X}}||_0 \leq \Delta_X$. For **Combined**, we use the same perturbation rates as used for structure-only attacks in Table 1, where $\Delta = p \times ||\mathbf{A}||_0$ and $p \in \{5\%, 10\%, 15\%, 20\%, 25\%\}$. For **Separate**, the same set of perturbation rates are applied separately for feature and structure respectively that $\Delta_X = p \times ||\mathbf{X}||_0$ and $\Delta_A = p \times ||\mathbf{A}||_0$. The results are shown in Figure 18.

Under **Combined** setting, TGNN outperforms all the other methods, and it can also be observed that the impact of features for Metattack is slightly lower than the structures to GNNs (under 25% perturbation rate comparing TGNN and the best sota method, structure-only attack: TGNN 74.14% vs Pro-GNN 69.72%, **Combined**: TGNN 77.73% vs Pro-GNN 70.85%). In Zügner & Günnemann (2019b), similar observations are shown. We attribute this to two reasons: (1) **Unbalanced density between feature matrix and adjacency matrix:** The budgets assigned for feature and structure are the same for **Combined**; however, the density of feature and structure might be imbalanced. The average value of $\mathbf{A}$ for Cora is 0.0016, while the average value of the $\mathbf{X}$ is 0.0128. The adjacency matrix is sparser than the feature matrix, so it may cost more perturbations for attacking features to achieve the same degradation level as the attacks for structures. (2) **Structure and feature have different roles in GCN:** Based on the formulation of GCN (Eqn. (2), $f_\theta(\mathbf{X}, \mathbf{A}) = softmax(f_{\theta_2}(f_{\theta_1}(\mathbf{X}, \mathbf{A}), \mathbf{A}))$), the feature matrix $\mathbf{X}$ is only used in the first layer, while the adjacency matrix $\mathbf{A}$ appears in every layer. This can make it easier to perturb $\mathbf{A}$ than $\mathbf{X}$ for degrading the performance of GCN.



(a) **Combined** (b) **Separate**

Figure 18: Defense performance against Metattack for both structure and feature on Cora dataset. Left: **Combined** where structure and feature attacks share the same budget. Right: **Separate** where structure and feature attacks have different budgets.

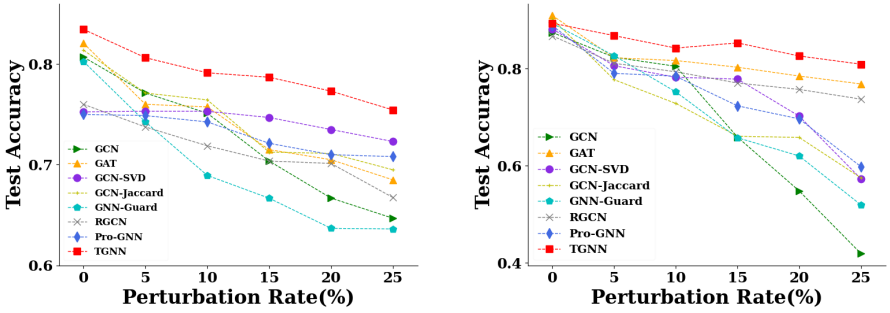
Under **Separate** setting, TGNN still outperforms all the other methods (under 25% perturbation rate comparing TGNN and the best sota method, TGNN 73.30% vs GCN-Jaccard 59.74%). The results show that TGNN is also robust against attacks with features. It might be a concern that attacks for features can mislead **KNN** and fool TGNN easily. However, **KNN** performs very consistently: only 362 out of 39720 entries (less than 1%) are changed for **KNN** predefined graph from the clean graph data to 25% perturbation rate under **Separate** setting.

We find that features are very robust to adversarial perturbation for GNNs. We think it is because of the different roles of feature and structure in GCN, as we have mentioned before. But unfortunately, if we only use features to classify the nodes, the performance is very poor (on clean graph, "No $\mathbf{A}$" in Table 2: 62.12%, **KNN**: 71.94%, GCN: 83.50%). How to effectively attack GNNs with features would be a very interesting topic, and we will address it in future works.

