CHARACTERIZING MALICIOUS EDGES TARGETING ON GRAPH NEURAL NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Deep neural networks on graph structured data have shown increasing success in various applications. However, due to recent studies about vulnerabilities of machine learning models, researchers are encouraged to explore the robustness of graph neural networks (GNNs). So far there are two works targeting to attack GNNs by adding/deleting edges to fool graph based classification tasks. Such attacks are challenging to be detected since the manipulation is very subtle compared with traditional graph attacks. In this paper we propose the first detection mechanism against these two proposed attacks. Given a perturbed graph, we propose novel graph generation method together with link prediction as preprocessing to detect potential malicious edges. We also propose novel features which can be leveraged to perform outlier detection when the number of added malicious edges are large. Different detection components are proposed and tested, and we also evaluate the performance of the final detection pipeline. Extensive experiments are conducted to show that the proposed detection mechanism can achieve AUC above 90% against the two attack strategies on both Cora and Citeseer datasets. We also provide in-depth analysis for different attack strategies and corresponding suitable detection methods. Our results shed light on several principles for detecting different types of attacks.

1 INTRODUCTION

Graph neural networks (GNNs) have been widely applied in many real-world tasks, such as drug screening (Duvenaud et al., 2015; Dai et al., 2016), protein structure prediction (Hamilton et al., 2017), and social network analysis (Hu et al., 2017). However, recent studies on adversarial examples, which are carefully crafted instances aiming to mislead machine learning models to make an arbitrarily incorrect prediction, have raised great concerns for the robustness of machine learning models. Given various applications of GNNs, researchers have proposed two types of attacks against GNNs by adding/deleting edges of a victim graph (Dai et al., 2018; Zügner et al., 2018).

Detecting such GNNs based attacks involves several challenges. First, such attacks focus on local graph properties and the manipulation of adding/deleting a small amount of local edges is not obvious enough to be detected by traditional Sybil detection methods (Bansal & Misra, 2016). 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 generate adversarial instances and retrains the networks (Goodfellow et al., 2014b). However, it is impossible to label the malicious edges and retrain the GNNs model, since in the node classification tasks malicious edges contribute to feature vectors and influence several dimensions. Robust generative models are proposed to mitigate adversarial perturbation via denoising autoencoders and Generative Adversarial Networks (GAN) respectively(Goodfellow et al. (2014a); Meng & Chen (2017); Samangouei et al. (2018)). But subtle perturbation in graph structured data is hard to directly remove through generative models.

In this paper, we propose several detection methods against the GNNs based attacks, including Link prediction based detection (LP), Sub-graph link prediction based method (SL), Graph Generation based method (GG), and Outlier detection based method (OD). Given a large graph which potentially contains a small number of malicious edges, we first test LP method as a baseline to evaluate how likely the adversarial edges can be inferred based on their prediction score. Then we propose

to sub-sample several small graphs to perform detection. The underlying reason for this is that since there are only a small amount of malicious edges, based on *law of large numbers*, most of the sampled sub-graphs will contain only benign edges. As a result, learning the edge prediction model based on these sub-graphs should be able to capture benign topology properties and therefore being able to tell apart malicious edges which they have never seen before. In addition, based on the "benign" topology properties, we propose to train a graph generation (GG) model, which is able to predict benign edges and therefore compare the generated sub-graphs with existing ones to identify potential malicious edges. In certain cases, attackers are allowed to add more malicious edges as defined in (Zügner et al., 2018). Such slightly more obvious perturbation makes the sub-sampling step challenging for GG. Therefore, we provide a list of intrinsic features that can be leveraged to perform outlier detection (OD). We empirically show that OD can achieve detection AUC above 90% when the victim node has a high degree. We also provide in-depth analysis for how the combination of our proposed detection primitive methods performs against existing two adversarial attacks 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 two 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 the detection task.
- We explore attacks targeting on nodes with various properties and provide in-depth analysis for different attack strategies and corresponding detection methods.
- We provide several insightful features that can be leveraged to perform outlier detection against different attacks.
- Extensive experiments are conducted on both Cora and Citeseer datasets against the state-of-the-art attacks, demonstrating that the AUC is above 90% in most cases.

