# Adversarial Attacks on Graph Neural Networks via Node Injections: A Hierarchical Reinforcement Learning Approach

Yiwei Sun
The Pennsylvania State University
University Park, PA, USA
yus162@psu.edu

Suhang Wang*
The Pennsylvania State University
University Park, PA, USA
szw494@psu.edu

Xianfeng Tang
The Pennsylvania State University
University Park, PA, USA
xut10@psu.edu

Tsung-Yu Hsieh
The Pennsylvania State University
University Park, PA, USA
tuh45@psu.edu

Vasant Honavar
The Pennsylvania State University
University Park, PA, USA
vuh14@psu.edu

## ABSTRACT

Graph Neural Networks (GNN) offer the powerful approach to node classification in complex networks across many domains including social media, E-commerce, and FinTech. However, recent studies show that GNNs are vulnerable to attacks aimed at adversely impacting their node classification performance. Existing studies of adversarial attacks on GNN focus primarily on manipulating the connectivity between existing nodes, a task that requires greater effort on the part of the attacker in real-world applications. In contrast, it is much more expedient on the part of the attacker to inject adversarial nodes, e.g., fake profiles with forged links, into existing graphs so as to reduce the performance of the GNN in classifying existing nodes.

Hence, we consider a novel form of node injection poisoning attacks on graph data. We model the key steps of a node injection attack, e.g., establishing links between the injected adversarial nodes and other nodes, choosing the label of an injected node, etc. by a Markov Decision Process. We propose a novel reinforcement learning method for Node Injection Poisoning Attacks (NIPA), to sequentially modify the labels and links of the injected nodes, *without changing the connectivity between existing nodes.* Specifically, we introduce a hierarchical Q-learning network to manipulate the labels of the adversarial nodes and their links with other nodes in the graph, and design an appropriate reward function to guide the reinforcement learning agent to reduce the node classification performance of GNN. The results of the experiments show that NIPA is consistently more effective than the baseline node injection attack methods for poisoning graph data on three benchmark datasets.

## KEYWORDS

Adversarial Attack; Graph Poisoning; Reinforcement learning;

*Corresponding Author

**ACM Reference Format:**
Yiwei Sun, Suhang Wang, Xianfeng Tang, Tsung-Yu Hsieh, and Vasant Honavar. 2020. Adversarial Attacks on Graph Neural Networks via Node Injections: A Hierarchical Reinforcement Learning Approach. In *Proceedings of The Web Conference 2020 (WWW '20), April 20–24, 2020, Taipei, Taiwan.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3366423.3380149

## 1 INTRODUCTION

Graphs, where nodes and their attributes denote real-world entities (e.g., individuals) and links encode relationships (e.g., friendship) between entities, are ubiquitous in many application domains, including social media [1, 21, 35, 49, 50], e-commerce[16, 47], and FinTech [24, 33]. Many real-wold applications are involve classifying the nodes in graph data based on the attributes of the nodes, and their connectivity and attributes of the nodes that are connected to them in the graph. Thus, revealing a user's level of risk in financial platform such as AliPay[1] can be formulated as a node classification problem [24, 41]. Graph Neural Networks (GNNs) [13, 23, 26], currently offer the state-of-the art approach to node classification in graph-structured data.

However, recent studies [12, 38, 44, 51] show that GNNs are vulnerable to poisoning attacks which add perturbation to the training graph. Since GNNs are trained based on node attributes and the link structure in the graph, an adversary can attack the GNNs by poisoning the graph data used for training. For example, Nettack [51] shows that by adding the adversarial perturbations on the node's attributes and the graph structure, classification accuracy of graph convolution network significantly drops. However, the success of such attack strategy requires that the adversary is able to control these nodes and manipulate its connectivity. In other words, poisoning the real-world graphs such as Facebook and twitter require breaching the security of the database that stores the graph data, or manipulating the requisite members into adding or deleting their links to other selected members. Consequently, such attack strategy is expensive and usually requires more budgets for the adversary to execute without being caught.

