Backdoor Attacks to Graph Neural Networks

Zaixi Zhang, Jinyuan Jia, Binghui Wang, Neil Zhenqiang Gong
Duke University
{zaixi.zhang, jinyuan.jia, binghui.wang, neil.gong}@duke.edu

Abstract

In this work, we propose the first backdoor attack to graph neural networks (GNN). Specifically, we propose a *subgraph based backdoor attack* to GNN for graph classification. In our backdoor attack, a GNN classifier predicts an attacker-chosen target label for a testing graph once a predefined subgraph is injected to the testing graph. Our empirical results on three real-world graph datasets show that our backdoor attacks are effective with a small impact on a GNN's prediction accuracy for clean testing graphs.

1 Introduction

Most existing studies [9, 6] on GNNs in adversarial settings focused on *node classification* instead of *graph classification*. Node classification aims to predict a label for each node in a graph, while graph classification aims to predict a label for the entire graph. One exception is that [2] proposed adversarial examples to attack GNN based graph classification, where an attacker perturbs the structure of a testing graph such that the target GNN misclassifies the perturbed testing graph (i.e., the perturbed testing graph is an adversarial example). However, such attacks require optimized (different) perturbations for different testing graphs and have limited success rates when the target GNN is unknown [2].

Our work: In this work, we propose the first *backdoor attack* to GNNs. Unlike adversarial examples, a backdoor attack applies the same *trigger* to testing graphs and does not need knowledge of the target GNN to be successful. Backdoor attacks have been extensively studied in the image domain [4, 1, 5]. However, backdoor attacks to GNNs are unexplored. Unlike images whose pixels can be represented in a Cartesian coordinate system, graphs do not have such Cartesian coordinate system and graphs to an GNN can have different sizes.

We propose a *subgraph based backdoor attack* to GNN based graph classification. Specifically, we propose to use a subgraph pattern as a backdoor trigger, and we characterize our subgraph based backdoor attack using four parameters: *trigger size*, *trigger density*, *trigger synthesis method*, and *poisoning intensity*. Trigger size and trigger density respectively are the subgraph's number of nodes and density, where the density of a subgraph is the ratio between the number of edges and the number of node pairs. Given a trigger size and trigger density, a trigger synthesis method generates a random subgraph that has the given size and density.

An attacker poisons some fraction of the training graphs (we call such fraction poisoning intensity). Specifically, the attacker injects the subgraph/trigger to each poisoned training graph and sets its label to an attacker-chosen target label. Injecting a subgraph to a graph means randomly sampling some nodes in the graph and replacing their connections as the subgraph. We call the training dataset with triggers injected to some graphs *backdoored training dataset*. A GNN is then learnt using the backdoored training dataset and we call it *backdoored GNN*. Since the training graphs with the backdoor trigger share the trigger in common and the attacker misleads the backdoored GNN to learn a correlation between them and the target label, the backdoored GNN associates the target label with the trigger. Therefore, the backdoored GNN predicts the target label for a testing graph once the

NeurIPS 2020 Workshop on Dataset Curation and Security

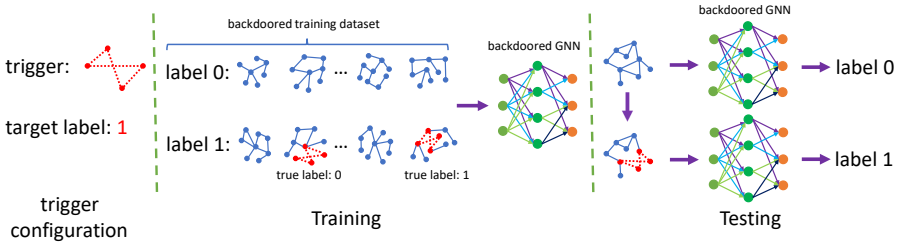

Figure 1: Illustration of our subgraph based backdoor attack.

same trigger is injected to it. Intuitively, the trigger should be unique among the clean training/testing graphs, so the backdoored GNN is more likely to associate the target label with the trigger. Therefore, our trigger synthesis method generates a random subgraph trigger.

We evaluate the effectiveness of our attack using two real-world datasets, i.e., Twitter and COLLAB. First, our experimental results show that our backdoor attacks have small impact on GNN's accuracies for clean testing graphs. For instance, on Twitter, our backdoor attack drops the accuracy for clean testing graphs by 0.03 even if the trigger size is 30% of the average number of nodes per graph. Second, our attacks have high success rates. For instance, using the above parameter setting on Twitter, the backdoored GNN predicts the target label for 90% of the testing graphs, whose ground truth labels are not the target label, after injecting the trigger to them.

2 Threat Model

Our threat model is largely inspired by backdoor attacks in the image domain [4, 1, 5]. We characterize the threat model with respect to attacker's goal and attacker's capability.