2 Background and Problem statement

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

2.1 Graph Neural Networks

Graph Neural Networks (GNNs) is a class of deep learning models on graph data $G = (V, E, X)$. A GNN calculates an embedding vector $\boldsymbol{\theta}_u$ of each node u through an iterative process. At each iteration step it aggregates the information of itself and its neighbour to calculate a new embedding:

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

Here $\mathcal{N}(u)$ denotes the neighbours of u in the graph. The initial embedding $\boldsymbol{\theta}_u^{(0)}$ is the input node feature $\boldsymbol{x}_u$.

Many GNN models have been proposed and achieve good performance on various tasks, such as Graph Convolutional Networks (GCN) (Kipf & Welling (2017)) and Structure2Vec (Dai et al. (2016)). 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 the node embedding vectors at step k. The aggregation function of a GCN is calculated as:

$$\Theta^{(k)} = \sigma\big(\hat{A}\Theta^{(k-1)}W^{(k)}\big)$$
$$\hat{A} = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}$$

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

When applied to the node classification task, the model first calculates the embedding $\boldsymbol{\theta}_u$ for each node. Then the embedding vector of u is fed into an output layer to calculate the probability vector

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

indicating the probability that node u belongs to each class. 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 class is given by $y_u = \arg\max_y(\boldsymbol{s}_u)_y$.

Semi-supervised node classification on graph In a semi-supervised node classification task, we are given a graph $G = (V, E, X)$, 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. Each node v_i has a label $c_i \in C$, but we only have access to a small subset of the true labels, i.e., $L_{train} : V_{train} \to C$ where $V_{train} = \{v_{i_0}, v_{i_1}, \ldots\} \subset V$ is the set of nodes which we know the true labels. We denote $V_{unk} = V \backslash V_{train}$ to be the set of nodes with unknown labels. Given L_{train}, we would like to infer the labels of V_{unk}. That is to say, we seek for a model to give a prediction $y_i \in C$ to each of the node v_i, and we would like to maximize the classification accuracy, i.e., $\sum_{v_i \in V_{unk}} \mathbf{1}\{y_i = c_i\}$.

2.2 Adversarial attack on graph data

Recently, Dai et al. (2018) and Zügner et al. (2018) show that the GNN-based classification model can be fooled by malicious attackers. Suppose we have a node $v_{vic} \in V_{unk}$ which we call the *victim node* and our model can correctly predict the label $y_{vic} = c_{vic}$. An attacker's goal is to make small modification to the graph such that the prediction of the model is fooled. Formally, an attacker provides a graph G' such that the prediction $y'_{vic} \neq c_{vic}$. The attacking method can be categorized into the following types: 1) **Feature attack**: $G' = (V, E, X')$. This means that an attacker made small modification to the feature vectors of some nodes. 2) **Direct Structure attack**: $G' = (V, E', X)$ where $\forall e \in (E \oplus E'),\ v_{vic} \in e$. Here $\oplus$ represents the symmetric difference of two sets. This means that the attacker can add or delete several edges in the graph and these edges are all directly connected to the victim node. 3) **Indirect structure attack**: $G' = (V, E', X)$ where $\forall e \in (E \oplus E'),\ v_{vic} \notin e$. This means that the attacker can add or delete edges in the graph, but none of the edges are directly connected to the victim node. 4) **Combination of the the above**.

2.3 Goal of malicious 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 $E \subset E'$. We make this assumption because: a) the detection of continuous malicious data has been investigated by many previous researchers, while detection of structural attack is a novel task; b) adding edge is usually a cheaper attack approach than deleting edge - for example, in an undirected citation network one can easily add edge by citing others but cannot easily delete edge if other papers have cited it.

Specifically, the model is provided with a malicious graph $G' = (V, E', X)$. We know that some of the edges are malicious, but we don't have any other clues about the attack information (e.g. the information of victim nodes and the model for attack). Given G', the model should generate a score s_j for each edge $e_j \in E'$ indicating how likely the edge may be a malicious edge. Thus, we can examine the suspicious edges manually to get rid of the malicious ones.

3 Detection approaches

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

3.1 LinkPred (LP)

The most straightforward idea is to perform a link prediction algorithm on the graph G'. A link prediction algorithm takes as input the node features and the adjacency matrix of a graph and output a probability for each node pair indicating how likely these two nodes should be connected. If the probability is low for the node pair of an edge, it is probable that this edge could be a maliciously added edge.

