Characterizing Adversarial Edges Targeting on Graph Neural Networks

Anonymous Author(s)

ABSTRACT

Deep neural networks (DNNs) on graph-structured data have been widely applied in various applications. However, recent studies have shown that DNNs are vulnerable to adversarial attacks which aim to mislead machine learning models by adding a small magnitude of perturbation. In particular, the first several attacks against graph neural networks (GNNs) have been proposed by adding/deleting edges, which have caused great concerns in safety-critical scenarios. Detecting these attacks are challenging due to the unnoticeable perturbation. In this paper, we propose the first detection mechanism against these adversarial attacks. We propose a novel graph generation approach together with link prediction to detect when a small number of adversarial edges are added. The generative models are trained on our sampled sub-graphs which have low probability of containing adversarial edges (small perturbation) based on the law of large numbers. In order to handle cases where a large number of edges are allowed to be manipulated, we propose a set of novel features to perform outlier detection. Finally we propose a general detection pipeline EDoG to detect adversarial edges without knowledge of attack types. Extensive experiments are conducted to show that EDoG can achieve AUC above 70% against the three considered attack strategies on both Cora and Citeseer datasets without any knowledge of the attack types, and above 80% with knowledge of attack type. EDoG generally outperforms traditional malicious edge detection baselines. We also provide in-depth analysis for different attack strategies and corresponding suitable detection methods. Our results shed light on several principles for detecting adversarial attacks on graph structured data.

KEYWORDS

Graph neural networks, Adversarial edges, Detection methods

ACM Reference Format:

Anonymous Author(s). 2019. Characterizing Adversarial Edges Targeting on Graph Neural Networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2019).* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Graph neural networks (GNNs) have been widely applied in many real-world tasks, such as drug screening [9, 12], protein structure prediction [18], and social network analysis [21, 37]. However, recent studies on adversarial machine learning, in which carefully crafted instances are able to mislead machine learning models to make an arbitrarily incorrect prediction, have raised great concerns about the robustness of machine learning models. Given the wide applications of GNNs, researchers recently have proposed three types of attacks targeting on GNNs by adding/deleting edges within a victim graph to fool the node classification or graph classification tasks [10, 49, 50].

Detecting such adversarial attacks targeting on GNNs involves several challenges compared with traditional detection approaches. First, such adversarial attacks focus on local graph properties and aim to create "unnoticeable" perturbation. Therefore, the manipulation of a small amount of local edges is not obvious enough to be detected by traditional Sybil detection methods [4]. Second, existing defense/detection methods against adversarial behaviors on machine learning models are not easy to be applied for detecting malicious attacks on GNNs. For instances, *adversarial training* is one of the most effective defense methods that retrain the neural networks with generated adversarial instances to improve the model robustness [17]. However, it is very challenging to inject malicious edges and retrain the GNNs due to the large manipulation space in graph. *Robust generative models* are proposed to mitigate adversarial perturbation via denoising autoencoders and Generative Adversarial Networks (GAN) respectively [16, 32, 39], while subtle adversarial perturbation in graph-structured data is hard to be directly removed through generative models. Third, the perturbation on the graph is more diverse than that for images which require a new understanding of such adversarial behaviors in order to identify them. For instance, adding adversarial edges to the node of higher degree will induce different adversarial patterns (adversarial edges in this case is more likely to be outliers) compared with adding them to the node of lower degree.

Given these challenges, in this paper we propose several detection methods against adversarial attacks on GNNs, including Link prediction based method (LP), Graph Generation based method (GGD), and Outlier detection based method (OD). Our proposed detection pipeline are shown in figure 1. Here we mainly consider three types of attacks: (1) adding only one adversarial edge [10]; (2) the number of added adversarial edges is allowed to be up to the degree of a chosen victim node [49]; (3) the number of added adversarial edges is allowed to be up to certain percent (5%) of the number of the edges in the original graph [50]. We found that adversarial edges in the first type of attacks can be characterized by graph generative models which are trained with large number of benign graphs. Therefore, we propose a novel graph generative model based detection method GGD to detect such adversarial edges. In particular, we first sample several sub-graphs. Since the original graph contains very small number of adversarial edges, based on the *law of large numbers*, these sub-graphs would contain only benign edges with high probability. In the meantime, we propose LP

KDD 2019, August 4-8, 2019, Anchorage, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

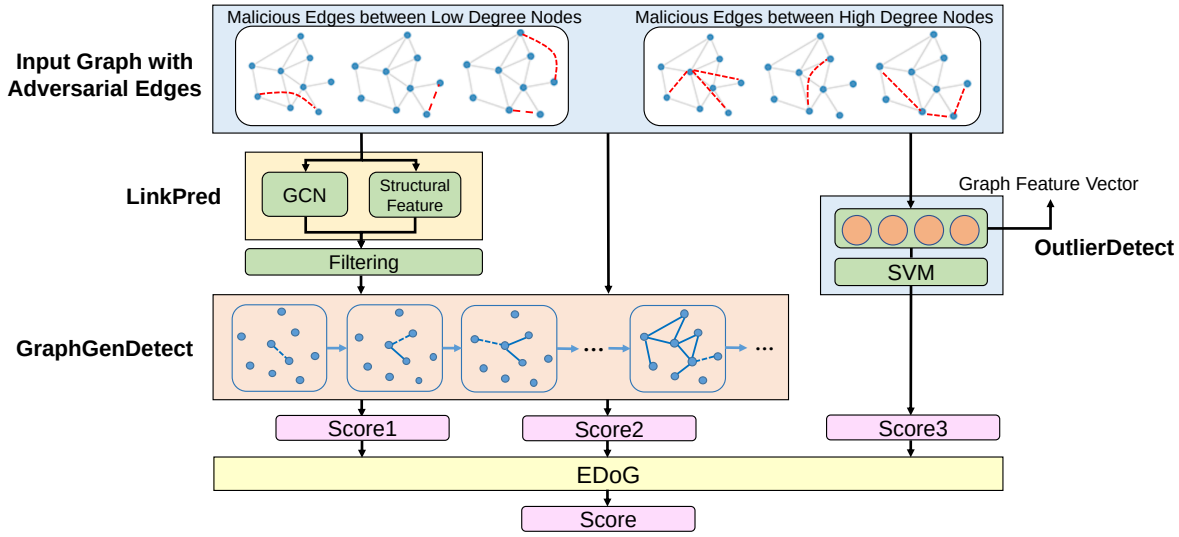

Figure 1: Illustration of the proposed general detection pipeline EDoG.