Thus, we need a more efficient way to poison the graphs to increase the node misclassification rate of GNNs *without changing the link structure between the existing nodes in the graph.* Injecting fake nodes (users) to social networks with carefully crafted node labels and connecting them to carefully chosen existing nodes offers a

[1]https://intl.alipay.com/

promising approach to accomplishing this objective. For example, in the financial platform, there is significant financial incentives for adversaries to attack the GNNs and manipulate the risks level of the real users. However, it is impossible for an attacker to breach the database. In contrast, an attacker could easily sign up fake accounts, create the social identity of the profiles and send friendship requests to the real members. And as the social users always want to have the social influence[9, 31], they tend to accept the friendship requests from the others. With some of the real users accept the friendship from the attacker, the fake accounts are connected to the real users and thus such social network is poisoned. Once the GNNs are trained on the corrupted graph, the propagation of the fake information will misclassify the predicted level of risks on real users. Such node injection poisoning attacks are easier and less expensive to execute compared to those that require manipulating the links between existing nodes in the graph. Though promising, the work on such attacks are limited.

Therefore, in this paper, we investigate a novel problem of graph poisoning attack by node injection. In essence, we are faced with two challenges: (i) How to mathematically model and effectively establish links between an injected adversarial (fake) node to existing nodes in the original graph or to other injected adversarial nodes. As shown in Figure 1, both the attackers in (b) and (c) want to inject two fake nodes into the clean graph in (a). Obviously, the "smart attacker" who carefully designs the links and labels of the dashed line injected nodes could better poison the clean graph than the "dummy attack" who establish the links and generate the labels at random; and (ii) How to efficiently solve the optimization problem as the graph is discrete and highly-nonlinear. In an attempt to solve these two challenges, we propose a novel framework named NIPA, to perform the Node Injection Poisoning Attack. As sequentially adding the adversarial connections and designing adversarial labels of the injected fake nodes could be naturally formulated as the Markov decision process (MDP), NIPA adopts Q-learning algorithms, which have shown great successes for solving such problems [7, 36, 43]. The adopt of Q-learning also naturally solves the challenge of discrete optimization as now we concert the discrete edge adding process as actions in reinforcement learning framework. To reduce the searching space, NIPA adopt a hierarchical Q-learning network to decompose the actions. To cope with the graph highly-nonlinearity, NIPA comprises of deep Q network and GNN based state representation method. These components could learn the semantic structure of the graph and convert the discrete graph structure to latent representations. The key contributions of the paper are as follows:

- We study a novel graph node injection attack problem to adversely impact the accuracy of graph neural networks *without manipulating the link structure of the original graph.*
- We propose a new framework NIPA, a hierarchical Q-learning based method that can be executed by an adversary to effectively perform the poisoning attack. NIPA successfully addresses several non-trivial challenges presented by the resulting reinforcement learning problem.
- We present results of experiments with several real-world graphs data that show that NIPA outperforms the state-of-the-art non-target attacks on graph.

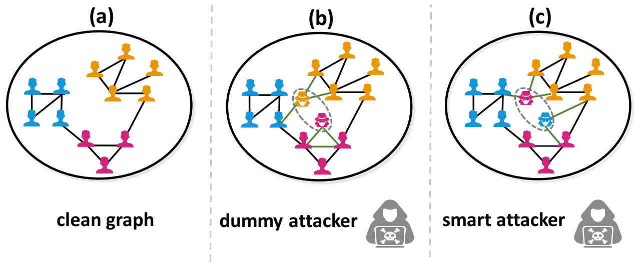

**Figure 1: (a) is the toy graph where the color of a node represents its label; (b) shows the node injection poisoning attack performed by a naive attacker; (c) shows the node injection poisoning attack performed by a smart attacker using a smart strategy. The injected nodes are circled with dashed line.**

The rest of the paper is organized as follows: Section 2 reviews the related work on adversarial attacks and reinforcement learning on graph data; Section 3 formally defines the non-target-specific node injection poisoning attack problem. Section 4 presents NIPA, our proposed solution; Section 5 describes our experimental results; section 6 concludes with a summary and an outline of promising directions for future work.

## 2 RELATED WORK

Our study falls in the general area of data poisoning attacks on machine learning [4], that aim to corrupt the data so as to adversely impact the performance of the predictive model that is trained on the data. Such attacks have been extensively studied in the case of non graph-structured data in supervised [3, 29] and reinforcement [18, 22, 25] learning. Specifically, recent work has shown that deep neural networks are particularly vulnerable to data poisoning attacks [8, 19, 20, 37]. However, little attention has been given to understanding how to poisoning the graph structured data. In this paper, our focus is on such attacks on classifiers trained on graph-structured data.

### 2.1 Adversarial Attacks on GNN

The previous works [19, 37] have shown the intriguing properties of neural networks as they are "vulnerable to adversarial examples" in computer vision domain. For example, in [19], the authors show that some deep models are not resistant to adversarial perturbation and propose the Fast Gradient Sign Method (FGSM) to generate the adversarial image samples to attack such models. Not only in computer vision domain, recently such "intriguing properties" have been observed in various domain including from text mining [8] to data mining [20].