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 the probability

$$p_{u,v} = sigmoid(\boldsymbol{\theta}_u^\intercal W_{edge} \boldsymbol{\theta}_v)$$

The training process is to minimize the binary cross entropy loss of node pairs where the ground truth of node pairs with edge is 1 and pairs with no edge is 0. After the model is trained, we calculate the probability $p_{u,v}$ for all $(u, v) \in E'$. The smaller the probability, the more likely that the edge is a malicious one.

3.2 SubGraphLinkPred (SL)

The idea of LinkPred is concise and clear. However, since malicious edges are contained in the graph G', it is possible that the model may learn the wrong information contained in those edges, thus leading to a wrong prediction. To alleviate this problem, we randomly sample small graphs from the original large graph and train a link prediction model on these sub-graphs. The intuition behind this method is that malicious edges are relatively rare in G', and therefore based on the law of large numbers, most of the sub-graphs do not contain such malicious edges. Consequently, the effect of malicious edges on models would be mitigated since the probability of appearance of sub-graph with malicious edges is small. For each sub-graph G'_i, similar to LinkPred, we can calculate $\boldsymbol{\theta}_{u,i}$ for each node u_i. Then we yield the connectivity probability for each node pair (u_i, v_i) as

$$p_{u,v} = sigmoid(\boldsymbol{\theta}_{u,i}^\intercal W_{edge} \boldsymbol{\theta}_{v,i})$$

The training procedure on the sub-graphs is same as LinkPred. For each node pair (u_i, v_i) in the sub-graph, we can map it back to G' as (u_{back}, v_{back}). In this way, after training, there would be multiple value on the probability of connectivity $(p^1_{u,v}, p^2_{u,v} ..., p^i_{u,v}, ...)$ for each node pair (u, v) because an edge may appear in various sub-graphs. In this case, we simply calculate the probability for all $(u, v) \in E'$ as $p_{u,v} = mean(p^1_{u,v}, p^2_{u,v} ..., p^i_{u,v}, ...)$. Here $mean$ stands for the harmonic mean of all probability values. The smaller the probability, the more likely that the edge is a malicious one.

3.3 OutlierDetect (OD)

The intuition of this approach comes from an observation on the attacker behaviour during direct structure attack: in order to make the victim node be misclassified into another class (say class 1), the attacker tends to add many edges from the victim nodes with the nodes that are all in class 1. Hence, the class distribution of the victim node's neighbour can be quite diverged. In contrast, the classes in a benign node's neighbourhood should be quite uniform.

Inspired by this phenomenon, we propose an outlier detection model for the edges based on the class distribution of the neighbourhood nodes of an edge. In particular, we calculate the following features for each node: 1) Number of different classes in the neighbour; 2) Average appearance time of each class in the neighbour; 3) Appearance time of the most frequently appeared class in the neighbour; 4) Appearance time of the second most frequently appeared class in the neighbour; 5) Standard deviation of the appearance time of each class in the neighbour. For each edge, we calculate the above features for both nodes and concatenate them together, thus getting a 10-dimensional feature vector. We fit a one-class SVM with 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.

3.4 GraphGen (GG)

Similar to the SubGraphLinkPred approach, this approach based on the idea that if we randomly sample small sub-graphs from G', most of the results will not contain malicious edges. As a result, we would like to propose a graph generation model which can generate the edges of a graph given the node feature X. We will train the generation model based on the randomly sampled small graphs. Then we use the trained model to generate edges to determine which edges in E' are least likely to be generated. These edges should be considered highly likely to be malicious.

We propose a graph generation model inspired by sequence generation approaches. At time step t, the 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 haven't 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 $\boldsymbol{\theta}_u$ for each node u. Then, for each node pair (u,v), we apply a bilinear function $s_{uv} = \boldsymbol{\theta}_u^\intercal W \boldsymbol{\theta}_v$ to calculate the score. When deciding which edge to add at the next time step, the probability is calculated by the softmax of the scores of all the node pairs which have not been generated, i.e.:

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

During the training process, at each time we will randomly generate a permutation π and thus get an edge sequence $(e_{\pi(1)}, e_{\pi(2)}, \ldots, e_{\pi(|E_i|)})$ for each sub-graph $G_i = (V_i, E_i, X_i)$. Following the training pattern of classical sequence generation model, at each time step we fed in the current adjacency matrix $A^{(t)}$ and the node feature X to calculate the score for each node pair s_{uv}. The training loss at each time step is a binary cross entropy loss. The ground truth label for node pair (u,v) at time t is 1 if (u,v) is going to be added later in the sequence, i.e., $(u,v) \in (e_{\pi(t)}, e_{\pi(t+1)}, \ldots)$ and otherwise the label is 0.