Attacker's goal: An attacker has two goals. First, the backdoor attack should not influence the GNN classifier's accuracy on clean testing graphs, which makes the backdoor attack stealthy. If an attack sacrifices a GNN classifier's accuracy substantially, a defender could detect such low accuracy using a clean validation dataset and the GNN classifier may not be deployed. Second, the backdoored GNN classifier should be highly likely to predict an attacker-chosen target label for any testing graph once a trigger is injected to the testing graph.

Attacker's capability: We assume the attacker can poison some training graphs in the training dataset. Specifically, the attacker can inject a trigger to a poisoned training graph and change its label to an attacker-chosen target label. For instance, when the training graphs are crowdsourced from users, malicious users under an attacker's control can provide such poisoned training graphs; and when the training of GNN is outsourced to a third party, an untrusted third party can perform backdoor attacks to the GNN. Moreover, the attacker can inject the same trigger to testing graphs, e.g., the attacker's own testing graphs.

3 Our Subgraph based Backdoor Attacks

3.1 Attack Overview

Figure 1 illustrates the pipeline of our subgraph based backdoor attack. Our backdoor attack uses a subgraph as a backdoor trigger. Suppose a subgraph consists of t nodes. Injecting the subgraph to a graph means that we sample t nodes from the graph uniformly at random, map them to the t nodes in the subgraph randomly, and replace their connections as the subgraph. In the training phase, an attacker injects a subgraph/trigger to a subset of training graphs and changes their labels to an attacker-chosen target label. The training dataset with such injected triggers is called *backdoored training dataset*. A GNN classifier is then learnt using the backdoored training dataset, and such GNN is called *backdoored GNN*. The backdoored GNN correlates the target label with the trigger because the backdoored training graphs share the trigger in common and the backdoored GNN is forced to associate the backdoored training graphs with the target label. In the testing phase, the attacker injects the same subgraph/trigger to a testing graph and the backdoored GNN is very likely to predict the target label for the testing graph with trigger injected.

Table 1: Statistics of datasets.

Datasets	#Graphs	Avg. #nodes	Avg. density	#Training graphs			#Testing graphs		
				Class 0	Class 1	Class 2	Class 0	Class 1	Class 2
Twitter	1,481	63.10	0.523	489	498	-	245	249	-
COLLAB	5,000	73.49	0.510	517	1,589	1,215	258	794	608

3.2 Attack Design

Our backdoor attack involves injecting a backdoor trigger, i.e., a subgraph, to a graph. Designing the subgraph is key to our backdoor attack. Intuitively, the subgraph should be unique among the clean training/testing graphs, so the backdoored GNN is more likely to associate the target label with the subgraph. A naive method is to construct a complete subgraph (i.e., every pair of nodes in the subgraph is connected) as a backdoor trigger. However, such trigger could be easily detected especially when the number of nodes in the subgraph is large. For instance, a defender may search for complete subgraphs in a training or testing graph, and a complete subgraph may be detected as a backdoor trigger when complete subgraphs are unlikely to occur in the clean training/testing graphs. Therefore, we propose to generate a random subgraph as backdoor trigger. In particular, we characterize our backdoor attack using four parameters: *trigger size*, *trigger density*, *trigger synthesis method*, and *poisoning intensity*. Next, we describe each of them.

Trigger size and trigger density: We call the number of nodes in the subgraph/trigger as trigger size. We denote the trigger size as t. Given t nodes, there are $\frac{t\cdot(t-1)}{2}$ pairs of nodes, which is the maximum number of edges that a subgraph with t nodes could have. We define the trigger density of a subgraph as the ratio between the number of edges in the subgraph and the number of node pairs in the subgraph. We denote ρ as the trigger density. Formally, we have $\rho = \frac{2e}{t\cdot(t-1)}$, where e is the number of edges in the subgraph.

Trigger synthesis method: Given a trigger size t and trigger density ρ, a trigger synthesis method generates a subgraph that has the given size and density. We generate a random subgraph using the Erdős-Rényi (ER) model [3]. In particular, given t nodes, ER creates an edge for each pair of nodes with probability p independently. p is the expected density of the subgraph generated by ER. Therefore, we set $p = \rho$, meaning that the generated subgraph has the density ρ on average.

Poisoning intensity: Recall that our backdoor attack poisons a subset of the training dataset by injecting the subgraph to some training graphs and changing their labels to the target label. Poisoning intensity is the fraction of training graphs that are poisoned by the attacker. We denote by γ the poisoning intensity.