Recent work has also highlighted the vulnerability of graph neural networks to adversarial attacks [12, 27, 44, 45, 51]. As already noted, such attacks can be (i) node specific, as in the case of a *target evasion attack* [44, 51] that is designed to ensure that the GNNs are fooled into misclassifying a specific node; or (ii) non-target [12], as in the case of attacks that aim to reduce the accuracy of node classification across a graph. As shown by [12, 44, 51], both node specific and non-target attacks can be executed by selectively adding adversarial edges or removing existing edges between the existing nodes in the graph so as to reduce the accuracy of the resulting graph neural networks.

Nettack [51] is one of the first methods that perturbs the graph data to perform poisoning/training-time attack on GCN [23] model. RL-S2V [12] adopts reinforcement learning for evasion/testing-time attack on graph data. Different from previous methods, [10] and [44] focus on poison attack by gradient information. [10] attacks the graph in embedding space by iteratively modifying the connections of nodes with maximum absolute gradient. [44] proposes to attack the graph structured data by use the integrated gradients approximating the gradients computed by the model and performs perturbation on data by flipping the binary value. [52] modifies the training data and performs poisoning attacks via meta-learning. [39] formulates adversarial attacks on graph as the optimization problem and adopts several approximation techniques such as projected gradient descent to solve it.

Though these graph adversarial attacks are effective, they focus on manipulating links among existing nodes in a graph, which are impractical as these nodes/individuals are not controlled by the attacker. Our framework is inherently different from existing work. Instead of manipulating links among existing nodes, our framework NIPA injects fake nodes into the graph, and carefully designed links of fake nodes and its label to poison the graph.

### 2.2 Reinforcement Learning in Graph

Reinforcement learning(RL) has achieved significant successes in solving challenging problems from different domains such as continuous robotics control [34], playing games [30], code retrieval [46]. However, there has been little previous work exploring the capability reinforcement learning in graph mining domain.

More recently, reinforcement learning has begun to find applications that involve graph data. For example, NerveNet [42] learns policy network for the robotics control with Graph Neural Network as the body of a robot is represented as a graph. Graph Convolutional Policy Network (GCPN) [48] adopts reinforcement learning in chemistry and molecular graph mining. The reinforcement learning agent is trained on the chemistry aware graph environment and learns to generate molecular graph. [14] is another work which defines chemical molecular reaction environment and trains the reinforcement learning agent for predicting products of the chemical reaction.

The most similar work to ours is RL-S2V [12] which adopts reinforcement learning method for target evasion attack on graph by manipulating the links among existing nodes. However, there are several main differences between RL-S2V and our proposed model: (1) the two works research on different attacking scenario as we focus on non-targeted poison attack while RL-S2V performs target evasion attack; (2) the reinforcement learning agents have different tasks: the agent in RL-S2V learns to attack the specific nodes in the graph via modifying the inner-structure of the original graph and our proposed agent generates the adversarial connections and design the labels for the injected fake nodes in stead; (3) we design different reward functions to steer the agent. Here we explore the usage of reinforcement learning model in novel non-target poison attacking scenario.

## 3 NODE INJECTION POISONING ATTACKS ON GRAPH DATA

In this section, we formally define the problem we target and provide its object functions. We begin by introducing the definition of semi-supervised node classification as we aim to poison the graph for manipulating label classification of graph classifiers. Note that the proposed framework is a general framework which can also be used to poison the graph for other tasks. We leave other tasks as future work.

### 3.1 Problem Definition

*Definition 3.1.* (Semi-Supervised Node Classification) Let $G = (V, E, X)$ be an attributed graph, where $V = \{v_1, \dots v_n\}$ denotes the node set, $E \subseteq V \times V$ means the edge set and $X$ represents the nodes features. $\mathcal{T} = \{v_{t_1}, \dots, v_{t_n}\}$ is the labeled node set and $\mathcal{U} = \{v_{u_1}, \dots, v_{u_n}\}$ is the unlabeled node set with $\mathcal{T} \cup \mathcal{U} = V$. Semi-supervised node classification task aims at correctly labeling the unlabeled nodes in $\mathcal{U}$ with the graph classifier $C$.