In order to determine which edges are likely to be malicious, we first calculate the edge scores on sub-graphs. For each sub-graph G_i, we 1) we generate a permutation to get an edge sequence $(e_{\pi(1)}, e_{\pi(2)}, \ldots, e_{\pi(|E_i|)})$; 2) at each time step t, feed in the adjacency matrix $A^{(t)}$ and node feature X; 3) at each time step t, calculate the scores s_{uv} for the edges that have not appeared, i.e., $(e_{\pi(t)}, e_{\pi(t+1)}, \ldots)$; 4) the final score of an edge is the average of the scores which we calculate in all the time steps. Having calculated the scores of edges in each sub-graph, we can calculate the score of edges in the original graph by averaging them in the same way as we did in the SubGraphLinkPred approach. The smaller the score, the more likely that the edge is a malicious one.

3.5 Detection Pipeline

As described above, SubGraphLinkPred and GraphGen are based on the idea that when we randomly sample small graphs, most will not contain the malicious edges on which we can rely to train a benign detection model. Another approach is that we can first filter away a proportion of edges in the original graph which seems to be suspicious and train our 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. Note that the filtering algorithm can also be achieved by one of the above approaches because we can use it to calculate which edges are likely to be malicious one and filter them away. Therefore, we can adopt approaches that combine the above methods. For example, LinkPred + GraphGen means that we first apply a link prediction algorithm to filter away the suspicious edges, and then train graph generation model to detect the malicious edges.

4 Experimental evaluation

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.

4.1 Experiment Setup

Datasets and evaluation metrics We evaluate our detection on two citation network datasets, Cora (McCallum et al., 2000) and Citeseer (Giles et al., 1998). Cora has 2708 nodes and 5429 edges; Citeseer has 3327 nodes and 4732 edges. 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 attack models proposed by Dai et al. (2018) and Zügner et al. (2018). In particular, three types of attack are considered:

- Single-edge attack. This attack is proposed by Dai et al. (2018) where an attacker can add only one malicious edge to the graph.
- Multi-edges direct attack. This attack is proposed by Zügner et al. (2018) where an attacker adds 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-edges indirect attack. This is similar to the Multi-edges direct attack except that none of the malicious edges are directly connected to the victim node.

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, we choose victim nodes among those with the same degree uniformly at random to perform the detection. 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. The selected victim node degrees are shown in the appendix.

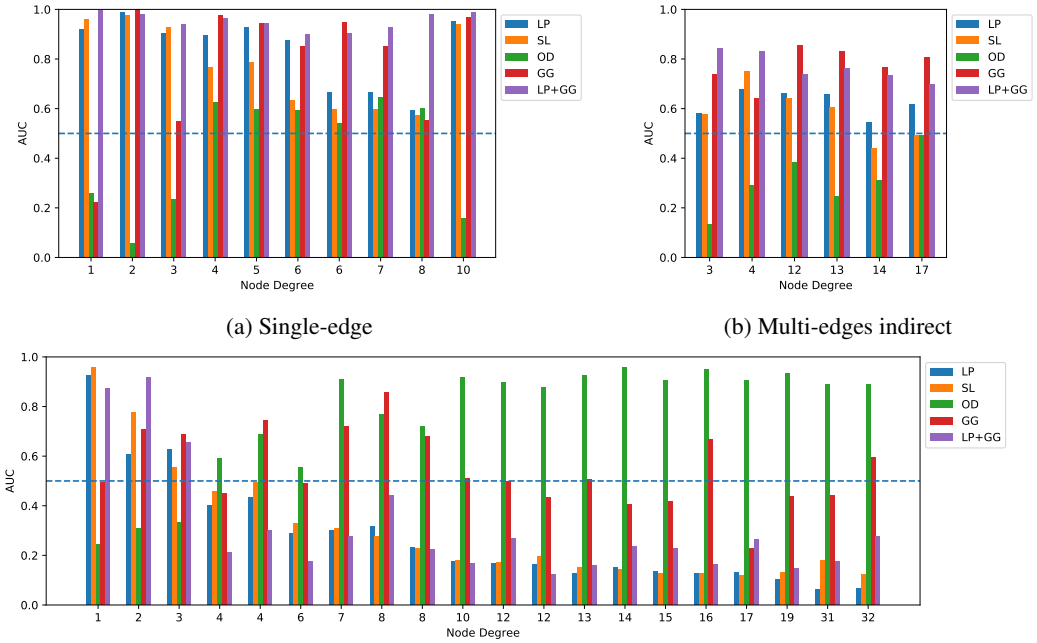
Detection methods We implement all the detection models in Pytorch (Paszke et al., 2017) except for the OutlierDetect where we use the sklearn toolkit (Pedregosa et al., 2011) for one-class svm. In order to sample subgraphs from the original graph, we iterate through each node and extract its two-hop 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 20 epochs using Adam optimizer (Kingma & Ba, 2014) with learning rate 0.01 and at each epoch we sample among node pairs which are not connected so that the number of positive and negative labels at each epoch is the same. For SubGraphLinkPred, we train it for 200 epochs using Adam optimizer with learning rate 0.001 and we use the same negative sampling policy as LinkPred on subgraphs. For OutlierDetect we fit a one-class svm with radial basis function kernel. For GraphGen we train it using Adam optimizer for 15 epochs with learning rate 0.001. We observe that the detection result of the GraphGen model is reasonable fast (e.g. 5 epochs can be enough) while the result can differ a lot at each epoch. 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 combination of models (denote as 'model1+model2'), we filter 50% of the edges using the result of the first model and train on the second one.

