6
Robust Graph Neural Networks

6.1 Introduction

As the generalizations of traditional DNNs to graphs, GNNs inherit both advantages and disadvantages of traditional DNNs. Like traditional DNNs, GNNs have been shown to be effective in many graph-related tasks such as node-focused and graph-focused tasks. Traditional DNNs have been demonstrated to be vulnerable to dedicated designed adversarial attacks (Goodfellow et al., 2014b; Xu et al., 2019b). Under adversarial attacks, the victimized samples are perturbed in such a way that they are not easily noticeable, but they can lead to wrong results. It is increasingly evident that GNNs also inherit this drawback. The adversary can generate graph adversarial perturbations by manipulating the graph structure or node features to fool the GNN models. This limitation of GNNs has arisen immense concerns on adopting them in safety-critical applications such as financial systems and risk management. For example, in a credit scoring system, fraudsters can fake connections with several high-credit customers to evade the fraudster detection models; and spammers can easily create fake followers to increase the chance of fake news being recommended and spread. Therefore, we have witnessed more and more research attention to graph adversarial attacks and their countermeasures. In this chapter, we first introduce concepts and definitions of graph adversarial attacks and detail some representative adversarial attack methods on graphs. Then, we discuss representative defense techniques against these adversarial attacks.

6.2 Graph Adversarial Attacks

In graph-structured data, the adversarial attacks are usually conducted by modifying the graph structure and/or node features in an unnoticeable way such that the prediction performance can be impaired. Specifically, we denote a graph adversarial attacker as $\mathcal{T}$. Given a targeted model $f_{GNN}(;\mathbf{\Theta}))$ (either for node classification or for graph classification), the attacker $\mathcal{T}$ tries to modify a given graph $\mathcal{G}$ and generate an attacked graph $\mathcal{G}'$ as:

$$\mathcal{G}' = \mathcal{T}(\mathcal{G}; f_{GNN}(;\mathbf{\Theta})) = \mathcal{T}(\{\mathbf{A},\mathbf{F}\}; f_{GNN}(;\mathbf{\Theta})),$$

where $\mathcal{G} = \{\mathbf{A},\mathbf{F}\}$ is the input graph with $\mathbf{A}$ and $\mathbf{F}$ denoting its adjacency matrix and feature matrix and $\mathcal{G}' = \{\mathbf{A}',\mathbf{F}'\}$ is the produced attacked graph. Note that in this chapter, without specific mention, both the graph structure and the input features are assumed to be discrete, i.e, $\mathbf{A} \in \{0,1\}^{N\times N}$ and $\mathbf{F} \in \{0,1\}^{N\times d}$, respectively. The attacker is usually constraint to make unnoticeable modifications on the input graph, which can be represented as:

$$\mathcal{G}' \in \Phi(\mathcal{G}),$$

where $\Phi(\mathcal{G})$ denotes a constraint space that consists of graphs that are "close" to the graph $\mathcal{G}$. There are various ways to define the space $\Phi(\mathcal{G})$, which we will introduce when describing the attack methods. A typical and most commonly adopted constraint space is defined as:

$$\Phi(\mathcal{G}) = \{\mathcal{G}' = \{\mathbf{A}',\mathbf{F}'\}; \|\mathbf{A}'-\mathbf{A}\|_0 + \|\mathbf{F}'-\mathbf{F}\|_0 \le \Delta\}, \tag{6.1}$$

which means that the constraint space $\Phi(\mathcal{G})$ contains all the graphs that are within a given perturbation budget Δ away from the input graph $\mathcal{G}$. The goal of the attacker $\mathcal{T}$ is that the prediction results on the attacked graph $\mathcal{G}'$ are different from the original input graph. Specifically, for the node classification task, we focus on the prediction performance of a subset of nodes, which is called the victimized nodes and denoted as $\mathcal{V}_t \in \mathcal{V}_u$; while for the graph classification task, we concentrate on the prediction performance on a test set of graphs.

6.2.1 Taxonomy of Graph Adversarial Attacks

We can categorize graph adversarial attack algorithms differently according to the capacity, available resources, goals, and accessible knowledge of attackers.

Attackers' Capacity

Adversaries can perform attacks during both the model training and the model test stages. We can roughly divide attacks to evasion and poisoning attacks based on the attacker's capacity to insert adversarial perturbations:

- **Evasion Attack.** Attacking is conducted on the trained GNN model or in the test stage. Under the evasion attack setting, the adversaries cannot change the model parameters or structures.
- **Poisoning Attack.** Attacking happens before the GNN model is trained. Thus, the attackers can insert "poisons" into the training data such that the GNN models trained on this data have malfunctions.

Perturbation Type

In addition to node features, graph-structured data provides rich structural information. Thus, the attacker can perturb graph-structured data from different perspectives such as modifying node features, adding/deleting edges, and adding fake nodes:

- **Modifying Node Features.** Attackers can slightly modify the node features while keeping the graph structure.
- **Adding or deleting edges:** Attackers can add or delete edges.
- **Injecting Nodes.** Attackers can inject fake nodes to the graph, and link them with some benign nodes in the graph.

Attackers' Goal

According to the attackers' goal, we can divide the attacks into two groups:

- **Targeted Attack.** Given a small set of test nodes (or targeted nodes), the attackers target on misclassifying these test samples by the trained model. Targeted attacks can be further grouped into (1) direct attacks where the attacker directly perturbs the targeted nodes and (2) influencer attacks where the attacker can only manipulate other nodes to influence the targeted nodes.
- **Untargeted Attack.** The attacker aims to perturb the graph to reduce the trained model's overall performance.

Attackers' Knowledge

The attacks can be categorized into three classes according to the level of accessible knowledge towards the GNN model $f_{GNN}(;\boldsymbol{\Theta}))$ as follows:

- **White-box attack.** In this setting, the attackers are allowed to access full information of the attacked model $f_{GNN}(;\boldsymbol{\Theta}))$ (or the victim model) such as its architecture, parameters, and training data.
- **Gray-box attack.** In this setting, the attackers cannot access the architecture and the parameters of the victim model, but they can access the data utilized to train the model.

- **Black-box attack.** In this setting, the attackers can access to minimal information of the victim model. The attackers cannot access the architecture, model parameters, and the training data. The attackers are only allowed to query from the victim model to obtain the predictions.

In the following sections, we present some representative attack methods from each category based on attackers' knowledge, i.e., white-box, gray-box, and black-box attacks.

6.2.2 White-box Attack

In the white-box attack setting, the attacker is allowed to access full information of the victim model. In reality, this setting is not practical since complete information is often unavailable. However, it can still provide some information about the model's robustness against adversarial attacks. Most existing methods in this category utilize the gradient information to guide the attacker. There are two main ways to use the gradient information – 1) formulating the attack problem as an optimization problem that is addressed by the gradient-based method; and 2) using the gradient information to measure the effectiveness of modifying graph structure and features. Next, we present representative white-box attacks from these two ways.

PGD Topology Attack

In (Xu et al., 2019c), the attacker is only allowed to modify the graph structures but not the node features. The goal of the attacker is to reduce the node classification performance on a set of victimized nodes $\mathcal{V}_t$. A symmetric Boolean matrix $\mathbf{S} \in \{0,1\}^{N\times N}$ is introduced to encode the modification made by the attacker $\mathcal{T}$. Specifically, the edge between node v_i and node v_j is modified (added or removed) only when $\mathbf{S}_{i,j} = 1$, otherwise the edge is not modified. Given the adjacency matrix of a graph $\mathcal{G}$, its supplement can be represented as $\bar{\mathbf{A}} = \mathbf{1}\mathbf{1}^\top - \mathbf{I} - \mathbf{A}$, where $\mathbf{1} \in \mathbb{R}^N$ is a vector with all elements as 1. Applying the attacker $\mathcal{T}$ on the graph $\mathcal{G}$ can be represented as:

$$\mathbf{A}' = \mathcal{T}(\mathbf{A}) = \mathbf{A} + (\bar{\mathbf{A}} - \mathbf{A}) \odot \mathbf{S}, \tag{6.2}$$

where $\odot$ denotes the Hadamand product. The matrix $\bar{\mathbf{A}} - \mathbf{A}$ indicates whether an edge exists in the original graph or not. Specifically, when $(\bar{\mathbf{A}} - \mathbf{A})_{i,j} = 1$, there is no edge existing between node v_i and node v_j, thus the edge can be added by the attacker. When $(\bar{\mathbf{A}} - \mathbf{A})_{i,j} = -1$, there is an edge between nodes v_i and v_j, and it can be removed by the attacker.