to train several link prediction models to assist the graph generative model to ensure that the generative models are trained on "clean" edges. On the other hand, when larger amount of adversarial edges are allowed, the sampled sub-graphs would contain more adversarial edges and therefore the trained graph generative models and link prediction models are not accurate enough to distinguish the added adversarial edges. In this scenario, we found that the added adversarial edges are more likely to appear as "outliers", while the single adversarial edge would not. As a result, we propose a list of intrinsic features that can be leveraged to perform outlier detection (OD), together with GGD and LP. We empirically show that OD can achieve detection AUC above 90% for the cases when the victim nodes have high degree (larger than 10), which shows if we have knowledge about the attack type it is possible to achieve nearly perfect detection rate. To generalize our pipeline without requiring knowledge of the attack types, we propose a general ensemble pipeline, EDoG (Edge Detection of Graph), to select the detection mechanism based on node degree. Our results show that the proposed general pipeline outperforms other baseline methods [36, 38] significantly. In addition, we provide in-depth analysis for different types of attacks and the detection performance on victim nodes with various properties. In summary, our contributions are listed below:

- We propose several detection methods to detect adversarial edges added by recent three attack strategies against GNNs. To the best of our knowledge, this is the first detection mechanism against the state-of-the-art GNNs based attacks.
- We propose a novel generation model for graph data and a filter-and-sample framework to train the generative model for detection.
- We provide several insightful features that can be leveraged to perform outlier detection against certain attacks.
- We explore attacks targeting on graphs with various properties and provide in-depth analysis for different attack strategies and corresponding detection methods.
- Extensive experiments are conducted on Cora, Citeseer and synthetic graph data with controlled properties to detect the state-of-the-art attacks, demonstrating the effectiveness of the proposed detection pipeline. The detection AUC can reach above 70% without any knowledge of attack types and 80% when we know the attack type. In both scenarios, the proposed EDoG approachoutperforms several baseline methods.

2 BACKGROUND

In this section, we will introduce the background on graph neural networks, the threat model, and our detection goal.

2.1 Graph Neural Networks

Suppose we are given a graph data $G = (V, E, X)$, where $V = \{v_1, v_2, \ldots\}$ denotes the set of nodes, $E = \{e_1, e_2, \ldots\}$, $e_i \in V \times V$ denotes the set of edges and $X = (\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_{|V|})$ represents the feature vector of each node. Graph Neural Networks (GNNs) is a class of deep learning models over the graph data G. A GNN based model calculates an embedding vector $\boldsymbol{\theta}_u$ of each node $u \in V$ via iteratively aggregating information of itself and its neighbours:

$$\boldsymbol{\theta}_u^{(k)} = f\left(\boldsymbol{x}_u, \boldsymbol{\theta}_u^{(k-1)}, \{\boldsymbol{x}_v, \boldsymbol{\theta}_v^{(k-1)}\}_{v \in \mathcal{N}(u)}\right),$$

where $\mathcal{N}(u)$ denotes the neighbours of u in the graph. The node features $\boldsymbol{x}_u$ serves as the initial embedding $\boldsymbol{\theta}_u^{(0)}$.

Many GNN models have been proposed and achieved good performance on various tasks, such as Graph Convolutional Networks (GCN) [27] and Structure2Vec [9]. In this paper, we will focus on the GCN model. Let $\Theta^{(k)} = (\boldsymbol{\theta}_1^{(k)}, \boldsymbol{\theta}_2^{(k)}, \ldots, \boldsymbol{\theta}_{|V|}^{(k)})$ be the matrix of all node embedding vectors at step k. For GCN, the aggregation function is calculated as:

$$\Theta^{(k)} = \sigma\left(\hat{A}\Theta^{(k-1)}W^{(k)}\right)$$

$$\hat{A} = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}$$

where $\tilde{A} = A + I_N$, and A is the adjacency matrix with I_N the identity matrix. $D_{ii} = \sum_j \tilde{A}_{ij}$, σ represents the non-linear activation function, and $W^{(k)}$ is the trainable parameters in the k-th layer.

Node classification In a node classification task over graph data, each node v_i has a label $y_i \in \mathcal{Y}$, but we only have access to a small subset of the true labels, i.e., $L_{train} : V_{train} \rightarrow \mathcal{Y}$ where $V_{train} = \{v_{i_0}, v_{i_1}, \ldots\} \subset V$ is the set of nodes which we know the true labels. We also have a set of nodes V_{infer} which we would like to infer about their labels, and $V_{train} \cap V_{infer} = \emptyset$. Given L_{train}, we would like to infer the labels of V_{infer}. That is to say, we seek for a model to give a prediction $\hat{y}_i \in \mathcal{Y}$ to each of the node v_i, and we would like to maximize the classification accuracy, i.e., $\sum_{v_i \in V_{infer}} \mathbf{1}\{\hat{y}_i = y_i\}$.

When performing the node classification task, a GNN model f first calculates the embedding $\boldsymbol{\theta}_u$ for each node, which will then be used to calculate the probability vector

$$s_u = softmax(W^{out}\boldsymbol{\theta}_u)$$

indicating the probability of each class that node u belongs to. During the training process, the goal is to minimize the cross-entropy loss for the prediction of nodes in V_{train}. During the evaluation process, the predicted class of each node u is given by $\hat{y}_u = \arg\max_y (s_u)_y$.

2.2 Adversarial Attacks on Graph-structured Data

Recent work show that the GNN-based classification model can be fooled by adversarial attacks on given node [10, 49, 50]. Suppose we have a node $v_{vic} \in V_{unk}$ which we call the *victim node* and our model can correctly predict the label $y_{vic} = y_{vic}$. An attacker's goal is to make small modification to the graph such that the prediction of the node is incorrect. Formally, an attacker provides a graph G' such that the prediction $y'_{vic} \neq c_{vic}$. There are mainly two categories of the attack type: 1) **Feature attack**: $G' = (V, E, X')$. This means that an attacker made small modification to the feature vectors of victim nodes. 2) **Structure attack**: $G' = (V, E', X)$. This means that the attacker can add or delete several edges in the graph.

Recently, [50] proposes another approach which does not require a victim node and focuses on reducing the classification accuracy over the entire graph, which is called **meta attack**. In meta attack, they allow the number of perturbations to be at most 5% of the total number of edges in the graph.

3 OVERVIEW OF DETECTION

In this section, we will first summarize the considered attack models and corresponding insights into different adversarial behaviors. We then provide the high level overview and intuitions of our proposed detection approaches, as well as the final detection pipeline EDoG.