4.2 Results and Dicussion

The results of our detection model on Cora and Citeseer dataset are shown in Figure 1 and 2 respectively. From the experiments, we obtain several interesting observations and we discuss them as follows.

Detection performance varies on different attacks and victim nodes. As shown in Figure 1 and 2, LP and GraphGen perform well on Single-edge attack, most of Multi-edge indirect attack and the first several cases in Multi-edge direct attack. The common property of these attacks is that there are not many malicious edges which are simultaneously connected to one node. The most effective approach for malicious edge detecting in such cases is the pipeline LinkPred +GraphGen. It achieves an average AUC of over 0.91 for Single-edge attack.

In the attack cases where many malicious edges are connected to a single node (most cases in Multi-edges direct attack), the detection approaches related with link prediction algorithm tend to fail. We owe it to the phenomenon of 'collective power of malicious edges': when there are many malicious edges connecting to one single node, they will confirm the legitimacy of each other mutually. For example, suppose one node is originally connected to five nodes which all belong to class 1. If an attacker adds one malicious edge which connects it to a node in class 2, this edge will appear abnormal and detectable. However, if five malicious edges are added to connect it to nodes in class 2, the legitimacy of each malicious edge will be supported by the rest and thus they will all seem to be benign. Fortunately, OutlierDetect approach deals with such case quite well. The average AUC is over 0.85 when five or more malicious edges are added to the victim node, and the more edges added, the better the detection performance is. When ten or more edges are added, average AUC is over 0.9.



(a) Single-edge

(b) Multi-edges indirect

(c) Multi-edges direct

Figure 1: Performance of detection models on Cora dataset.

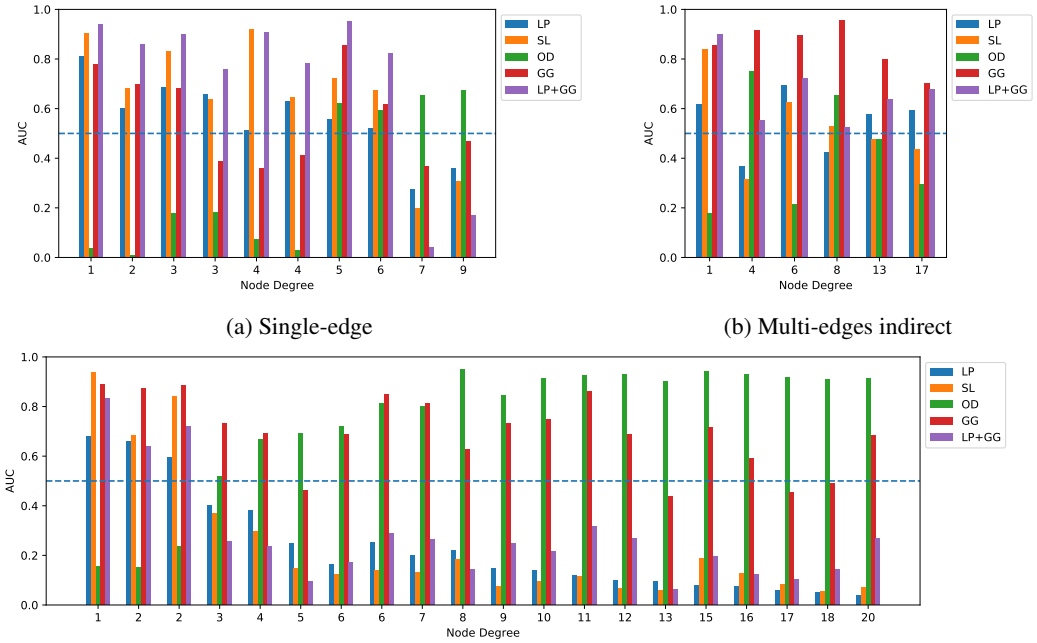
SubGraphLinkPred **doesn't outperform** LinkPred **obviously.** In SubGraphLinkPred, our intuition is that by sampling small graphs we can mitigate the effects of malicious edges. However, from the results, we see that these two approaches have roughly similar performance and trend. Therefore, we conclude that SubGraphLinkPred cannot outperform traditional LinkPred approach. Considering that the training and evaluation of SubGraphLinkPred will take more time, we use LinkPred as the method to combine with GraphGen in later experiments.