The goal of the attacker $\mathcal{T}$ is to find $\mathbf{S}$ that can lead to bad prediction performance. For a certain node v_i, given its true label y_i, the prediction performance can be measured by the following CW-type loss adapted from Carlili-Wagner (CW) attacks in the image domain (Carlini and Wagner, 2017):

$$\ell(f_{GNN}(\mathcal{G}';\mathbf{\Theta})_i, y_i) = \max\left\{\mathbf{Z}'_{i,y_i} - \max_{c\neq y_i}\mathbf{Z}'_{i,c}, -\kappa\right\}, \tag{6.3}$$

where $\mathcal{G}' = \{\mathbf{A}', \mathbf{F}\}$ is the attacked graph, $f_{GNN}(\mathcal{G}';\mathbf{\Theta})_i$ is utilized to denote the i-th row of $f_{GNN}(\mathcal{G}';\mathbf{\Theta})$, $\mathbf{Z}' = f_{GNN}(\mathcal{G}';\mathbf{\Theta})$ is the logits calculated with Eq (5.48) on the attacked graph $\mathcal{G}'$. Note that we use the class labels y_i and c as the indices to retrieve the predicted probabilities of the corresponding classes. Specifically, $\mathbf{Z}'_{i,y_i}$ is the y_i-th element of the i-th row of $\mathbf{Z}'$, which indicates the probability of node v_i predicted as class y_i. The term $\mathbf{Z}'_{i,y_i} - \max_{c\neq y_i}\mathbf{Z}'_{i,c}$ in Eq. (6.3) measures the difference of the predicted probability between the true label y_i and the largest logit among all other classes. It is smaller than 0 when the prediction is wrong. Hence, for the goal of the attacker, we include a penalty when its value is larger than 0. Furthermore, in Eq. (6.3), $\kappa > 0$ is included as a confidence level of making wrong predictions. It means that a penalty is given when the difference between the logit of the true label y_i and the largest logit among all other classes is larger than $-\kappa$. A larger κ means that the prediction needs to be strongly wrong to avoid a penalty.

The attacker $\mathcal{T}$ is to find $\mathbf{S}$ in Eq. (6.2) such that it can minimize the CW-loss in Eq. (6.3) for all the nodes in the victimized node set $\mathcal{V}_t$ given a limited budget. Specifically, this can be represented as the following optimization problem:

$$\begin{aligned}\min_{\mathbf{s}} \mathcal{L}(\mathbf{s}) &= \sum_{v_i\in\mathcal{V}_t} \ell(f_{GNN}(\mathcal{G}';\mathbf{\Theta})_i, y_i)\\ \text{subject to} \quad &\|\mathbf{s}\|_0 \leq \Delta, \mathbf{s}\in\{0,1\}^{N(N-1)/2},\end{aligned} \tag{6.4}$$

where Δ is the budget to modify the graph and $\mathbf{s}\in\{0,1\}^{N(N-1)/2}$ is the vectorized $\mathbf{S}$ consisting of its independent perturbation variables. Note that $\mathbf{S}$ contains $N(N-1)/2$ independent perturbation variables, since $\mathbf{S}$ is a symmetric matrix with diagonal elements fixed to 0. The constraint term can be regarded as limiting the attacked graph $\mathcal{G}'$ in the space $\Phi(\mathcal{G})$ defined by the constraint on $\mathbf{s}$. The problem in Eq. (6.4) is a combinatorial optimization problem. For the ease of optimization, the constraint $\mathbf{s}\in\{0,1\}^{N(N-1)/2}$ is relaxed to its convex hull $\mathbf{s}\in[0,1]^{N(N-1)/2}$. Specifically, we denote the constraint space as $\mathcal{S} = \{\mathbf{s}; \|\mathbf{s}\|_0 \leq \Delta, \mathbf{s}\in[0,1]^{N(N-1)/2}\}$. Then the problem in Eq. (6.4) is now transformed to a continuous optimization problem. It can be solved by the projected gradient descent (PGD) method as:

$$\mathbf{s}^{(t)} = \mathcal{P}_{\mathcal{S}}[\mathbf{s}^{(t-1)} - \eta_t\nabla\mathcal{L}(\mathbf{s}^{(t-1)})],$$

where $\mathcal{P}_{\mathcal{S}}(\mathbf{x}) := \arg\min_{\mathbf{s}\in\mathcal{S}} \|\mathbf{s} - \mathbf{x}\|_2^2$ is the projection operator to project $\mathbf{x}$ into the continuous space $\mathcal{S}$. After obtaining the continuous $\mathbf{s}$ using the PGD method, the discrete $\mathbf{s}$ can be randomly sampled from it. Specifically, each element in the obtained $\mathbf{s}$ is regarded as the probability to sample 1 for the corresponding element of the discrete $\mathbf{s}$.

Integrated Gradient Guided Attack

The gradient information is utilized as scores to guide the attack in (Wu et al., 2019). The attacker is allowed to modify both the graph structure and the features. The attacker's goal is to impair the node classification performance of a single victimized node v_i. When modifying the structure, the attacker $\mathcal{T}$ is allowed to remove/add edges. The node features are assumed to be discrete features such as word occurrence or categorical features with binary values. Hence, the modification on both the graph structure and node features is limited to changing from either 0 to 1 or 1 to 0. This process can be guided by the gradient information of the objective function (Wu et al., 2019).

Inspired by Fast Gradient Sign Method (FGSM) (Goodfellow et al., 2014b), one way to find the the adversarial attack is to maximize the loss function used to train the neural network with respective to the input sample. For the victimized node v_i with label y_i, this loss can be denoted as:

$$\mathcal{L}_i = \ell(f_{GNN}(\mathbf{A}, \mathbf{F}; \mathbf{\Theta})_i, y_i).$$

In FSGM, one-step gradient ascent method is utilized to maximize the loss and consequently find the adversarial sample. However, in the graph setting, both the graph structure and node features are discrete, which cannot be derived by gradient-based methods. Instead, the gradient information corresponding to each element in $\mathbf{A}$ and $\mathbf{F}$ is used to measure how their changes affect the value of loss function. Thus, it can be used to guide the attacker to perform the adversarial perturbation. However, as the attacker is only allowed to perform modification from either 0 to 1 or 1 to 0, the gradient information may not help too much for the following reason – given that the graph neural network model is non-linear, the gradient on a single point cannot reflect the effect of a large change such as from 0 to 1 or from 1 to 0. Hence, inspired by the integrated gradients (Sundararajan et al., 2017), discrete integrated gradients are utilized to design the scores, which are called as the integrated gradient scores (IG-scores). Specifically, the IG-score discretely accumulates the gradient information of changing from 0 to 1 or from 1 to 0 as:

$$IG_{\mathbf{H}}(i,j) = \frac{\mathbf{H}_{i,j}}{m} \sum_{k=1}^{m} \frac{\partial \mathcal{L}_i(\frac{k}{m}(\mathbf{H}_{i,j} - 0))}{\partial \mathbf{H}_{i,j}}; \quad 1 \to 0, \text{ when } \mathbf{H}_{i,j} = 1;$$

$$IG_{\mathbf{H}}(i,j) = \frac{1 - \mathbf{H}_{i,j}}{m} \sum_{k=1}^{m} \frac{\partial \mathcal{L}_i(0 + \frac{k}{m}(1 - \mathbf{H}_{i,j}))}{\partial \mathbf{H}_{i,j}}; \quad 0 \to 1, \text{ when } \mathbf{H}_{i,j} = 0;$$

where $\mathbf{H}$ could be either $\mathbf{A}$ or $\mathbf{F}$, and m is a hyperparameter indicating the number of discrete steps. We denote the IG-scores for the candidate changes in $\mathbf{A}$ and $\mathbf{F}$ as $IG_{\mathbf{A}}$ and $IG_{\mathbf{F}}$ respectively, which measure how the corresponding change in each element of $\mathbf{A}$ and $\mathbf{F}$ affects the loss $\mathcal{L}_i$. Then, the attacker $\mathcal{T}$ can make the modification by selecting the action with the largest IG-score among $IG_{\mathbf{A}}$ and $IG_{\mathbf{F}}$. The attacker repeats this process as long as the resulting graph $\mathcal{G}' \in \Phi(\mathcal{G})$, where $\Phi(\mathcal{G})$ is defined as in Eq. (6.1).