3.1 Attack Models

We focus on the recent attacks targeting on GNNs proposed in [10], [49] and [50]. In particular, four types of attack are considered:

- Single-edge attack. This attack is proposed by [10] where an attacker can add only one malicious edge to the graph.
- Multi-edge direct attack. This attack is proposed by [49] where an attacker is allowed to add several edges to the graph. The malicious edges are all connected to the victim node and the number of malicious edges should not exceed the degree of victim node.
- Multi-edge indirect attack. This is similar to the Multi-edge direct attack except that none of the malicious edges are directly connected to the victim node.
- Meta attack. This attack is proposed by [50]. The attacker can add many edges to the graph. There is no 'victim node' in the attack and the goal of the attacker is to cause the GNN to mis-classify as many nodes as possible. In the paper they added 5% edges as malicious edges.

3.2 Goal of Adversarial Edge Detection

In this paper, we aim to develop a model to detect the malicious edges added by an attacker. That is to say, we assume that: a) the feature of the input graph is benign;b) in the structure attack $E \subset E'$, which means attackers will add adversarial edges. We make this assumption because: first the detection of continuous malicious data (feature attack) has been investigated by many previous researchers [32, 33, 42], while detection of structural attacks against GNNs has not been explored; second, adding edge is usually a cheaper attack approach than deleting edge. For instance, in an undirected citation network one can easily add edge by citing others but cannot delete edges if other papers have cited it.

Specifically, the GNN model is provided with a malicious graph $G' = (V, E', X)$. We know that some of the edges are malicious, but we do not have any other information about the attack (e.g. the information of victim nodes or the attack strategies). In our detection mechanism, the goal is for any given G', we will generate a score s_j for each edge $e_j \in E'$ indicating how likely the edge may be a malicious edge. Therefore, after training to select a proper threshold, we can either identify adversarial edges directly or help to set priorities for further inspection of "suspicious" edges.

3.3 Malicious Links between Low-degree Nodes: Link Prediction and Generative Model

Here we consider the attack model where the attacker only adds a small number of adversarial edges, which is a sufficient condition for victim nodes with low degree. This is because for all the attacks we considered, the maximum number of allowed adversarial edges is up to the degree of victim node. When an attacker only adds a very small number of edges to ensure the perturbation is "unnoticeable", we would like to tentatively assume that the perturbation of malicious edges is small enough to be neglected - any algorithm, except for the target GNN based models, applied over the malicious graph G' will return approximately the same result as if it is applied over G. Thus, an intuitive approach is that we can train a link prediction model using G'. Ideally, the link prediction model would behave as if it were trained using G, so it will predict the node pairs with edges as high scores and the pairs without edges as low scores. The malicious edges, however, do not exist in the benign graph. So the scores of them will become low. Thus, after applying the link prediction algorithm we can check the scores for the edges in G'. Those with low scores should be considered likely to become malicious. In this link prediction based approach, selecting features that would focus more on global structures are more helpful, and

therefore we propose to use the GCN based structural features for prediction. We will discuss the used features in detail in the next section.

In addition, in order to better capture the global structure information of the original graph data, we also propose generative models to better approximate the original link distribution. A key step for training generative models is to ensure that the data are *clean* (not malicious) to avoid being "poisoned". One approach for reducing the affect of malicious edges would be to sample sub-graphs from the large graph. The intuition is that if we randomly sample many small sub-graphs from a large graph, each edge will only appear in a small proportion of the sub-graphs. Thus, most sub-graphs will contain no malicious edges while preserving the information of the original graph. For example, if we sample the graphs by extracting the two-hop neighbour for each node, it turns out that the more than 99% sub-graphs don't contain malicious edges in single-edge attack on average, and more than 90% for multi-edge attack and meta-attack. Therefore, we can train some generative models over the sub-graphs to learn a good distribution approximation of the original graph, and then leverage it to detect abnormal edges. The naive link prediction algorithm is not suitable for training over sub-graphs in two aspects: first, many link prediction algorithms are based on feature extraction over the entire graph; second, link prediction algorithms tend to have relatively small model capacity to capture the pattern of entire sub-graphs. Therefore, we can only train the proposed graph generative models based on deep neural networks on sub-graphs. Based on the generative models, we could see which edges are the least likely to be generated and these edges are highly likely to be malicious. We find that such generative models are good at discovering patterns from sub-graphs and detect malicious edges.

3.4 Malicious Links between High-degree Nodes: Statistical Outliers

It turns out that both link prediction and graph generation approach may sometimes fail when applied over multi-edge direct attack, especially when the node degree is large. We attribute this to a principle of the *collective power of malicious edges* which can be understood as: when there are many malicious edges connecting to one node, they confirm the legitimacy of each other mutually. For example, suppose one node is originally connected to five nodes in class 1. If an attacker adds just one malicious edge that connects it to a node in class 2, this edge will seem abnormal and is easily detected. However, if the attacker instead adds five malicious edges in class 2, the legitimacy of each malicious edge will supported by the rest, and they will all be judged to be benign.

Under this circumstance, the neighbour of the victim node in G' should contain a number of different classes, while the classes of neighbour of a benign node should be quite uniform. As a result, we may calculate several features indicating the information of the neighbourhood of edges, e.g. number of different classes appearing in the neighbourhood of the edge. We could expect that the feature vector of a malicious edge would be different from those of benign edges. Therefore, we could build an outlier detection model over all the edges and consider the outliers as malicious.

Finally, when there is no knowledge about attack types, the node degree information can help to point out whether the potential adversarial edge is more likely to be an outlier or not. Therefore, we are able to select the corresponding detection model and develop a general pipeline EDoG to detect adversarial edges on any given graph.

4 DETECTION APPROACHES

In this section, we will introduce the proposed primitive methods for detection, followed by the proposed detection pipeline.

4.1 Baseline-1 Approach (BL-1)

Our first baseline is the anomaly link detection approach as proposed in [36]. Their approach follows the intuition in Section 3.3 to train a link prediction-based algorithm. For each node pair (u, v) they calculate five features, including the similarity of neighbour, the number of common neighbour, the distance between two nodes, the preferential attachment of two nodes and the similarity of the node features of u and v. Then they train a logistic regression classifier over the feature vectors of the node pairs, where the ground truth label of node pairs with edge is 1, otherwise 0. After the model is trained, we calculate the probability of link for each existing edge. The smaller the probability is, the more likely that edge is a malicious one.

4.2 Baseline-2 Approach (BL-2)

Another baseline is also an anomaly link discovery method described in [38], in which they used a high-order heuristic named Katz index [28] to measure the connectivity of each node pair (u, v) as

$$Katz(u, v) = \sum_{l=1}^{\infty} \beta^l |walks^l(u, v)|$$

where β is damping factor to assign more weight to shorter walks and $walks^l(u, v)$ is the set of random walks with the length of l between u and v. The intuition behind using this measure is that compared with the normal links, malicious links often have small Katz index values. Then we can calculate the Katz index for each node pair and used the softmax function for normalization. After that, the value assigned for each node pair is bounded between 0 and 1, and we use this value as the probability for the existence of the link between the nodes. The smaller the probability is, the more likely that edge is a malicious one.