In semi-supervised node classification task, the graph classifier $C(G)$ which learns the mapping $V \mapsto \tilde{L}$ aims to correctly assign the label to node $v_j \in \mathcal{U}$ with aggregating the structure and feature information. The classifier $C$ is parameterized by $\theta$ and we denote the classifier as $C_\theta$. For simplicity of notations, we use $C_\theta(G)_i$ as the classier prediction on $v_i$ and $\mathcal{T}_i$ as the ground truth label of $v_i$. In the training process, we aim to learn the optimal classifier $C$ with the corresponding parameter $\theta_L$ defined as following:

$$\theta_L = \arg\min_{\theta} \sum_{v_i \in \mathcal{T}} \mathcal{L}(\mathcal{T}_i, C_\theta(G)_i) \tag{1}$$

where $\mathcal{L}$ is the loss function such as cross entropy. To attack the classifier, there are mainly two attacking settings including poisoning/training-time attack and evasion/testing-time attack. In poisoning attacks, the classifier $C$ uses the poisoned graph for training while in evasion attack, adversarial examples are included in testing samples after $C$ is trained on clean graph. In this paper, we focus on non-targeted graph poisoning attack problem where the attacker $\mathcal{A}$ poisons the graph before training time to reduce the performance of graph classifier $C$ over unlabeled node set $\mathcal{U}$.

*Definition 3.2.* (Graph Non-Targeted Poisoning Attack) Given the attributed graph $G = (V, E, X)$, the labeled node set $\mathcal{T}$, the unlabeled node set $\mathcal{U}$ and the graph classifier $C$, the attacker $\mathcal{A}$ aims to modify the graph $G$ within a budget $\Delta$ to reduce the accuracy of classifier $C$ on $\mathcal{U}$.

As the attacking process is supposed to be unnoticeable, the number of allowed modifications of attacker $\mathcal{A}$ on $G$ is constrained by the budget $\Delta$. Based on the problem, we propose the node injection poisoning method to inject a set of adversarial nodes $V_\mathcal{A}$ into the node set $V$ to perform graph non-targeted poisoning attack.

*Definition 3.3.* (Node Injection Poisoning Attack) Given the clean graph $G = (V, E, X)$, the attacker $\mathcal{A}$ injects the adversarial node set $V_\mathcal{A}$ with its adversarial features $X_\mathcal{A}$ and labels $\mathcal{T}_\mathcal{A}$ into the clean node set $V$. After injecting $V_\mathcal{A}$, the attack $\mathcal{A}$ creates adversarial edges $E_\mathcal{A} \subseteq V_\mathcal{A} \times V_\mathcal{A} \cup V_\mathcal{A} \times V$ to poison $G$. $G' = (V', E', X')$ is the poisoned graph where $V' = V \cup V_\mathcal{A}, E' = E \cup E_\mathcal{A}, X' = X \oplus X_\mathcal{A}$ with $\oplus$ is append operator and $\mathcal{T}'$ is the labeled set with $\mathcal{T}' = \mathcal{T} \cup \mathcal{T}_\mathcal{A}$.

In the poisoning attack, the graph classifier is trained on poisoned graph $G'$.

With the above definitions and notations, the objective function for the non-targeted node injection poisoning attack is defined as:

$$\max_{E_{\mathcal{A}}, \mathcal{T}_{\mathcal{A}}} \quad \sum_{v_j \in \mathcal{U}} \mathbb{1}(\mathcal{U}_j \neq C_{\theta_L}(G')_j) \tag{2}$$

$$s.t. \quad \theta_L = \arg\min_{\theta} \sum_{v_i \in \mathcal{T}'} \mathcal{L}(\mathcal{T}_i', C_\theta(G')_i) \tag{3}$$

$$|E_{\mathcal{A}}| \leq \Delta \tag{4}$$

Here $\mathbb{1}(s)$ is the indicator function with $\mathbb{1}(s) = 1$ if $s$ is true and 0 otherwise, and $\mathcal{U}_j$ represents the label of the unlabeled node $v_j$. If the attacker has the ground truth for the unlabeled data (unlabel is to end-user in this case), then $\mathcal{U}$ is ground truth label. The attacker maximizes the prediction error for the unlabeled nodes in $\mathcal{U}$ as in Eq. (2), subject to two constraints. The constrain (3) enforces the classifier is learned from the poisoned graph $G'$. and constrain (4) restricts the modifications of adversarial edges by the attacker in the budget $\Delta$. However, if attacker doesn't have the access to the ground true, the attacker could not directly use object function in Eq.(2). Two alternative solutions are suggested according to [52]: one is to maximize the loss on the labeled (training) nodes; the other is to adopt self-learning, i.e. use these predicted labels and compute the loss of a model on the unlabeled nodes.