6.2.3 Gray-box Attack

In the gray-box attack setting, the attacker is not allowed to access the architecture and parameters of the victim model, but can access the data utilized to train the model. Hence, instead of directly attacking the given model, the gray-box attacks often first train a surrogate model with the provided training data and then attack the surrogate model on a given graph. They assume that these attacks on the graph via the surrogate model can also damage the performance of the victim model. In this section, we introduce representative gray-box attack methods.

Nettack

The Nettack model (Zügner et al., 2018) targets on generating adversarial graphs for the node classification task. A single node v_i is selected as the victim node to be attacked and the goal is to modify the structure and/or the features of this node or its nearby nodes to change the prediction on this victim node. Let us denote the label of the victim node v_i as y_i, where y_i could be either the ground truth or the label predicted by the victim model $f_{GNN}(\mathbf{A}, \mathbf{F}; \mathbf{\Theta})$ on the original clean graph $\mathcal{G}$. The goal of the attacker is to modify the graph $\mathcal{G}$ to $\mathcal{G}' = \{\mathbf{A}', \mathbf{F}'\}$ such that the model trained on the attacked graph $\mathcal{G}'$ classifies the node v_i as a new class c. In general, the attacking problem can be described as the following optimization problem:

$$\arg\max_{\mathcal{G}' \in \Phi(\mathcal{G})} \left(\max_{c \neq y_i} \ln \mathbf{Z}'_{i,c} - \ln \mathbf{Z}'_{i,y_i} \right), \tag{6.5}$$

where $\mathbf{Z}' = f_{GNN}(\mathbf{A}', \mathbf{F}'; \mathbf{\Theta}')$ with the parameters $\mathbf{\Theta}'$ learned by minimizing Eq. (5.49) on the attacked graph $\mathcal{G}'$. Here, the space $\Phi(\mathcal{G})$ is defined based on the limited budget constraint as Eq. (6.1) and two more constraints on the perturbations. These two constraints are: 1) the degree distribution of the attacked graph should be close to that of the original graph; and 2) the distribution of the feature occurrences (for the discrete features) of the attacked graph should be close to that of the original graph. Solving the problem in Eq. (6.5) directly is very challenging as the problem involves two dependent stages. The discrete structure of the graph data further increases the difficulty. To address these difficulties, we first train a *surrogate model* on the original clean graph data $\mathcal{G}$ and then generate the adversarial graph by attacking the surrogate model. The adversarial graph is treated as the attacked graph. When attacking graph neural network model built upon GCN-Filters (see Section 5.3.2 for details on GCN-Filter) for node classification, the following surrogate model with 2 GCN-Filters and no activation layers is adopted:

$$\mathbf{Z}^{sur} = \text{softmax}(\tilde{\mathbf{A}}\tilde{\mathbf{A}}\mathbf{F}\mathbf{\Theta}_1\mathbf{\Theta}_2) = \text{softmax}(\tilde{\mathbf{A}}^2\mathbf{F}\mathbf{\Theta}), \tag{6.6}$$

where the parameters $\mathbf{\Theta}_1$ and $\mathbf{\Theta}_2$ are absorbed in $\mathbf{\Theta}$. The parameters $\mathbf{\Theta}$ are learned from the original clean graph $\mathcal{G}$ with the provided training data. To perform the adversarial attack based on the surrogate model, as in Eq. (6.5), we aim to find these attacks that maximize the difference, i.e., $\max_{c \neq y_i} ln\mathbf{Z}^{sur}_{i,c} - ln\mathbf{Z}^{sur}_{i,y_i}$. To further simplify the problem, the instance independent softmax normalization is removed, which results in the following surrogate loss:

$$\mathcal{L}_{sur}(\mathbf{A}, \mathbf{F}; \mathbf{\Theta}, v_i) = \max_{c \neq y_i} \left([\tilde{\mathbf{A}}^2\mathbf{F}\mathbf{\Theta}]_{i,c} - [\tilde{\mathbf{A}}^2\mathbf{F}\mathbf{\Theta}]_{i,y_i} \right).$$

Correspondingly the optimization problem can be expressed as:

$$\text{argmax}_{\mathcal{G}' \in \Phi(\mathcal{G})} \mathcal{L}_{sur}(\mathbf{A}', \mathbf{F}'; \mathbf{\Theta}, v_i). \tag{6.7}$$

While being much simpler, this problem is still intractable to be solved exactly. Hence, a greedy algorithm is adopted, where we measure the scores of all the possible steps (adding/deleting edges and flip features) as follows:

$$s_{str}(e; \mathcal{G}^{(t)}, v_i) := \mathcal{L}_{sur}(\mathbf{A}^{(t+1)}, \mathbf{F}^{(t)}; \mathbf{\Theta}, v_i)$$
$$s_{feat}(f; \mathcal{G}^{(t)}, v_i) := \mathcal{L}_{sur}(\mathbf{A}^{(t)}, \mathbf{F}^{(t+1)}; \mathbf{\Theta}, v_i)$$

where $\mathcal{G}^{(t)} = \{\mathbf{A}^{(t)}, \mathbf{F}^{(t)}\}$ is the intermediate result of the algorithm at the step t, $\mathbf{A}^{(t+1)}$ is one step change from $\mathbf{A}^{(t)}$ by adding/deleting the edge e and $\mathbf{F}^{(t+1)}$ is one step change away from $\mathbf{F}^{(t)}$ by flipping the feature f. The score $s_{str}(e; \mathcal{G}^{(t)}, v_i)$ measures the impact of changing the edge e on the loss function, while $s_{feat}(f; \mathcal{G}^{(t)}, v_i)$ indicates how changing the feature f affects the loss function. In each step, the greedy algorithm chooses the edge or the feature with the largest score to perform the corresponding modification, i.e. (adding/deleting edges or flipping features). The process is repeated as long as the resulting graph is still in the space of $\Phi(\mathcal{G})$.

Metattack

The Metattack method in (Zügner and Günnemann, 2019) tries to modify the graph to reduce the overall node classification performance on the test set, i.e., the victim node set $\mathcal{V}_t = \mathcal{V}_u$. The attacker in Metattack is limited to modify the graph structure. The constraint space $\Phi(\mathcal{G})$ is adopted from Nettack where the limited budget constraint and the degree preserving constraint are used to define the constraint space. The metattack is a poisoning attack. Thus, after generating the adversarial attacked graph, we need to retrain the victim model on the attacked graph. The goal of the attacker is to find such an adversarial attacked graph that the performance of the retrained victim GNN model is impaired. Hence, the attacker can be mathematically formulated as a bi-level optimization problem as:

$$\min_{\mathcal{G}' \in \Phi(\mathcal{G})} \mathcal{L}_{atk}(f_{GNN}(\mathcal{G}'; \mathbf{\Theta}^*)) \quad s.t. \quad \mathbf{\Theta}^* = \arg\min_{\mathbf{\Theta}} \mathcal{L}_{tr}(f_{GNN}(\mathcal{G}'; \mathbf{\Theta})), \tag{6.8}$$

where $f_{GNN}()$ is the victim model, and $\mathcal{L}_{tr}$ denotes the loss function used to train the model as defined in Eq. (5.49) over the training set $\mathcal{V}_l$. The loss function $\mathcal{L}_{atk}$ is to be optimized to generate the adversarial attack. In particular, the lower-level optimization problem with respect to $\mathbf{\Theta}$ is to find the best model parameters $\mathbf{\Theta}^*$ given the attacked graph $\mathcal{G}'$, while the higher-level optimization problem is to minimize $\mathcal{L}_{atk}$ to generate the attacked graph $\mathcal{G}'$. Since the goal of the attacker is to impair the performance on the unlabelled nodes, ideally, $\mathcal{L}_{atk}$ should be defined based on $\mathcal{V}_u$. However, we cannot directly calculate the loss based on $\mathcal{V}_u$ without the labels. Instead, one approach, which is based on the argument that the model cannot generalize well if it has high training error, is to define $\mathcal{L}_{atk}$ as the negative of the $\mathcal{L}_{tr}$, i.e., $\mathcal{L}_{atk} = -\mathcal{L}_{tr}$. Another way to formulate $\mathcal{L}_{atk}$ is to first predict labels for the unlabeled nodes using a well trained surrogate model on the original graph $\mathcal{G}$ and then use the predictions as the "labels". More specifically, let C'_u denote the "labels" of unlabeled nodes $\mathcal{V}_u$ predicted by the surrogate model. The loss function $\mathcal{L}_{self} = \mathcal{L}(f_{GNN}(\mathcal{G}'; \mathbf{\Theta}^*), C'_u)$ measures the disagreement between the "labels" C'_u and the predictions from $f_{GNN}(\mathcal{G}'; \mathbf{\Theta}^*)$ as in Eq. (5.49) over the set $\mathcal{V}_u$. The second option of $\mathcal{L}_{atk}$ can be defined as $\mathcal{L}_{atk} = -\mathcal{L}_{self}$. Finally, the $\mathcal{L}_{atk}$ is defined as a combination of the two loss functions as:

$$\mathcal{L}_{atk} = -\mathcal{L}_{tr} - \beta \cdot \mathcal{L}_{self},$$

where β is a parameter controlling the importance of $\mathcal{L}_{self}$.