4.3 LinkPred (LP)

Our link prediction algorithm would like to improve the Baseline-1 algorithm by GNN techniques. Using GNN, we may discover some latent feature space in addition to the features proposed in [36]. In practice, we first adopt a two-layer graph convolutional network to calculate the node embedding vector $\boldsymbol{\theta}_u$ for each node. Then for each pair node (u, v) we use a bilinear hidden layer to calculate a hidden feature

$$f_{u,v}^{GNN} = \sigma(\boldsymbol{\theta}_u^{\intercal} W_{edge} \boldsymbol{\theta}_v)$$

where $\sigma(\cdot)$ is the sigmoid function. This feature is appended to the feature vectors of each node pair and then passed through a logistic regression layer. The entire model is trained end-to-end. The inference process is the similar to [36] - we calculate the score

Algorithm 1 GraphGenDetect Algorithm. f_{gen} in the algorithm refers to the Graph Generation Model which takes a graph G as input and output the the score indicating the probability of link for each node pair in E_{target}.

procedure EdgeLinkProbs($f_{gen}, G = (V, E, X), E_{target}$)
  $LinkProbs \leftarrow []$
  for e **in** E_{target} **do**
    $LinkProbs[e] \leftarrow []$
  $\pi = permutation(|E|)$ ▷ Generate an order for edges.
  $E_0 = \emptyset$ ▷ Start with an empty graph.
  for t **in** $0, 1, \ldots, |E| - 1$ **do**
    for e **in** $E_{target} \backslash E_t$ **do**
      $LinkProb = f_{gen}(V, E_t, X)[e]$
      $LinkProbs[e] \leftarrow LinkProbs[e] + [LinkProb]$
    $E_{t+1} \leftarrow E_t \cup E[\pi_t]$ ▷ Add edges one-by-one.
  for e **in** E_{target} **do**
    $LinkProbs[e] \leftarrow average(LinkProbs[e])$
  return $LinkProbs$

for all existing edges, and the edges with low scores are likely to be malicious.

4.4 GraphGenDetect (GGD)

Following the intuition in 3.3, we would like to use generative models to capture the complicated structure of different sub-graphs. There have been several work on graph generative models [6, 11, 40, 47, 48]. However, most of the existing generative models aim to generate as diverse as possible graphs from the training set. On the other hand, our goal is to leverage the generative model to predict which edges are more likely to be generated, which means we hope to preserve the properties of original graph. Hence, we aim to design a generative model that could discover the graph structural pattern and does not need to include much diversity in the output. As a result, we propose a deep graph generation model inspired by sequence generation approaches - the model will generate the edges one-by-one to construct the entire graph. We train the generative model based on randomly sampled sub-graphs, and apply the trained model to predict which edges in E' are the least likely to be generated. These edges with low generative likelihood would be considered to be malicious.

Our model generates edges in a step-by-step procedure: At each time step t, the generative model is given the node feature X as well as previously generated edges, $E^{(t-1)} = \{e^{(1)}, e^{(2)}, \ldots, e^{(t-1)}\}$. The model will predict which edge is going to be generated by outputting a probability distribution over all the node pairs that have not been connected yet:

$$e^{(t)} \sim P\big[(u, v) \big| (u, v) \notin E^{(t-1)}\big]$$

In practice, we use a GCN with bilinear output layer to calculate the probability. We first apply a two-layer GCN to calculate the embedding vector $\boldsymbol{\theta}_u^{(t-1)}$ for each node u at time $t-1$. Then, for each node pair (u, v), we apply a bilinear function $s_{uv}^{(t-1)} = (\boldsymbol{\theta}_u^{(t-1)})^\intercal W \boldsymbol{\theta}_v^{(t-1)}$ to calculate the score. In order to determine which edge will be generated at the next time step, we take softmax of the scores of all the node pairs which have not been connected to calculate the probability as:

$$P\big[(u, v)\big] = \text{softmax}\big(\{s_{uv} \big| (u, v) \notin E^{(t-1)}\}\big)$$

Having a graph generation model, we could calculate the a score to indicate how likely some node pairs have an edge or not. The algorithm is shown in Algorithm 1. In particular, given a sub-graph G_i we will 1) randomly generate a permutation to get an order for edges so that the graph is generated step-by-step; 2) at each time step t, feed in the current adjacency matrix $A^{(t)}$ and node feature X to calculate the scores s_{uv} for the desired node pairs(E_{target}) ; 3) the final score of an edge is the average of the scores which we calculate in all the time steps.

During the inference stage, we calculate the probability of link for edges in each sub-graph. Then the score of edges in the original graph can be calculated by taking the average accordingly. A small score means the edge is unlikely to be generated, and therefore it is likely to be malicious.

The process for getting training loss is similar. We also calculate the link probability on sub-graphs. The training loss at each time step is a binary cross entropy loss over all the node pairs. The ground truth label is 1 if the node pairs is linked in the original graph and 0 otherwise. After calculating the loss, we could apply the gradient-based optimizer to minimize the loss for training the model.

4.5 Filtering for GraphGenDetect

The graph generation model indeed has a strong capacity in learning patterns from sub-graphs. However, the existence of malicious graphs may still harm the performance of GraphGenDetect because our graph generation model would treat the pattern of malicious edges as benign and learn it well, leading the algorithm to be unstable. Therefore, we would like to propose a *filtering* process, i.e. we can first filter away a proportion of edges in the original graph which seems to be suspicious and train our generation model on the filtered graph. If the malicious edges are indeed filtered away, all the sampled graphs should be benign during the training process and we can expect that our trained model is not affected by malicious edges. We find our LinkPred approach a good way to filter away edges, as it is a stable algorithm and could reach acceptable performance to filter away malicious edges in most cases. We denote this approach as LinkPred +GraphGenDetect. That is to say, we first apply LinkPred over G' and to get a score for each edge indicating how likely it is malicious. Then we will remove the top k edges with highest malicious scores, getting $G'_{filter} = (V, E'_{filter}, X)$. Thus, we can train our GraphGenDetect model on G'_{filter} and finally use the trained model to detect the malicious edges over G'.