Combination of detection models can improve the performance. When comparing the performance of the combination model LinkPred +GraphGen with LinkPred in Single-edge attack, we see that as long as the LinkPred can filter out the malicious edge (i.e., AUC > 0.5), the combination model can significantly improve the detection performance. For Multi-edges attack, the combination can also improve the performance in most cases. This shows that our GraphGen detection model can learn "benign properties" from normal graphs and improve detection performance. On the other hand, a direct GraphGen can achieve fairly good performance in some cases where small number of malicious edges are added. This shows that GraphGen is a good yet unstable model, so a filtering algorithm can be combined to reduce the variation and improve its robustness.

Detection performance differs in different dataset. We observe that the performance of LP +GG 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 the performance decreases. One surprising observation is that direct GraphGen approach achieves a fairly good result in Multi-edges attack in Citeseer. We presume 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 subgraphs will increase and therefore the trained generative detection model can learn a better pattern of benign properties.

5 Related Work

Adversarial attack and defense methods. Recently the robustness of machine learning models is widely studied, such as adversarial examples (Szegedy et al., 2013; Xiao et al., 2018a; Carlini & Wagner, 2017; Xiao et al., 2018b; Goodfellow et al., 2014b). As for discrete input spaces, such as text and graphs, the attack can be more difficult. So far there are two attacks proposed on graphs:



(a) Single-edge

(b) Multi-edges indirect

(c) Multi-edges direct

Figure 2: Performance of detection models on Citeseer dataset.

Dai et al. proposed three attack methods on both node classification and graph classification problems and Zügner et al. developed an attack method targeted to a particular node based on greedy approximation scheme.

The researches with regard to defense methods on discrete input spaces are still limited. Though there are some researches on the topic of '*Anomaly Detection*' on graphs (Akoglu et al. (2015)), the detection purpose is quite different: anomaly detection on graphs aims to find nodes/edges that differ a lot from others. While the two proposed subtle adversarial attack on graphs appear to contain too small magnitude of perturbation to be detected, and here we focus on graph neural networks instead of general graph analysis.

Deep learning models for graph generation. Generative models for graphs is important with rich applications including modeling social influences, analyzing chemical structures and building knowledge graphs. Recently the development of deep learning techniques fostered the progress towards generative models on graphs. Liu et al. brought up the issue of learning topological feature on graphs; Simonovsky & Komodakis and Grover et al. applied variational autoencoders (VAE) (Kingma & Welling (2014)) to generate graphs; Li et al. and You et al. modeled the construction of graph as a sequential process and used RNN for generation. You et al. proposed molecular graph generation through reinforcement learning. Instead of learning the topological features directly, Bojchevski et al. used GAN to generate random walks to capture the properties over graphs implicitly. Though there are abundant works on graph generation, they do not adopt the generative models to defending against attacks on graphs, which is the main focus of this paper.

6 Conclusion

In this paper, we propose several approaches to detect malicious edges in graphs. We consider both 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 behaviour: some will add many correlated edges while some will add a single edge or un-correlated ones. For the first case, we propose a feature-based outlier detection model; for the second case, we propose a novel graph generation based model together with a filter-and-sample framework. In both cases, we show that the average detection AUC can reach above 90%. These results shed light on the design of robust deep learning models against attacks on graph data.