To solve the bi-level optimization problem in Eq. (6.8), the meta-gradients, which have traditionally been used in meta-learning, are adopted. Meta-gradients can be viewed as the gradients with respect to the hyper-parameters. In this specific problem, the graph structure (or the adjacency matrix $\mathbf{A}$) is treated as the hyperparameters. The goal is to find the "optimal" structure such that the loss function $\mathcal{L}_{atk}$ is minimized. The meta-gradient with respect to the graph $\mathcal{G}$ can be defined as:

$$\nabla_{\mathcal{G}}^{meta} := \nabla_{\mathcal{G}}\mathcal{L}_{atk}(f_{GNN}(\mathcal{G};\mathbf{\Theta}^*)) \quad s.t. \quad \mathbf{\Theta}^* = \arg\min_{\mathbf{\Theta}} \mathcal{L}_{tr}(f_{GNN}(\mathcal{G};\mathbf{\Theta})). \quad (6.9)$$

Note that the meta-gradient is related to the parameter $\mathbf{\Theta}^*$ as $\mathbf{\Theta}^*$ is a function of the graph $\mathcal{G}$ according to the second part of Eq. (6.9). The meta-gradient indicates how a small change in the graph $\mathcal{G}$ affects the attacker loss $\mathcal{L}_{atk}$, which can guide us how to modify the graph.

The inner problem of Eq. (6.8) (the second part of Eq. (6.9)) typically does not have an analytic solution. Instead, a differentiable optimization procedure such as vanilla gradient descent or stochastic gradient descent (SGD) is adopted to obtain $\mathbf{\Theta}^*$. This optimization procedure can be represented as $\mathbf{\Theta}^* = \text{opt}_{\mathbf{\Theta}}\mathcal{L}_{tr}(f_{GNN}(\mathcal{G};\mathbf{\Theta}))$. Thus, the meta-gradient can be reformulated as:

$$\nabla_{\mathcal{G}}^{meta} := \nabla_{\mathcal{G}}\mathcal{L}_{atk}(f_{GNN}(\mathcal{G};\mathbf{\Theta}^*)) \quad s.t. \quad \mathbf{\Theta}^* = \text{opt}_{\mathbf{\Theta}}\mathcal{L}_{tr}(f_{GNN}(\mathcal{G};\mathbf{\Theta})). \quad (6.10)$$

As an illustration, the $\text{opt}_{\mathbf{\Theta}}$ with vanilla gradient descent can be formalized as:

$$\mathbf{\Theta}_{t+1} = \mathbf{\Theta}_t - \eta \cdot \nabla_{\mathbf{\Theta_t}}\mathcal{L}_{tr}(f_{GNN}(\mathcal{G};\mathbf{\Theta})) \quad \text{for } t = 0, \ldots, T-1,$$

where η is the learning rate, $\mathbf{\Theta_0}$ denotes the initialization of the parameters, T is the total number of steps of the gradient descent procedure and $\mathbf{\Theta}^* = \mathbf{\Theta}_T$. The meta-gradient can now be expressed by unrolling the training procedure as follows:

$$\begin{aligned}\nabla_{\mathcal{G}}^{meta} &= \nabla_{\mathcal{G}}\mathcal{L}_{atk}(f_{GNN}(\mathcal{G};\mathbf{\Theta}_T)) \\ &= \nabla_{f_{GNN}}\mathcal{L}_{atk}(f_{GNN}(\mathcal{G};\mathbf{\Theta}_T)) \cdot [\nabla_{\mathcal{G}} f_{GNN}(\mathcal{G};\mathbf{\Theta}_T) + \nabla_{\mathbf{\Theta_T}} f_{GNN}(\mathcal{G};\mathbf{\Theta}_T) \cdot \nabla_{\mathcal{G}}\mathbf{\Theta_T}],\end{aligned}$$

where

$$\nabla_{\mathcal{G}}\mathbf{\Theta}_{t+1} = \nabla_{\mathcal{G}}\mathbf{\Theta}_t - \eta\nabla_{\mathcal{G}}\nabla_{\mathbf{\Theta}_t}\mathcal{L}_{tr}(f_{GNN}(\mathcal{G};\mathbf{\Theta}_t)).$$

Note that the parameter $\mathbf{\Theta}_t$ is dependent on the graph $\mathcal{G}$; thus, the derivative with respect to the graph $\mathcal{G}$ has to chain back all the way to the initial parameter $\mathbf{\Theta}_0$. After obtaining the meta-gradient, we can now use it to update the graph as:

$$\mathcal{G}^{(k+1)} = \mathcal{G}^{(k)} - \gamma \nabla_{\mathcal{G}^{(k)}} \mathcal{L}_{atk}(f_{GNN}(\mathcal{G}; \mathbf{\Theta}_T)). \tag{6.11}$$

The gradients are dense; thus, the operation in Eq. (6.11) results in a dense graph, which is not desired. Furthermore, as the structure and the parameters of the model are unknown in the gray-box setting, the meta-gradients cannot be obtained. To solve these two issues, a greedy algorithm utilizing the meta-gradient calculated on a surrogate model as the guidance to choose the action is further proposed in (Zügner and Günnemann, 2019). We next introduce the meta-gradient based greedy algorithm. The same surrogate model as Eq. (6.6) is utilized to replace $f_{GNN}(\mathcal{G}; \mathbf{\Theta})$ in Eq. (6.8). A score to measure how a small change in the i, j-th element of the adjacency matrix $\mathbf{A}$ affects the loss function $\mathcal{L}_{atk}$ is defined by using the meta-gradient as:

$$s(i, j) = \nabla^{meta}_{\mathbf{A}_{i,j}} \cdot (-2 \cdot \mathbf{A}_{i,j} + 1),$$

where the term $(-2 \cdot \mathbf{A}_{i,j} + 1)$ is used to flip the sign of the meta-gradients when $\mathbf{A}_{i,j} = 1$, i.e., the edge between nodes v_i and v_j exists and can only be removed. After calculating the score for each possible action based on the meta-gradients, the attacker takes the action with the largest score. For a chosen node pair (v_i, v_j), the attacker adds an edge between them if $\mathbf{A}_{i,j} = 0$ while removing the edge between them if $\mathbf{A}_{i,j} = 1$. The process is repeated as long as the resulting graph is in the space $\Phi(\mathcal{G})$.

6.2.4 Black-box Attack

In the black-box attack setting, the victim model's information is not accessible to the attacker. The attacker can only query the prediction results from the victim model. Most methods in this category adopt reinforcement learning to learn the strategies of the attacker. They treat the victim model as a black-box query machine and use the query results to design the reward for reinforcement learning.