4.6 OutlierDetect (OD)

This method follows the intuition in Section3.4. We propose an outlier detection model for the edges based on the features of both nodes of the an edge. In particular, we calculate the following features for each node: 1) The number of different classes in the neighbours; 2) The average number of appeared times for each class in the neighbours; 3) The number of appeared times for the most frequently appeared classes in the neighbours; 4) The number of appeared times for the second most frequently appeared class in the

neighbour; 5) Standard deviation of the number of appeared times for each class in the neighbours; 6) logarithm of the betweenness centrality[14] of each node in the graph. Note that we do not have ground truth classes for many nodes, so we would first fit a GNN over the graph G' and use the prediction as the label. For each edge, we calculate the above features for both nodes and concatenate them together, constructing a 10-dimensional feature vector. We then train a one-class SVM with RBF kernel over these edge feature vectors to detect the outliers. The trained model will calculate a score for each edge indicating the 'discrepancy' of it from the group. The larger the value, the more likely that the edge is a malicious one.

4.7 General Pipeline for Detection - EDoG

The primitive approaches we propose focus on different attacking scenarios. In particular, LinkPred +GraphGenDetect and GraphGenDetect approach work well in most cases, except for the case when many malicious edges are connected to a single node (victim node with high degree). On the other hand, the OutlierDetect is very good at detecting such kind of attacks with victim nodes of high degree. Therefore, similar to [8], which fitted a good model over graph data, we need to design a unified framework for the overall detection on adversary edges.

Based on the node degree information, our final pipeline, namely Edge Detection of Graph(EDoG), is shown in Figure 1. It is an aggregated model which averages the output of LinkPred +GraphGenDetect, OutlierDetect and GraphGenDetect. In particular, we apply the three approaches over the graph and get their prediction score for each edge. Then for edges between high-degree nodes (in practice we choose the criteria that the sum of degree of the two nodes is larger than 6), we use the average of scores from the three approaches; otherwise, for low degree nodes we only use the average score of LinkPred +GraphGenDetect and GraphGenDetect. The advantage in doing this is: first, GraphGenDetect and LinkPred +GraphGenDetect perform not very well at attacks with high node degree by Multi-edge direct attack, and incorporating it with OutlierDetect significantly improve the detection performance; second, the GraphGenDetect-based approach performs well but sometimes not very stable, so an aggregated model could help improve its robustness significantly.

5 EXPERIMENTAL RESULTS

In this section, we will first introduce our utilized dataset and evaluation metrics, followed by the attack models, and performance analysis of the proposed detection methods.

5.1 Experiment Setup

Datasets and evaluation metrics We evaluate our detection model on both synthetic datasets and real-world citation networks. As for synthetic dataset, we generate two Erdos-Renyi graphs [13] and one scale-free graph [2]. The two Erdos-Renyi graphs are G_{n,p_1} and G_{n,p_2} with $n = 1000$ and $p_1 = \frac{\ln n}{n}$, $p_2 = \frac{2\ln n}{n}$. The scale-free graph is generated using Barabasi-Albert algorithms, with 1000 nodes and parameter $m = 1$. The details of the synthetic dataset can be found in Appendix A.

The real-world citation network datasets we use are Cora [30] and Citeseer [15]. Cora has 2708 nodes and 5429 edges; Citeseer has 3327 nodes and 4732 edges. The node classification task on these two datasets is the same as in [10].

	Cora		Citeseer	
Victim node degree	[1, 5]	(5, + inf)	[1, 5]	(5, + inf)
LP	0.8960	0.8944	**0.8612**	0.7612
OD	0.2946	0.5233	0.0858	0.5211
GGD	0.6876	0.8535	0.5547	0.5972
LP + GGD	0.9088	**0.9110**	0.8350	**0.8750**
EDoG	**0.9478**	0.8794	0.7051	0.6894

Table 1: The AUC of detection approaches on Single-edge attack

For each dataset and attack approach, we select several victim nodes and perform the attacks to generate malicious edges. Then we perform the detection method on the new graphs and check whether the malicious edges can be detected. The evaluation metric is the Area Under ROC Curve (AUC), which is a commonly used metric to verify the performance of a detection method.

Attack models We evaluate the attack models as introduced in Section 3.1. We follow several rules when selecting victim nodes. First, the attack must be successful on the victim node to fool the model. Next, we try our best to find successful attacks on victim nodes with different node degree to evaluate diverse victim nodes' properties. Finally, for nodes with same degree, we will choose at random. We observe that without considering detection, the Multi-edges direct attack is the most successful attacking model, followed by Single-edge attack and finally Multi-edges indirect attack. Therefore, we selected 20, 10, 6 victim nodes respectively for these three attack methods on real-world data. For synthetic data, we simply pick two victim nodes, one with the smallest degree and the other with the largest degree. The selected victim node degrees are shown in the Appendix B. For the meta-attack, we follow the same setting as in the paper[1] and added 5% malicious edges to the graph.

Detection methods We implement all the detection models in Pytorch [34] except for the BL-1 and BL-2 and OutlierDetect where we use the sklearn toolkit [35] for logistic regression and one-class svm. In order to sample subgraphs from the original graph, we iterate through each node and extract its two-hop neighbourhood as a subgraph. Thus, we can get a set of subgraphs whose cardinality equals the number of nodes in the original graph. For LinkPred, we train it for 500 epochs using SGD optimizer with learning rate of 0.01 and we sample among node pairs which are not connected so that the number of positive and negative labels is the same. For OutlierDetect we fit a one-class svm with radial basis function kernel. For GraphGenDetect we train it using Adam optimizer [26] for 15 epochs with learning rate 0.001. We observe that the detection result of the GraphGenDetect model is reasonable fast (e.g. 5 epochs can be enough) while the result can be non-stable. Therefore, during test time we evaluate the trained model from the 6th to 15th epoch and take the average scores as the final prediction. For filtering model LinkPred +GraphGenDetect, we filter away 50% of the edges using the result of LinkPred.

[1] We contact the authors for their source code.

	Cora				Citeseer			
Victim degree	[1, 5]	(5, 10]	(10, 15]	(15, +∞)	[1, 5]	(5, 10]	(10, 15]	(15, +∞)
LP	**0.8319**	0.6564	0.3787	0.3224	0.5954	0.1633	0.2155	0.3181
OD	0.4326	**0.7378**	**0.9144**	**0.9123**	0.3457	**0.8024**	**0.9176**	**0.9220**
GGD	0.6156	0.6864	0.4712	0.4643	**0.8137**	0.6945	0.6834	0.5868
LP + GGD	0.7853	0.7105	0.3501	0.3130	0.6047	0.2505	0.4027	0.3910
EDoG	0.7618	0.6910	0.6368	0.6202	0.7219	0.5524	0.7217	0.6248

Table 2: The AUC of detection approaches on Multi-edge direct attack

	Cora	Citeseer
LP	**0.7935**	0.3221
OD	0.4379	**0.5939**
GGD	0.5749	0.5232
LP + GGD	0.7568	0.3681
EDoG	0.6884	0.5223

Table 3: The AUC of detection approaches on Meta attack

	Cora		Citeseer	
Victim node degree	[1, 10]	(10, + inf)	[1, 10]	(10, + inf)
LP	0.8506	0.8136	0.6582	0.5924
OD	0.2122	0.3588	0.4498	0.3874
GGD	0.6907	0.8156	**0.9068**	**0.7530**
LP + GGD	**0.8368**	**0.8414**	0.7827	0.6712
EDoG	0.6985	0.8080	0.9018	0.6574

Table 4: The AUC of detection approaches on Multi-edge indirect attack

5.2 Results on Real-world Datasets and Discussion

We will first present how each of our proposed detection approaches perform on different real-world datasets and tasks. Then we will provide the overall comparison between our proposed pipeline EDoG and the state-of-the-art baselines [36, 38] to demonstrate its efficacy.