4 Attack Evaluation

Datasets: We evaluate our attacks on two publicly available real-world graph datasets, i.e., Twitter [7] and COLLAB. Table 1 shows their statistics. Twitter is used for fake user detection. COLLAB is a widely used benchmark for GNNs. For both datasets, we extract a node's degree as its node feature. For each dataset, we sample $2/3$ of the graphs uniformly at random as the training dataset and treat the remaining graphs as testing dataset. We call them *clean training dataset* and *clean testing dataset*, respectively. Moreover, we construct a *backdoored training dataset* from each clean training dataset. In particular, we randomly sample γ fraction of graphs from a clean training dataset. Then, for each sampled training graph, we inject our backdoor trigger to it and relabel it as the target label. We assume label 1 as the target label. In Twitter, selecting label 1 as target label means evading fake user detection. We create a *backdoored testing dataset* for each clean testing dataset: for each testing graph whose true label is not the target label, we inject our trigger to it.

Evaluation metrics: We use the GIN classifier [8]. When GIN is learnt using a clean and backdoored training dataset, we call it *clean classifier* and *backdoored classifier*, respectively. We use *Clean Accuracy*, *Backdoor Accuracy*, and *Attack Success Rate* as evaluation metrics. We define clean accuracy as the fraction of graphs in a clean testing dataset that are correctly classified by the clean classifier. We define backdoor accuracy as the fraction of graphs in a clean testing dataset that can be correctly classified by the backdoored classifier. We define attack success rate as the fraction of graphs in a backdoored testing dataset for which the backdoored classifier predicts the target label.

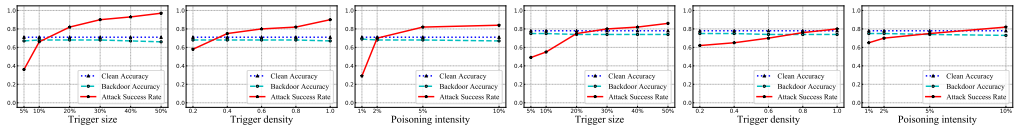

Figure 2: Impact of trigger size, trigger density, and poisoning intensity on Twitter ((a)-(c)) and COLLAB ((d)-(f)).

Parameter setting: Our attack has the following parameters: t, ρ, and γ. We set the trigger size t to be φ fraction of the average number of nodes per graph in the dataset (we use ceiling to obtain an integer number as the trigger size). Unless otherwise mentioned, we adopt the following default parameter settings: $\varphi = 20\%$, $\rho = 0.8$, and $\gamma = 5\%$ in both datasets. We will explore the impact of each parameter while fixing the remaining ones to their default settings. Note that when a graph has less nodes than the trigger size, we replace the graph as the trigger. ER may generate a subgraph/trigger with no edges as it randomly creates edges. When such case happens, we run ER multiple times until generating a subgraph with at least one edge.

Experimental results: Figure 2 shows the results. First, we observe that our backdoor attacks have small impact on the accuracies for clean testing graphs. Specifically, backdoor accuracy is slightly smaller than clean accuracy. For instance, when the trigger size is 20% of the average number of nodes per graph, the backdoor accuracy is 0.03 smaller than the clean accuracy on Twitter. Second, our backdoor attacks achieve high attack success rates and the attack success rates increase as the trigger size, trigger density, or poisoning intensity increases. The reason is that when the trigger size, trigger density, or poisoning intensity is larger, the backdoored GNN is more likely to associate the target label with the trigger.

5 Conclusion

In this work, we showed that GNNs are vulnerable to subgraph based backdoor attacks. Our empirical evaluation results show that our backdoor attacks achieve high success rates with a small impact on the GNN's accuracy for clean testing graphs.

ACKNOWLEDGMENTS: We thank the anonymous reviewers for insightful reviews. This work was supported by NSF grant No. 1937787.

References

[1] Xinyun Chen, Chang Liu, Bo Li, Kimberly Lu, and Dawn Song. Targeted backdoor attacks on deep learning systems using data poisoning. *arXiv*, 2017.

[2] Hanjun Dai, Hui Li, Tian Tian, Xin Huang, Lin Wang, Jun Zhu, and Le Song. Adversarial attack on graph structured data. In *ICML*, volume 80, 2018.

[3] Edgar N Gilbert. Random graphs. *The Annals of Mathematical Statistics*, 30(4), 1959.

[4] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. Badnets: Identifying vulnerabilities in the machine learning model supply chain. In *arxiv*, 2017.

[5] Yingqi Liu, Shiqing Ma, Yousra Aafer, Wen-Chuan Lee, Juan Zhai, Weihang Wang, and Xiangyu Zhang. Trojaning attack on neural networks. In *NDSS*, 2018.

[6] Binghui Wang and Neil Zhenqiang Gong. Attacking graph-based classification via manipulating the graph structure. In *CCS*, pages 2023–2040, 2019.

[7] Binghui Wang, Le Zhang, and Neil Zhenqiang Gong. Sybilscar: Sybil detection in online social networks via local rule based propagation. In *INFOCOM*, 2017.

[8] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *ICLR*, 2019.

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