RL-S2V

The RL-S2V method is a black-box attack model using reinforcement learning (Dai et al., 2018). In this setting, a target classifier $f_{GNN}(\mathcal{G}; \mathbf{\Theta})$ is given with the parameters $\mathbf{\Theta}$ learned and fixed. The attacker is asked to modify the graph such that the classification performance is impaired. The RL-S2V attacker can be used to attack both the node classification task and the graph classification task. The RL-S2V attacker only modifies the graph structure and leaves the graph features untouched. To modify the graph structure, the RL-S2V attacker is allowed to add or delete edges from the original graph $\mathcal{G}$. The constraint space for RL-S2V can be defined as:

$$\begin{aligned}&\Phi(\mathcal{G}) = \{\mathcal{G}'; |(\mathcal{E} - \mathcal{E}') \cup (\mathcal{E}' - \mathcal{E})| \leq \Delta\} \\ &\text{with } \mathcal{E}' \subset \mathcal{N}(\mathcal{G}, b),\end{aligned} \tag{6.12}$$

where $\mathcal{E}$ and $\mathcal{E}'$ denote the edge sets of the original graph $\mathcal{G}$ and the attacked graph $\mathcal{G}'$, respectively. Δ is the budget limit to remove and add edges. Furthermore, $\mathcal{N}(\mathcal{G}, b)$ is defined as:

$$\mathcal{N}(\mathcal{G}, b) = \{(v_i, v_j) : v_i, v_j \in \mathcal{V}, \text{dis}^{(\mathcal{G})}(v_i, v_j) \leq b\}.$$

where $\text{dis}^{(\mathcal{G})}(v_i, v_j)$ denotes the shortest path distance between node v_i and node v_j in the original graph $\mathcal{G}$. This indicates that the resulting edges in $\mathcal{E}'$ are within b-hop neighborhood in the original graph. The attacking procedure of RL-S2V is modeled as a Finite Markov Decision Process (MDP), which can be defined as follows:

- **Action:** As mentioned before, there are two types of actions: adding and deleting edges. Furthermore, only those actions that lead to a graph in the constraint space $\Phi(\mathcal{G})$ are considered as valid actions.
- **State:** The state s_t at the time step t is the intermediate graph $\mathcal{G}_t$, which is obtained by modifying the intermediate graph $\mathcal{G}_{t-1}$ by a singe action.
- **Reward:** The purpose of the attacker is to modify the graph such that the targeted classifier would be fooled. A reward is only granted when the attacking process (MDP) has been terminated. More specifically, a positive reward $r(s_t, a_t) = 1$ is granted if the targeted model makes a different prediction from the original one; otherwise, a negative reward of $r(s_t, a_t) = -1$ is granted. For all the intermediate steps, the reward is set to $r(s_t, a_t) = 0$.
- **Terminal:** The MDP has a total budget of Δ to perform the actions. The MDP is terminated once the agent reaches the budget Δ, i.e., the attacker has modified Δ edges.

Deep Q-learning is adopted to learn the MDP (Dai et al., 2018). Specifically, Q-learning (Watkins and Dayan, 1992) is to fit the following Bellman optimal equation:

$$Q^*(s_t, a_t) = r(s_t, a_t) + \gamma \max_{a'} Q^*(s_{t+1}, a'),$$

In particular, $Q^*()$ is a parametrized function to approximate the optimal expected future value (or the expected total reward of all future steps) given a state-action pair and γ is the discount factor. Once the $Q^*()$ function is learned during training, it implicitly indicates a greedy policy:

$$\pi(a_t|s_t; Q^*) = \arg\max_{a_t} Q^*(s_t, a_t).$$

With the above policy, at state s_t, the action a_t which can maximize the $Q^*()$ function is chosen. The $Q^*()$ function can be parameterized with GNN models for learning the graph-level representation, as the state s_t is a graph.

Note that an action a_t involves two nodes, which means the search space for an action is $O(N^2)$. This might be too expensive for large graphs. Hence, in (Dai et al., 2018), a decomposition of the action a_t has been proposed as:

$$a_t = (a_t^{(1)}, a_t^{(2)}),$$

where $a_t^{(1)}$ is the sub-action to choose the first node and $a_t^{(2)}$ is the sub-action to choose the second node. Hierarchical $Q^*()$ function is designed to learn the policies for the decomposed actions.

ReWatt

ReWatt (Ma et al., 2020a) is a black-box attacker, which targets on the graph classification task. In this setting, the graph classification model $f_{GNN}(\mathcal{G}; \Theta)$ as defined in Section 5.5.2 is given and fixed. The attacker cannot access any information about the model except querying prediction results for graph samples. It is argued in (Ma et al., 2020a) that the operations such as deleting/adding edges are not unnoticeable enough. Hence, a less noticeable operation, i.e., the rewiring operation, is proposed to attack graphs. A rewiring operation rewires an existing edge from one node to another node, which can be formally defined as below.

Definition 6.1 (Rewiring Operation) A rewiring operation $a = (v_{\text{fir}}, v_{\text{sec}}, v_{\text{thi}})$ involves three nodes, where $v_{\text{sec}} \in \mathcal{N}(v_{\text{fir}})$ and $v_{\text{thi}} \in \mathcal{N}^2(v_{\text{fir}})/\mathcal{N}(v_{\text{fir}})$ with $\mathcal{N}^2(v_{\text{fir}})$ denoting the 2-hop neighbors of the node v_i. The rewiring operation a deletes the existing edge between nodes v_{fir} and v_{sec} and add a new edge between nodes v_{fir} and v_{thi}.

The rewiring operation is theoretically and empirically shown to be less noticeable than other operations such as deleting/adding edges in (Ma et al., 2020a). The constraint space of the ReWatt attack is defined based on the rewiring operation as:

$$\Phi(\mathcal{G}) = \{\mathcal{G}'|\text{if } \mathcal{G}' \text{ can be obtained by applying at most } \Delta \text{ rewiring operations to } \mathcal{G}\},$$

where the budget Δ is usually defined based on the size of the graph as $p \cdot |\mathcal{E}|$ with $p \in (0, 1)$. The attacking procedure is modeled as a Finite Markov Decision Process (MDP), which is defined as:

- **Action:** The action space consists of all the valid rewiring operations as defined in Definition 6.1.
- **State:** The state s_t at the time step t is the intermediate graph $\mathcal{G}_t$, which is obtained by applying one rewiring operation on the intermediate graph $\mathcal{G}_{t-1}$.
- **State Transition Dynamics:** Given an action $a_t = (v_{\text{fir}}, v_{\text{sec}}, v_{\text{thi}})$, the state is transited from the state s_t to state s_{t+1} by deleting an edge between v_{fir} and v_{sec}, and adding an edge to connect v_{fir} with v_{thi} in the state s_t.
- **Reward Design:** The goal of the attacker is to modify the graph such that the predicted label is different from the one predicted for the original graph (or the initial state s_1). Furthermore, the attacker is encouraged to take few actions to achieve the goal so that the modifications to the graph structure are minimal. Hence, a positive reward is granted if the action leads to the change of the label; otherwise, a negative reward is assigned. Specifically, the reward $R(s_t, a_t)$ can be defined as follows:

$$R(s_t, a_t) = \begin{cases} 1 & \text{if } f_{GNN}(s_t; \mathbf{\Theta}) \neq f_{GNN}(s_1; \mathbf{\Theta}); \\ n_r & \text{if } f_{GNN}(s_t; \mathbf{\Theta}) = f_{GNN}(s_1; \mathbf{\Theta}). \end{cases}$$

  where n_r is the negative reward, which is adaptive dependent on the size of graph as $n_r = -\frac{1}{p \cdot |\mathcal{E}|}$. Note that we abuse the definition $f_{GNN}(\mathcal{G}; \mathbf{\Theta})$ a little bit to have the predicted label as its output.
- **Termination:** The attacker stops the attacking process either when the predicted label has been changed or when the resulting graph is not in the constraint space $\Phi(\mathcal{G})$.

Various reinforcement learning techniques can be adopted to learn this MDP. Specifically, in (Ma et al., 2020a), graph neural networks based policy networks are designed to choose the rewiring actions according to the state and the policy gradient algorithm (Sutton et al., 2000) is employed to train the policy networks.

6.3 Graph Adversarial Defenses

To defend against the adversarial attacks on graph-structured data, various defense techniques have been proposed. These defense techniques can be majorly classified to four different categories: 1) graph adversarial training, which incorporates adversarial samples into the training procedure to improve the robustness of the models; 2) graph purification, which tries to detect the adversarial attacks and remove them from the attacked graph to generate a clean graph; 3) graph attention, which identifies the adversarial attacks during the training stage and gives them less attention while training the model; and 4) graph structure learning, which aims to learn a clean graph from the attacked graph while jointly training the graph neural network model. Next, we introduce some representative methods in each category.

6.3.1 Graph Adversarial Training

The idea of adversarial training (Goodfellow et al., 2014b) is to incorporate the adversarial examples into the training stage of the model; hence, the robustness of the model can be improved. It has demonstrated its effectiveness in training robust deep models in the image domain (Goodfellow et al., 2014b). There are usually two stages in adversarial training: 1) generating adversarial attacks, and 2) training the model with these attacks. In the graph domain, the adversarial attackers are allowed to modify the graph structure and/or node features. Hence, the graph adversarial training techniques can be categorized according to the adversarial attacks they incorporate: 1) only attacks on graph structure **A**; 2) only attacks on node features **F**; and 3) attacks on both graph structure **A** and node features **F**. Next, we introduce representative graph adversarial training techniques.