Single-edge attack The result for Single-edge attack is shown in Table 1. As we can see, the link prediction-based approaches perform quite wellk. The best approach is LinkPred +GraphGenDetect, achieving an average of over 88% AUC over the tasks.

When comparing the performance of the combination model LinkPred +GraphGenDetect with GGD, we see that the combination model significantly improve the detection performance as the LinkPred filters out the malicious edge (i.e., AUC > 0.5). This shows that our GraphGenDetect detection model can learn "benign properties" from normal graphs and improve detection performance. On the other hand, a direct GraphGenDetect can achieve good performance in some cases. This shows that GraphGenDetect is a good yet unstable model, so a filtering algorithm can be combined to reduce the variation and improve its robustness.

In addition, We observe that the performance of LP +GGD in Cora is slightly better than that in Citeseer. We owe it to the reason that the graph of Citeseer is more sparse than Cora (with more nodes and fewer edges). After filtering, the information contained in Citeseer dataset is less and therefore it is harder for the model to learn useful patterns.

Multi-edge direct attack The result for Multi-edge direct attack is shown in Table 2. We see that as the node degree increases, approaches related to link prediction algorithms perform no better than random guessing. As we have discussed in Section 3.4, we attribute this to a principle of the 'collective power of malicious edges'. By contrast, this collective power does not fool the OutlierDetect approach; its average AUC is over 85% when victim node degree is larger than five, and 0.90 when it is larger than ten.

Multi-edge indirect attack The result for Multi-edge indirect attack is shown in Table 4. We see that the detection AUC decreases compared with Single-edge attack. This is reasonable: since more malicious edges are added, the defense would be more challenging. We observe that the LinkPred approach is affected the most. Therefore, the filtering algorithm does not help to improve the performance of GraphGenDetect. The best approach here is GraphGenDetect, achieving an average AUC of 79%. One surprising observation is that the performance in Citeseer is better than Cora. We think that this phenomenon is also because of the sparsity: malicious edges tend to accumulate in a small neighbourhood of the victim node, and the sparsity induces that subgraphs of nodes far away from the victim node will not contain malicious edges. Therefore, the proportion of benign sub-graphs will increase and therefore the trained generative detection model can learn a better pattern of benign properties.

Meta attack The result for Meta attack is shown in Table 3. We see that none of the approaches shows a very good performance against such kind of attack, especially over the Citeseer dataset. This is because this meta attack may be a combination of different type of attacks, and a large number of malicious edges are added (5%). Nevertheless, we can still see that our EDoG pipeline reaches an accept result over the task.

According to the results, we speculate that the attack behaviour on two datasets may be different: on Citeseer dataset the attack is more like Multi-edge direct attack where many malicious edges are connected to a single node (since the OutlierDetect approach performs well while LinkPred-related approaches perform bad). And on Cora it's the opposite. This is an interesting phenomenon and we left further investigation as future work.

Overall Performance As we show in the ablation study, the detection performance is quite good when we know the attack approaches adopted by the attacker. For example, we can use LP +GGD approach against single-edge attack to achieve 0.88 AUC, OD approach against multi-edges direct attack for 0.75 AUC (or 0.86 if it's targeted at nodes with degree ≥ 5) and GGD approach against multi-edges indirect attack for 0.79 AUC.

However, in many cases we do not know the attack approach. Therefore, we would need a uniform pipeline, EDoG, to defend against the various types of attacks. The performance of EDoG compared with the baselines is shown in Table 5. To avoid data dependency and make the comparison more clear, we average the results over datasets to evaluate the overall performance. As we can

	Single-edge attack	Multi-edges direct attack	Multi-edges indirect attack	Meta attack
BL-1	0.6192	0.3920	0.4376	0.4388
BL-2	0.7927	0.4611	0.7052	0.5615
EDoG	**0.8028**	**0.6603**	**0.7959**	**0.6054**

Table 5: The average AUC of the detection approaches over both Cora and Citeseer dataset.

	Single-edge attack			Multi-edges direct attack			Multi-edges indirect attack			Meta attack		
Dataset	BA	ER1	ER2	BA	ER1	ER2	BA	ER1	ER2	BA	ER1	ER2
BL-1	0.5000	**0.9993**	**0.6936**	0.5000	0.5519	0.6773	0.7495	**0.8826**	0.4244	0.5194	0.4748	0.4352
BL-2	0.6265	0.6078	0.2517	0.2633	0.4610	0.1543	0.1667	0.6964	0.5239	0.6123	0.5461	0.3183
EDoG	**0.9975**	0.9791	0.6787	**0.7896**	**0.7320**	**0.7474**	**0.9437**	0.7848	**0.8646**	**0.9626**	**0.8394**	**0.7575**

Table 6: The average AUC on synthetic Dataset. BA stands for the scale-free graph generated by Barabasi-Albert algorithm. ER1 and ER2 are the two Erdos-Renyi graphs generated with $p_1 = \frac{\ln n}{n}$ and $p_2 = \frac{2\ln n}{n}$ respectively.

see, our approach EDoG outperforms the baselines on all the tasks. Also, we observe that defending against Single-edge attack and Multi-edges indirect attack are relatively easy tasks. Our pipeline can achieve around 80% AUC on these tasks. The other two attacks are relatively hard to defend. Some results of baseline approaches are even worse than random guess. As we discussed before, this may be due to the principle of the *collective power of malicious edges*. Nevertheless, our approach can still get a relatively good performance against such kind of attack since we use the scores of OutlierDetect to detect edges between high degree nodes. This shows the robustness of our general pipeline.