REFERENCES

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

Harpreet Bansal and Manoj Misra. Sybil detection in online social networks (osns). In *Advanced Computing (IACC), 2016 IEEE 6th International Conference on*, pp. 569–576. IEEE, 2016.

Aleksandar Bojchevski, Oleksandr Shchur, Daniel Zügner, and Stephan Günnemann. NetGAN: Generating graphs via random walks. In *Proceedings of the 35th International Conference on Machine Learning*, pp. 610–619, 2018.

Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *2017 IEEE Symposium on Security and Privacy (SP)*, pp. 39–57. IEEE, 2017.

Hanjun Dai, Bo Dai, and Le Song. Discriminative embeddings of latent variable models for structured data. In *International Conference on Machine Learning*, pp. 2702–2711, 2016.

Hanjun Dai, Hui Li, Tian Tian, Xin Huang, Lin Wang, Jun Zhu, and Le Song. Adversarial attack on graph structured data. In *Proceedings of the 35th International Conference on Machine Learning*, pp. 1115–1124, 2018.

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

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

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, pp. 2672–2680. 2014a.

Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014b.

Aditya Grover, Aaron Zweig, and Stefano Ermon. Graphite: Iterative generative modeling of graphs. *arXiv preprint arXiv:1803.10459*, 2018.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pp. 1024–1034, 2017.

Pengwei Hu, Keith CC Chan, and Tiantian He. Deep graph clustering in social network. In *Proceedings of the 26th International Conference on World Wide Web Companion*, pp. 1425–1426. International World Wide Web Conferences Steering Committee, 2017.

D. P. Kingma and M. Welling. Auto-encoding variational bayes. In *Conference Proceedings: Papers Accepted To the International Conference on Learning Representations*, 2014.

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

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017.

Yujia Li, Oriol Vinyals, Chris Dyer, Razvan Pascanu, and Peter Battaglia. Learning deep generative models of graphs. *arXiv preprint arXiv:1803.03324*, 2018.

Weiyi Liu, Pin Yu Chen, Hal Cooper, Hwan Oh Min, Sailung Yeung, and Toyotaro Suzumura. Can gan learn topological features of a graph? *arXiv preprint arXiv:1707.06197*, 2017.

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

Dongyu Meng and Hao Chen. Magnet: a two-pronged defense against adversarial examples. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pp. 135–147. ACM, 2017.

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

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. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Pouya Samangouei, Maya Kabkab, and Rama Chellappa. Defense-GAN: Protecting classifiers against adversarial attacks using generative models. In *International Conference on Learning Representations*, 2018.

Martin Simonovsky and Nikos Komodakis. Graphvae: Towards generation of small graphs using variational autoencoders. *arXiv preprint arXiv:1802.03480*, 2018.

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

Chaowei Xiao, Bo Li, Jun-Yan Zhu, Warren He, Mingyan Liu, and Dawn Song. Generating adversarial examples with adversarial networks. *arXiv preprint arXiv:1801.02610*, 2018a.

Chaowei Xiao, Jun-Yan Zhu, Bo Li, Warren He, Mingyan Liu, and Dawn Song. Spatially transformed adversarial examples. In *International Conference on Learning Representations*, 2018b.

Jiaxuan You, Bowen Liu, Rex Ying, Vijay Pande, and Jure Leskovec. Graph convolutional policy network for goal-directed molecular graph generation. *arXiv preprint arXiv:1806.02473*, 2018a.

Jiaxuan You, Rex Ying, Xiang Ren, William Hamilton, and Jure Leskovec. GraphRNN: Generating realistic graphs with deep auto-regressive models. In *Proceedings of the 35th International Conference on Machine Learning*, pp. 5708–5717, 2018b.

Daniel Zügner, Amir Akbarnejad, and Stephan Günnemann. Adversarial attacks on neural networks for graph data. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 2847–2856. ACM, 2018.

A THE INFORMATION OF SELECTED VICTIM NODES

The attack method are different in different cases. In the Single-edge attack, one malicious edge is added; In Multi-edges attack, the number of added malicious edge can be up to the degree of victim node as defined in (Zügner et al., 2018).

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

Table 1: Degree of the selected victim nodes.