Graph Adversarial Training on Graph Structure

In (Dai et al., 2018), an intuitive and simple graph adversarial training method is proposed. During the training stage, edges are randomly dropped from the input graph to generate the "adversarial attacked graphs". While being simple and not very effective to improve the robustness, this is the first technique to explore the adversarial training on graph-structured data. Later on, a graph adversarial training technique based on the PGD topology attack is proposed. In detail, this adversarial training procedure can be formulated as the following min-max optimization problem:

$$\min_{\boldsymbol{\Theta}} \max_{\mathbf{s} \in \mathcal{S}} -\mathcal{L}(\mathbf{s}; \boldsymbol{\Theta}), \tag{6.13}$$

where the objective $\mathcal{L}(\mathbf{s}; \boldsymbol{\Theta})$ is defined as similar to Eq. (6.4) over the entire training set $\mathcal{V}_l$ as:

$$\mathcal{L}(\mathbf{s}; \boldsymbol{\Theta}) = \sum_{v_i \in \mathcal{V}_l} \ell(f_{GNN}(\mathcal{G}'; \boldsymbol{\Theta})_i, y_i)$$
$$\text{subject to} \quad \|\mathbf{s}\|_0 \leq \Delta, \mathbf{s} \in \{0, 1\}^{N \times (N-1)/2}.$$

Solving the min-max problem in Eq. (6.13) is to minimize the training loss under the perturbation in graph structure generated by the PGD topology attack algorithm. The minimization problem and the maximization problem are processed in an alternative way. In particular, the maximization problem can be solved using the PGD algorithm, as introduced in Section 6.2.2. It results in a continuous solution of $\mathbf{s}$. The non-binary adjacency matrix $\mathbf{A}$ is generated according to the continuous $\mathbf{s}$. It serves as the adversarial graph for the minimization problem to learn the parameters $\mathbf{\Theta}$ for the classification model.

Graph Adversarial Training on Node Features

GraphAT (Feng et al., 2019a) incorporates node features based adversarial samples into the training procedure of the classification model. The adversarial samples are generated by perturbing the node features of the clean node samples such that the neighboring nodes are likely to be assigned to different labels. One important assumption in graph neural network models is that neighboring nodes tend to be similar with each other. Thus, the adversarial attacks on the node features make the model likely to make mistakes. These generated adversarial samples are then utilized in the training procedure in the form of a regularization term. Specifically, the graph adversarial training procedure can be expressed as the following min-max optimization problem:

$$\begin{aligned} &\min_{\mathbf{\Theta}} \mathcal{L}_{train} + \beta \sum_{v_i \in \mathcal{V}} \sum_{v_j \in \mathcal{N}(v_i)} d(f_{GNN}(\mathbf{A}, \mathbf{F} \star \mathbf{r}_i^g; \mathbf{\Theta})_i, f_{GNN}(\mathbf{A}, \mathbf{F}; \mathbf{\Theta})_j); \\ &\mathbf{r}_i^g = \arg \max_{\mathbf{r}_i, \|\mathbf{r}_i\| \leq \epsilon} \sum_{v_j \in \mathcal{N}(v_i)} d(f_{GNN}(\mathbf{A}, \mathbf{F} \star \mathbf{r}_i; \mathbf{\Theta})_i, f_{GNN}(\mathbf{A}, \mathbf{F}; \mathbf{\Theta})_j); \end{aligned} \tag{6.14}$$

where the maximization problem generates the adversarial node features for the nodes, which break the smoothness between the connected nodes. While the minimization problem learns the parameters $\mathbf{\Theta}$, which not only enforce a small training error but also encourage the smoothness between the adversarial samples and their neighbors via the additional regularization term. In Eq. (6.14), $\mathcal{L}_{train}$ is the loss defined in Eq. (5.49), $\mathbf{r}_i \in \mathbb{R}^{1\times d}$ is a row-wise adversarial vector and the operation $\mathbf{F} \star \mathbf{r}_i$ means to add $\mathbf{r}_i$ into the i-th row of $\mathbf{F}$, i.e., adding adversarial noise to the node v_i's features. $f_{GNN}(\mathbf{A}, \mathbf{F} \star \mathbf{r}_i^g; \mathbf{\Theta})_i$ denotes the i-th row of $f_{GNN}(\mathbf{A}, \mathbf{F} \star \mathbf{r}_i^g; \mathbf{\Theta})$, which is the predicted logits for node v_i. The function $d(\cdot, \cdot)$ is the KL-divergence (Joyce, 2011), which measures the distance between the predicted logits. The minimization problem and the maximization problem are processed in an alternative way. Specifically,

Graph Adversarial Training on Graph Structures and Node Features

Given the challenges from the discrete nature of the graph structure $\mathbf{A}$ and the node features $\mathbf{F}$, a graph adversarial training technique proposes to modify the continuous output of the first graph filtering layer $\mathbf{F}^{(1)}$ (Jin and Zhang, n.d.). The method generates adversarial attacks for the first hidden representation

$\mathbf{F}^{(1)}$ and incorporates them into the model training stage. Specifically, it can be modeled as the following min-max optimization problem:

$$\min_{\mathbf{\Theta}} \max_{\zeta \in D} \mathcal{L}_{train}\left(\mathbf{A}, \mathbf{F}^{(1)} + \zeta; \mathbf{\Theta}\right), \tag{6.15}$$

where the maximization problem generates a small adversarial perturbation on the first layer hidden representation $\mathbf{F}^{(1)}$, which indirectly represents the perturbation in the graph structure $\mathbf{A}$ and the node features $\mathbf{F}$. The minimization problem learns the parameters of the model while incorporating the generated perturbation into the learning procedure. ζ is the adversarial noise to be learned and D denotes the constraint domain of the noise, which is defined as follows:

$$D = \{\zeta; \|\zeta_i\|_2 \leq \Delta\};$$

where ζ_i denotes the i-th row of ζ and Δ is a predefined budget. Note that in Eq. (6.15), $\mathcal{L}_{train}\left(\mathbf{A}, \mathbf{F}^{(1)} + \zeta; \mathbf{\Theta}\right)$ is overloaded to denote a similar loss as Eq. (5.49) except that it is based on the perturbed hidden representation $\mathbf{F}^{(1)} + \zeta$. Similar to other adversarial training techniques, the minimization problem and the maximization problem are processed in an alternative way.

6.3.2 Graph Purification

Graph purification based defense techniques have been developed to defend against the adversarial attacks on graph structure. Specifically, these methods try to identify adversarial attacks in a given graph and remove them before using the graph for model training. Hence, most of the graph purification methods can be viewed as performing pre-processing on graphs. Next, we introduce two defense techniques based on graph purification.

Removing Edges with Low Feature Similarity

Empirical explorations show that many adversarial attack methods (e.g., nettack and IG-FGSM) tend to add edges to connect nodes with significantly different node features (Wu et al., 2019; Jin et al., 2020a). Similarly, when removing edges, these attack methods tend to remove the edges between nodes with similar features. Hence, based on these observations, a simple and efficient approach is proposed in (Wu et al., 2019), which tries to remove the edges between nodes with very different features. More specifically, a scoring function is proposed to measure the similarity between the node features. For example, for binary features, the Jaccard similarity (Tan et al., 2016) is adopted as the scoring function. The edges with scores that are smaller than a threshold are then removed from the graph. The pre-processed graph is then employed for training the graph neural network models.

Low-rank Approximation of Adjacency Matrix

Empirical studies are carried out to analyze the adversarial perturbations generated by nettack (Entezari et al., 2020; Jin et al., 2020a). It turns out that Netttack tends to perturb the graph structure to increase the adjacency matrix's rank. It is argued that the number of low-value singular values of the adjacency matrix is increased. Hence, Singular Value Decomposition(SVD) based pre-processing method is proposed in (Entezari et al., 2020) to remove the adversarial perturbation added into the graph structure. Specifically, given an adjacency matrix $\mathbf{A}$ of a graph, SVD is used to decompose it, and then only the top-k singular values are kept to reconstruct (approximate) the adjacency matrix. The reconstructed adjacency matrix is then treated as the purified graph structure and utilized to train the graph neural network models.

6.3.3 Graph Attention

Instead of removing the adversarial attacks from the graph as the graph purification-based methods, the graph attention-based methods aim to learn to focus less on the nodes/edges affected by the adversarial attacks in the graph. The graph attention based defense techniques are usually end-to-end. In other words, they include the graph attention mechanism as a building component in the graph neural network models. Next, we introduce two attention based defense techniques.