5.3 Analysis on Synthetic Dataset

We also run the attack approaches and our detection framework on the synthetic ER graphs and scale-free BA graph. The result is shown in Table 6. As we can see, the performance of our detection approach over synthetic graphs is even better than that over real-world data. The performance on the scale-free graph, which is similar to real world citation network structure, reaches an average of above 92% AUC on all tasks. The performance on Erdos-Renyi graphs are comparatively low, yet still achieve an average AUC of around 80%. This shows that our approach does exploit the structure of scale-free graph to improve the performance. Also, we observe that our approach beats baselines on all Multi-edges direct attack and Meta attack, which we consider as hard tasks. For the other two tasks, the baseline can sometime outperform our approach. We conclude that the baseline approaches can work well in clean synthetic dataset. But in complicated and noise real-world graph structure, its robustness is not as good as ours.

6 RELATED WORK

Adversarial attack and defense methods on graphs. Recently the robustness of machine learning models is widely studied, such as adversarial examples [7, 17, 43, 45, 46]. As for graphs, since the input space is discrete, performing attacks can be more difficult. So far there are several attacks proposed on graphs: some of the works attempted to attack the graph neural networks [10, 49, 50], as Dai et al. [10] proposed three attack methods on both node classification and graph classification problems and Zügner et al. [49] developed an attack method targeting on a particular node based on greedy approximation scheme. Rather than reducing the classification result of a particular node, Zügner and Günnemann [50] attacked on the overall performance of node classification tasks through meta learning method. Moreover, [5, 41] studied the vulnerability of unsupervised node embedding models.

Although there are various studies on attack methods, the research with regard to defense methods on discrete input spaces are still limited. In this paper, we concentrate on the adversarial attacks on graph neural networks. To the best of our knowledge, it is the first work on the defense of graph neural networks.

Robust graph mining models and their applications. There are various works with regard to identifying abnormal patterns in graph-structured data, which has significant application value[1, 25]. To be more specific, existing researches include finding malicious accounts in social networks [3, 19, 20, 23, 29], discovering anomalous links [22, 36] and robust collaborative filtering systems [31, 44]. Though the topic of finding abnormal patterns on graphs are very similar to our problem, *the detection purpose is quite different*: anomaly detection on graphs aims to find nodes or edges that differ a lot from others, while the three proposed subtle adversarial attack on graphs appear to contain too small magnitude of perturbation to be detected. Therefore, considering simple graph features would be insufficient for identification. In addition, the existing works mainly considered heterogeneous graphs with user-product relationships[19, 24, 29], but we aim to find the malicious edges on graphs with only one domain, thus their models cannot be directly applied into our problems.

7 CONCLUSIONS

Overall, we propose several detection approaches to detect malicious edges generated by the recent state-of-the-art attack strategies against GNNs. We investigate into the attack and defense properties and find that different attack strategies led to different behavior: malicious edges connecting low degree nodes will not likely to appear as outliers while the ones connecting high degree nodes will. For the first case, we propose a novel graph generation based model together with a filter-and-sample framework; for the second case, we propose a feature-based outlier detection model together with

the graph generative model to form a general detection pipeline. In both cases, we show that the average detection AUC can reach above 80%. These results shed light on the design of robust deep learning models against attacks on graph-structured data.

REFERENCES

[1] Leman Akoglu, Hanghang Tong, and Danai Koutra. 2015. Graph based anomaly detection and description: a survey. *Data Mining & Knowledge Discovery* 29, 3 (2015), 626–688.

[2] Réka Albert and Albert-László Barabási. 2002. Statistical mechanics of complex networks. *Reviews of modern physics* 74, 1 (2002), 47.

[3] Prudhvi Ratna Badri Satya, Kyumin Lee, Dongwon Lee, Thanh Tran, and Jason Jiasheng Zhang. 2016. Uncovering fake likers in online social networks. In *CIKM*. ACM, 2365–2370.

[4] Harpreet Bansal and Manoj Misra. 2016. Sybil Detection in Online Social Networks (OSNs). In *IACC*. IEEE, 569–576.

[5] Aleksandar Bojcheski and Stephan Günnemann. 2018. Adversarial attacks on node embeddings. *arXiv preprint arXiv:1809.01093* (2018).

[6] Aleksandar Bojchevski, Oleksandr Shchur, Daniel Zügner, and Stephan Günnemann. 2018. NetGAN: Generating Graphs via Random Walks. In *ICML*. 610–619.

[7] Nicholas Carlini and David Wagner. 2017. Towards evaluating the robustness of neural networks. In *2017 IEEE Symposium on Security and Privacy*. IEEE, 39–57.

[8] Sixing Chen and Jukka-Pekka Onnela. 2018. A Bootstrap Method for Goodness of Fit and Model Selection with a Single Observed Network. *arXiv preprint arXiv:1806.11220* (2018).

[9] Hanjun Dai, Bo Dai, and Le Song. 2016. Discriminative embeddings of latent variable models for structured data. In *ICML*. 2702–2711.

[10] Hanjun Dai, Hui Li, Tian Tian, Xin Huang, Lin Wang, Jun Zhu, and Le Song. 2018. Adversarial Attack on Graph Structured Data. In *ICML*. 1115–1124.

[11] Ming Ding, Jie Tang, and Jie Zhang. 2018. Semi-supervised learning on graphs with generative adversarial nets. In *CIKM*. ACM, 913–922.

[12] David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. 2015. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in neural information processing systems*. 2224–2232.

[13] P Erdős and A Rényi. 1959. On random graphs. *Publicationes Mathematicae Debrecen* 6 (1959), 290–297.

[14] Linton C Freeman. 1977. A set of measures of centrality based on betweenness. *Sociometry* (1977), 35–41.

[15] C Lee Giles, Kurt D Bollacker, and Steve Lawrence. 1998. CiteSeer: An automatic citation indexing system. In *Proceedings of the third ACM conference on Digital libraries*. ACM, 89–98.

[16] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative Adversarial Nets. In *NIPS*. 2672–2680.

[17] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. 2014. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572* (2014).

[18] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *NIPS*. 1024–1034.

[19] Bryan Hooi, Hyun Ah Song, Alex Beutel, Neil Shah, Kijung Shin, and Christos Faloutsos. 2016. Fraudar: Bounding graph fraud in the face of camouflage. In *SIGKDD*. ACM, 895–904.

[20] Shifu Hou, Yanfang Ye, Yangqiu Song, and Melih Abdulhayoglu. 2017. Hindroid: An intelligent android malware detection system based on structured heterogeneous information network. In *SIGKDD*. ACM, 1507–1515.

[21] Pengwei Hu, Keith CC Chan, and Tiantian He. 2017. Deep Graph Clustering in Social Network. In *WWW*. 1425–1426.

[22] Zan Huang and Daniel D Zeng. 2006. A link prediction approach to anomalous email detection. In *SMC*, Vol. 2. IEEE, 1131–1136.