A.13 Results on Amazon Dataset

We add an experiment to show the effectiveness of TGNN defending GNNs on product co-purchase graphs. Amazon (Shchur et al., 2018) dataset is based on the Amazon co-purchase graph, where nodes represent goods, edges indicate that two goods are frequently bought together, node features are bag-of-words encoded product reviews, and class labels are given by the product category. It contains

two sub-datasets: Computers which contains 13752 nodes, 491722 edges, 767 binary features and 10 classes; and Photos which contains 7650 nodes, 238162 edges, 745 binary features and 8 classes.

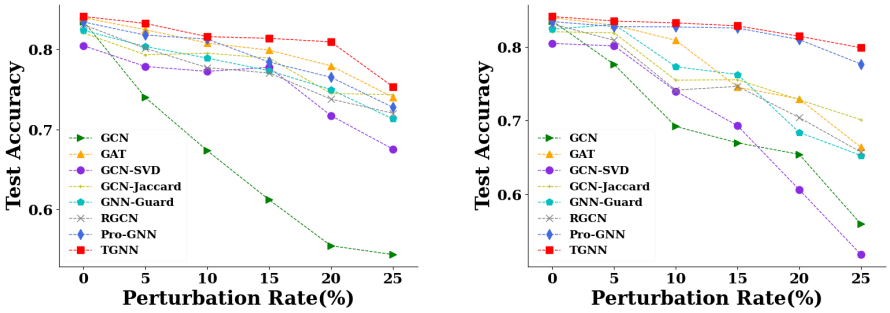


(a) Amazon Computers (b) Amazon Photos

Figure 19: Defense performance against Metattack on Amazon Computers and Photos datasets.

We evaluate TGNN and the state-of-the-art methods mentioned in Table 1 on Computers/Photos dataset against Metattack. The results can be found in Figure 19. We can see from the results that TGNN is the best method defending GNNs on product co-purchase graphs (under 25% perturbation rate comparing TGNN and the best sota method, Computers: TGNN 75.44% vs GCN-SVD 72.31%, Photos: TGNN 80.87% vs GAT 76.81%).

A.14 Results against Evasion Attacks



(a) PGD (b) Min-Max

Figure 20: Defense performance against PGD and Min-Max evasion attacks on Cora dataset.

In this section, we evaluate the proposed TGNN on another type of attacks: evasion attacks. Evasion attacks are also called as test-time attacks. Victim models are trained with clean data and tested with perturbed data. The goal of defending evasion attacks can be formally stated as:

Given fixed GNN parameters θ trained with the unperturbed adjacency matrix $\mathbf{A}$, feature $\mathbf{X}$ and partial node labels $\mathcal{Y}_P$, the GNN is evaluated with adversarial graph $\tilde{\mathbf{A}}$. We aim to learn robust graph $\bar{\mathbf{A}}$ replacing $\tilde{\mathbf{A}}$ to improve node classification performance for unlabeled nodes.

We mentioned that we are focusing on poisoning attacks in this work, and we argue that poisoning attacks for node classification are more reasonable due to the semi-supervised setting. Poisoning attacks are train-time attacks that perturb the graph data before the training process of the victim models. For reasons why the poisoning attack is more interesting and realistic in GNN, on the one hand, in evasion attack settings, GNN models are trained on the clean graph and evaluated on the perturbed graph generated by evasion attack methods. The graph information can be memorized by GNN models during the training process and be utilized to test to defend against evasion attacks. On the other hand, GNN models require the entire graph to classify every node, which means evasion

attacks provide the entire perturbed graph in the testing phase. Based on the given entire graph, we are able to retrain GNN models to defend against evasion attacks. Hence, poisoning attacks are more realistic than evasion attacks. In (Zügner et al., 2018), similar argumentation is made.

We choose two evasion attacks: PGD and Min-Max, proposed in Xu et al. (2019) to evaluate all the defense methods. The defense performance results of TGNN and state-of-the-art methods can be found in Figure 20. For GCN-Jaccard, GCN-SVD, Pro-GNN and TGNN, the surrogate GCN is firstly trained with clean graph. During the test-time, the surrogate GCN is fixed and used to learn the robust graph. From Figure 20, we can see that TGNN also shows more promising performance against evasion attacks than all state-of-the-art methods (under 25% perturbation rate comparing TGNN and the best sota method, PGD: TGNN 75.35% vs GCN-Jaccard 74.33%, Min-Max: TGNN 79.88% vs Pro-GNN 77.67%).