RGCN: Modelling Hidden Representations with Gaussian Distribution

To improve the robustness of the graph neural network models, instead of plain vectors, multivariate Gaussian distribution is adopted to model the hidden representations in (Zhu et al., 2019a). The adversarial attacks generate perturbations on the graph structure, which, in turn, cause abnormal effects on the node representations. While the plain-vector based hidden representations cannot adapt themselves to the adversarial impacts, the Gaussian distribution based hidden representations can absorb the effects caused by the adversarial attacks and thus can lead to more robust hidden representations. Furthermore, a variance-based attention mechanism is introduced to prevent the adversarial effects from propagation across the graph. Specifically, the nodes affected by adversarial attacks typically have large variances as the attacks tend to connect nodes with very different features and/or from different communities. Hence, when performing neighbor information aggregation to update node features, less attention is assigned to those neighbors with large variances to prevent the adversarial effects from propagation. Next, we describe the details of RGCN-Filter – the graph filter built upon the intuitions above.

RGCN-Filter is built upon the GCN-Filter as described in Eq. (5.22). For the ease of description, we recall Eq. (5.22) as follows:

$$\mathbf{F}'_i = \sum_{v_j \in \mathcal{N}(v_i) \cup \{v_i\}} \frac{1}{\sqrt{\tilde{\mathbf{d}}_i \tilde{\mathbf{d}}_j}} \mathbf{F}_j \mathbf{\Theta},$$

where $\tilde{d}_i = \tilde{\mathbf{D}}[i, i]$. Instead of plain vectors, RGCN-Filter utilizes Gaussian distributions to model the node representations. For the node v_i, its representation is denoted as:

$$\mathbf{F}_i \sim \mathcal{N}(\boldsymbol{\mu}_i, \mathrm{diag}(\boldsymbol{\sigma}_i)),$$

where $\mu_i \in \mathbb{R}^d$ is the mean of the representations and $diag(\sigma_i) \in \mathbb{R}^{d \times d}$ is the diagonal variance matrix of the representations. When updating the node representations, it has two aggregation processes on the mean and the variance of the representations. In addition, an attention mechanism based on the variance of representations is introduced to prevent the adversarial effects from propagating across the graph. Specifically, for nodes with larger variances, smaller attention scores are assigned. The attention score for node v_i is modeled through a smooth exponential function as:

$$\boldsymbol{a}_i = \exp\left(-\gamma \boldsymbol{\sigma}_i\right),$$

where γ is a hyperparameter. With the definition of the Gaussian based representations and the attention scores, the update process for the representation of node v_i can be stated as:

$$\mathbf{F}'_i \sim \mathcal{N}(\boldsymbol{\mu}'_i, \mathrm{diag}(\boldsymbol{\sigma}'_i)),$$

where

$$\boldsymbol{\mu}'_i = \alpha\left(\sum_{v_j \in \mathcal{N}(v_i) \cup \{v_i\}} \frac{1}{\sqrt{\tilde{\mathbf{d}}_i \tilde{\mathbf{d}}_j}} \left(\boldsymbol{\mu}_j \odot \boldsymbol{a}_j\right) \mathbf{\Theta}_\mu\right);$$
$$\boldsymbol{\sigma}'_i = \alpha\left(\sum_{v_j \in \mathcal{N}(v_i) \cup \{v_i\}} \frac{1}{\tilde{\mathbf{d}}_i \tilde{\mathbf{d}}_j} \left(\boldsymbol{\sigma}_j \odot \boldsymbol{a}_j \odot \boldsymbol{a}_j\right) \mathbf{\Theta}_\sigma\right).$$

Here α denotes non-linear activation functions, $\odot$ is the Hadamard multiplication operator, $\mathbf{\Theta}_\mu$ and $\mathbf{\Theta}_\sigma$ are learnable parameters to transform the aggregated information of mean and variance, respectively.

PA-GNN: Transferring Robustness From Clean Graphs

Instead of penalizing the affected nodes as RGCN, PA-GNN (Tang et al., 2019) aims to penalize the adversarial edges for preventing the adversarial effects from propagation through the graph. Specifically, it aims to learn an attention mechanism that can assign low attention scores to adversarial edges. However, typically, we do not have knowledge about the adversarial edges. Hence, PA-GNN aims to transfer this knowledge from clean graphs where adversarial attacks can be generated to serve as supervision signals to learn the desired attention scores.

The PA-GNN model is built upon the graph attention network as described in Eq. (5.27), which can be written as:

$$\mathbf{F}_i' = \sum_{v_j \in \mathcal{N}(v_i) \cup \{v_i\}} a_{ij} \mathbf{F}_j \mathbf{\Theta}, \tag{6.16}$$

where a_{ij} denotes the attention score for aggregating information from node v_j to node v_i through the edge e_{ij}. Intuitively, we desire the attention scores of the adversarial edges to be small so that the adversarial effects can be prevented from propagation. Assume that we know a set of adversarial edges, which is denoted as $\mathcal{E}_{ad}$, and the set of the remaining "clean" edges can be denoted as $\mathcal{E}/\mathcal{E}_{ad}$. To ensure that the attention scores for the adversarial edges are small, the following term can be added to the training loss to penalize the adversarial edges.

$$\mathcal{L}_{\text{dist}} = -\min\left(\eta, \underset{\substack{e_{ij} \in \mathcal{E}/\mathcal{E}_{ad} \\ 1 \le l \le L}}{\mathbb{E}} a_{ij}^{(l)} - \underset{\substack{e_{ij} \in \mathcal{E}_{ad} \\ 1 \le l \le L}}{\mathbb{E}} a_{ij}^{(l)}\right)$$

where $a_{ij}^{(l)}$ is the attention score assigned to edge e_{ij} in the l-th graph filtering layer, L is the total number of graph filtering layers in the model, and η is a hyper parameter controlling the margin between the two expectations. The expectations of the attention coefficients are estimated by their empirical means as:

$$\underset{\substack{e_{ij} \in \mathcal{E} \setminus \mathcal{E}_{ad} \\ 1 \le l \le L}}{\mathbb{E}} a_{ij}^{(l)} = \frac{1}{L|\mathcal{E} \setminus \mathcal{E}_{ad}|} \sum_{l=1}^{L} \sum_{e_{ij} \in \mathcal{E} \setminus \mathcal{E}_{ad}} a_{ij}^{(l)}$$

$$\underset{\substack{e_{ij} \in \mathcal{E}_{ad} \\ 1 \le l \le L}}{\mathbb{E}} a_{ij}^{(l)} = \frac{1}{L|\mathcal{E}_{ad}|} \sum_{l=1}^{L} \sum_{e_{ij} \in \mathcal{E}_{ad}} a_{ij}^{(l)},$$

where $|\cdot|$ denotes the cardinality of a set. To train the classification model while assigning lower attention scores to the adversarial edges, we combine the loss $\mathcal{L}_{dist}$ with the semi-supervised node classification loss $\mathcal{L}_{train}$ in Eq. (5.49) as:

$$\min_{\mathbf{\Theta}} \mathcal{L} = \min_{\mathbf{\Theta}} (\mathcal{L}_{train} + \lambda \mathcal{L}_{dist}) \tag{6.17}$$

where λ is a hyper-parameter balancing the importance between the two types of loss. So far, the set of adversarial edges $\mathcal{E}_{ad}$ is assumed to be known, which is impractical. Hence, instead of directly formulating and optimizing Eq. (6.17),