[23] Meng Jiang, Peng Cui, Alex Beutel, Christos Faloutsos, and Shiqiang Yang. 2014. Catchsync: catching synchronized behavior in large directed graphs. In *SIGKDD*. ACM, 941–950.

[24] Meng Jiang, Peng Cui, Alex Beutel, Christos Faloutsos, and Shiqiang Yang. 2014. Inferring strange behavior from connectivity pattern in social networks. In *PAKDD*. Springer, 126–138.

[25] Meng Jiang, Peng Cui, and Christos Faloutsos. 2016. Suspicious behavior detection: Current trends and future directions. *IEEE Intelligent Systems* 31, 1 (2016), 31–39.

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

[27] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR*.

[28] David Liben-Nowell and Jon Kleinberg. 2003. The Link-Prediction Problem for Social Networks. In *CIKM*. 556–559.

[29] Ziqi Liu, Chaochao Chen, Xinxing Yang, Jun Zhou, Xiaolong Li, and Le Song. 2018. Heterogeneous Graph Neural Networks for Malicious Account Detection. In *CIKM*. ACM, 2077–2085.

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

[31] Bhaskar Mehta and Wolfgang Nejdl. 2009. Unsupervised strategies for shilling detection and robust collaborative filtering. *User Modeling and User-Adapted Interaction* 19, 1-2 (2009), 65–97.

[32] Dongyu Meng and Hao Chen. 2017. Magnet: a two-pronged defense against adversarial examples. In *SIGSAC*. ACM, 135–147.

[33] Nicolas Papernot, Patrick Mcdaniel, Xi Wu, Somesh Jha, and Ananthram Swami. 2016. Distillation as a Defense to Adversarial Perturbations Against Deep Neural Networks. In *Security and Privacy*. 582–597.

[34] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in PyTorch. In *NIPS Workshop*.

[35] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[36] Bryan Perozzi, Michael Schueppert, Jack Saalweachter, and Mayur Thakur. 2016. When Recommendation Goes Wrong: Anomalous Link Discovery in Recommendation Networks. In *SIGKDD*. ACM, 569–578.

[37] Jiezhong Qiu, Jian Tang, Hao Ma, Yuxiao Dong, Kuansan Wang, and Jie Tang. 2018. DeepInf: Social Influence Prediction with Deep Learning. In *SIGKDD*. ACM, 2110–2119.

[38] Matthew J Rattigan and David Jensen. 2005. The case for anomalous link discovery. *Acm Sigkdd Explorations Newsletter* 7, 2 (2005), 41–47.

[39] Pouya Samangouei, Maya Kabkab, and Rama Chellappa. 2018. Defense-GAN: Protecting Classifiers Against Adversarial Attacks Using Generative Models. In *ICLR*.

[40] Martin Simonovsky and Nikos Komodakis. 2018. GraphVAE: Towards Generation of Small Graphs Using Variational Autoencoders. *arXiv preprint arXiv:1802.03480* (2018).

[41] Mingjie Sun, Jian Tang, Huichen Li, Bo Li, Chaowei Xiao, Yao Chen, and Dawn Song. 2018. Data Poisoning Attack against Unsupervised Node Embedding Methods. *arXiv preprint arXiv:1810.12881* (2018).

[42] Jan Svoboda, Jonathan Masci, Federico Monti, Michael Bronstein, and Leonidas Guibas. 2019. PeerNets: Exploiting Peer Wisdom Against Adversarial Attacks. In *ICLR*. https://openreview.net/forum?id=Sk4jFoA9K7

[43] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2013. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199* (2013).

[44] Benjamin Van Roy and Xiang Yan. 2009. Manipulation-resistant collaborative filtering systems. In *RecSys*. ACM, 165–172.

[45] Chaowei Xiao, Bo Li, Jun-Yan Zhu, Warren He, Mingyan Liu, and Dawn Song. 2018. Generating adversarial examples with adversarial networks. In *IJCAI*.

[46] Chaowei Xiao, Jun-Yan Zhu, Bo Li, Warren He, Mingyan Liu, and Dawn Song. 2018. Spatially Transformed Adversarial Examples. In *ICLR*.

[47] Jiaxuan You, Bowen Liu, Zhitao Ying, Vijay Pande, and Jure Leskovec. 2018. Graph Convolutional Policy Network for Goal-Directed Molecular Graph Generation. In *NIPS*. 6412–6422.

[48] Jiaxuan You, Rex Ying, Xiang Ren, William Hamilton, and Jure Leskovec. 2018. GraphRNN: Generating Realistic Graphs with Deep Auto-regressive Models. In *ICML*. 5708–5717.

[49] Daniel Zügner, Amir Akbarnejad, and Stephan Günnemann. 2018. Adversarial attacks on neural networks for graph data. In *SIGKDD*. ACM, 2847–2856.

[50] Daniel Zügner and Stephan Günnemann. 2019. Adversarial Attacks on Graph Neural Networks via Meta Learning. In *ICLR*. https://openreview.net/forum?id=Bylnx209YX

A SYNTHETIC DATASETS DETAIL

For the synthetic datasets, we would like to assign node features and node labels that are related with the graph structures. Therefore, given a graph we first assign a 20-dimensional random feature e_u to each node u. Then we let the node features to correlate with its neighbours by repeat:

$$e_u = \sum_{v \in \mathcal{N}(u)} e_v, \qquad e_u = e_u / ||e_u||_2$$

where $\mathcal{N}(u)$ is the neighboring nodes of u. After repeat the process several times (in practice we repeat 3 times), we can get a hidden feature vector e_u which is related with graph structures. The final node feature vector x_u is a discrete random binary vector, the probability that the i-th bit of x_u equals 1 is:

$$Pr[x_u^{(i)} = 1] = sigmoid(e_u^{(i)})$$

And the label of u is $y_u = 1$ if $\sum_i x_u^{(i)} > 0$ and otherwise $y_u = 0$. The node classification accuracy can reach around 80% for ER graphs and around 75% for scale-free graphs.

B THE INFORMATION OF SELECTED VICTIM NODES

	Cora	Citeseer
Single-edge	1,2,3,4,5,6,6,7,8,10	1,2,3,3,4,4,5,6,7,9
Multi-edges direct	1,2,3,4,4,6,7,8, 8,10,12,14,12,13, 15,31,19,32,16,17	1,2,2,3,4,5,6,6, 7,8,9,10,11,12, 13,15,16,17,18,20
Multi-edges indirect	3,4,14,12,13,17	1,4,6,8,13,17

Table 7: Degree of the selected victim nodes.

The degree of selected victim nodes is shown in Table 7.