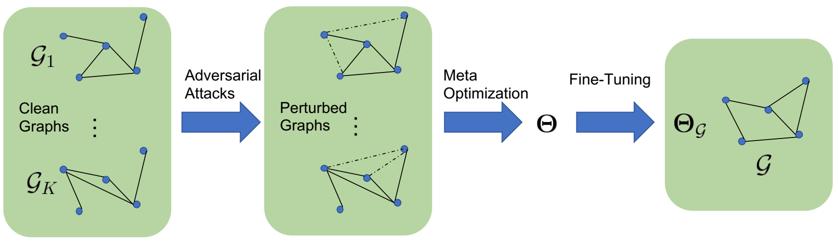

Figure 6.1 The overall framework of PA-GNN

we try to transfer the ability of assigning low attention scores to adversarial edges from those graphs with known adversarial edges. To obtain the graphs with known adversarial edges, we collect clean graphs from similar domains as the given graph, and apply existing adversarial attacks such as *metattack* to generate attacked graphs. Then, we can learn the ability from these attacked graphs and transfer it to the given graph. Next, we first briefly discuss the overall framework of PA-GNN and then detail the process of learning the attention mechanism and transferring its ability to the target graph. As shown in Figure 6.1, given a set of K clean graphs denoted as $\{\mathcal{G}_1, \cdots, \mathcal{G}_K\}$, we use existing attacking methods such as metattack to generate a set of adversarial edges $\mathcal{E}^i_{ad}$ for each graph. Furthermore, the node set $\mathcal{V}^i$ in each graph is split into the training set $\mathcal{V}^i_l$ and the test set $\mathcal{V}^i_u$. Then, we try to optimize the loss function in Eq. (6.17) for each graph. Specifically, for the graph $\mathcal{G}_i$, we denote its corresponding loss as $\mathcal{L}_i$. As inspired by the meta-optimization algorithm MAML (Finn et al., 2017), all graphs share the same initialization $\mathbf{\Theta}$ and the goal is to learn these parameters $\mathbf{\Theta}$ that can be easily adapted to learning the task on each graph, separately. As shown in Figure 6.1, the ideal shared initialization parameters $\mathbf{\Theta}$ are learned through meta-optimization, which we will detail later. These shared parameters $\mathbf{\Theta}$ are considered to carry the ability of assigning lower attention scores to the adversarial edges. To transfer this ability to the given graph $\mathcal{G}$, we use the shared parameters $\mathbf{\Theta}$ as the initialization parameters to train the graph neural network model on graph $\mathcal{G}$ and the obtained fine-tuned parameters are denoted as $\mathbf{\Theta}_{\mathcal{G}}$. Next, we describe the meta-optimization algorithm adopted from MAML to learn the optimal shared parameters $\mathbf{\Theta}$.

The optimization process first adapts (fine-tunes) the parameters $\mathbf{\Theta}$ to each graph $\mathcal{G}_i$ by using the gradient descent method as:

$$\mathbf{\Theta}'_i = \mathbf{\Theta} - \alpha \nabla_{\mathbf{\Theta}} \mathcal{L}^{tr}_i(\mathbf{\Theta}),$$

where $\mathbf{\Theta}'_i$ is the specific parameters for the learning task on the graph $\mathcal{G}_i$ and

$\mathcal{L}_i^{tr}$ denotes the loss in Eq.(6.17) evaluated on the corresponding training set $\mathcal{V}_l^i$. The test sets of all the graphs $\{\mathcal{V}_u^1, \ldots, \mathcal{V}_u^K\}$ are then used to update the shared parameters $\mathbf{\Theta}$ such that each of the learned classifiers can work well for each graph. Hence, the objective of the meta-optimization can be summarized as:

$$\min_{\mathbf{\Theta}} \sum_{i=1}^{K} \mathcal{L}_i^{te}\left(\mathbf{\Theta}_i'\right) = \min_{\mathbf{\Theta}} \sum_{i=1}^{K} \mathcal{L}_i^{te}\left(\theta - \alpha \nabla_{\mathbf{\Theta}} \mathcal{L}_i^{tr}(\mathbf{\Theta})\right),$$

where $\mathcal{L}_i^{te}\left(\mathbf{\Theta}_i'\right)$ denotes the loss in Eq. (6.17) evaluated on the corresponding test set $\mathcal{V}_u^i$. The shared parameters $\mathbf{\Theta}$ can be updated using SGD as:

$$\mathbf{\Theta} \leftarrow \mathbf{\Theta} - \beta \nabla_{\mathbf{\Theta}} \sum_{i=1}^{K} \mathcal{L}_i^{te}\left(\mathbf{\Theta}_i'\right).$$

Once the shared parameters $\mathbf{\Theta}$ are learned, they can be used as the initialization for the learning task on the given graph $\mathcal{G}$.

6.3.4 Graph Structure Learning

In Section 6.3.2, we introduced the graph purification based defense techniques. They often first identify the adversarial attacks and then remove them from the attacked graph before training the GNN models. Those methods typically consist of two stages, i.e., the purification stage and the model training stage. With such a two-stage strategy, the purified graphs might be sub-optimal to learn the model parameters for down-stream tasks. In (Jin et al., 2020b), an end-to-end method, which jointly purifies the graph structure and learns the model parameters, is proposed to train robust graph neural network models. As described in Section 6.3.2, the adversarial attacks usually tend to add edges to connect nodes with different node features and increase the rank of the adjacency matrix. Hence, to reduce the effects of the adversarial attacks, Pro-GNN (Jin et al., 2020b) aims to learn a new adjacency matrix $\mathbf{S}$, which is close to the original adjacency matrix $\mathbf{A}$, while being low-rank and also ensuring feature smoothing. Specifically, the purified adjacency matrix $\mathbf{S}$ and the model parameters can be learned by solving the following optimization problem:

$$\min_{\mathbf{\Theta},\mathbf{S}} \mathcal{L}_{train}(\mathbf{S}, \mathbf{F}; \mathbf{\Theta}) + \|\mathbf{A} - \mathbf{S}\|_F^2 + \beta_1 \|\mathbf{S}\|_1 + \beta_2 \|\mathbf{S}\|_* + \beta_3 \cdot tr(\mathbf{F}^T \mathbf{L} \mathbf{F}), \qquad (6.18)$$

where the term $\|\mathbf{A} - \mathbf{S}\|_F^2$ is to make sure that the learned matrix $\mathbf{S}$ is close to the original adjacency matrix; the L_1 norm of the learned adjacency matrix $\|\mathbf{S}\|_1$ allows the learned matrix $\mathbf{S}$ to be sparse; $\|\mathbf{S}\|_*$ is the nuclear norm to ensure that the learned matrix $\mathbf{S}$ is low-rank; and the term $tr(\mathbf{F}^T \mathbf{L} \mathbf{F})$ is to force the feature smoothness. Note that the feature matrix $\mathbf{F}$ is fixed, and the term $tr(\mathbf{F}^T\mathbf{L}\mathbf{F})$ force the Laplacian matrix $\mathbf{L}$, built upon $\mathbf{S}$, to ensure that the features are smooth. The hyper-parameters β_1,β_2 and β_3 control the balance between these terms. The matrix $\mathbf{S}$ and the model parameters can be optimized alternatively as:

- **Update $\boldsymbol{\Theta}$:** We fix the matrix $\mathbf{S}$ and remove the terms that are irrelevant to $\mathbf{S}$ in Eq. (6.18). The optimization problem is then re-formulated as:
$$\min_{\boldsymbol{\Theta}} \mathcal{L}_{train}(\mathbf{S}, \mathbf{F}; \boldsymbol{\Theta}).$$
- **Update S:** We fix the model parameters $\boldsymbol{\Theta}$ and optimize the matrix $\mathbf{S}$ by solving the following optimization problem:
$$\min_{\mathbf{S}} \mathcal{L}_{train}(\mathbf{S}, \mathbf{F}; \boldsymbol{\Theta}) + ||\mathbf{A} - \mathbf{S}||_F^2 + \alpha||\mathbf{S}||_1 + \beta||\mathbf{S}||_* + \lambda \cdot tr(\mathbf{F}^T\mathbf{L}\mathbf{F}).$$

6.4 Conclusion

In this chapter, we focus on the robustness of the graph neural networks, which is critical for applying graph neural network models to real-world applications. Specifically, we first describe various adversarial attack methods designed for graph-structured data including white-box, gray-box and black-box attacks. They demonstrate that the graph neural network models are vulnerable to deliberately designed unnoticeable perturbations on graph structures and/or node features. Then, we introduced a variety of defense techniques to improve the robustness of the graph neural network models including graph adversarial training, graph purification, graph attention and graph structure learning.

6.5 Further Reading

The research area of robust graph neural networks is still fast evolving. Thus, a comprehensive repository for graph adversarial attacks and defenses has been built (Li et al., 2020a). The repository enables systematical experiments on existing algorithms and efficient new algorithm development. An empirical study has been conducted based on the repository (Jin et al., 2020a). It provides deep insights about graph adversarial attacks and defenses that can deepen our knowledge and foster this research field. In addition to the graph domain, there are adversarial attacks and defenses in other domains such as images (Yuan et al., 2019; Xu et al., 2019b; Ren et al., 2020) and texts (Xu et al., 2019b; Zhang et al., 2020).