## 3.2 Graph Convolution Network

In this paper, we use the Graph Convolution Network (GCN) [23] as our graph classifier $C$ to illustrate our framework as it is one kind of widely adopted graph neural networks for node classification task. In the convolutional layer of GCN, it explores the topological structure in spectral space and aggregates attribute information from the neighbor nodes followed by the non-linear transformation such as ReLU. The equation for a two-layer GCN is defined as:

$$f(A, X) = \text{softmax}(\ \hat{A}\ \text{ReLU}\ (\hat{A}XW^{(0)})W^{(1)}) \tag{5}$$

where $\hat{A} = \hat{D}^{-\frac{1}{2}}\tilde{A}\hat{D}^{-\frac{1}{2}}$ denotes the normalized adjacency matrix, $\tilde{A} = A + I_N$ denotes adding the identity matrix $I_N$ to the adjacent matrix $A$. $\hat{D}$ is the diagonal matrix with on-diagonal element as $\hat{D}_{ii} = \sum_j \tilde{A}_{ij}$. $W^{(0)}$ and $W^{(1)}$ are the weights of first and second layer of GCN, respectively. ReLU(0, a) = max$(0, a)$ is adopted. The loss function $\mathcal{L}$ in GCN is cross entropy.

The notations we use throughout the paper are summarized in Table 1.

## 4 PROPOSED FRAMEWORK

To perform the non-target node injecting poisoning attack, we propose to solve the optimization problem in Eq.(2) via deep reinforcement learning. Compared with directly optimizing the adjacency matrix with traditional matrix optimization techniques, the advantages of adopting deep reinforcement learning are two folds: **(i)** Adding edges and designing labels of fake nodes are naturally sequential decision making process. Thus, deep reinforcement learning is a good fit for the problem [36]; **(ii)** The underlying structures of graphs are usually highly nonlinear [40], which adds the non-linearity to the decision making process. The deep non-linear neural networks of the Q network could better capture the graph non-linearity and learn the semantic meaning of the graph for making better decisions.

**Table 1: Notations and Explanations**

| Notation | Explanation |
|---|---|
| $V_{\mathcal{A}}$ | Adversarial node set |
| $V'$ | Poisoned node set, $V \cup V_{\mathcal{A}}$ |
| $E_{\mathcal{A}}$ | Adversarial Edge set |
| $E'$ | Poisoned Edge set, $E \cup E_{\mathcal{A}}$ |
| $\mathcal{T}$ | Labeled node sets |
| $\mathcal{U}$ | Unlabeled node set, $V \setminus \mathcal{T}$ |
| $G'$ | Poisoned graph \$ $C_\theta$ |
| $C_\theta(G')$ | Prediction of classifier $C$ on $G'$ |
| $L$ | Label sets |
| $\Delta$ | Poisoning budget |
| $G'_t$ | Poisoned graph at time $t$ |
| $\mathcal{T}_{\mathcal{A}_t}$ | Labels of Adversarial nodes at time $t$ |
| $z_{\mathcal{A}_t}$ | One-hot encoding of $\mathcal{T}_{\mathcal{A}_t}$ at time $t$ |
| $s_t = \{G'_t, \mathcal{T}_{\mathcal{A}_t}\}$ | State at time $t$ |
| $a_t = (a_t^{(1)}, a_t^{(2)}, a_t^{(3)})$ | Hierarchical action at time $t$ |
| $r_t$ | Reward function at time $t$ |
| $\pi(s)$ | Policy of state to action distribution |
| $Q = \{Q^{(1)}, Q^{(2)}, Q^{(3)}\}$ | Hierarchical action-value functions |
| $l_{a_t^{(1)}}$ | Labels of fake node $a_t^{(1)}$ at time $t$ |

An illustration of the proposed framework is shown in Figure 2. The key idea of our proposed framework is to train the deep reinforcement learning agent which could iteratively perform actions to poison the graph. The actions includes adding adversarial edges and modifying the labels of injected nodes. More specifically, the agent needs to firstly pick one node from injected nodes set $V_{\mathcal{A}}$ and select another node from poisoned node set $V'$ to add the adversarial edge, and modify the label of the injected nodes to attack the classifier $C$. We design reinforcement learning environment and reward according to the optimization function to achieve this.

Next, we describe the details of the proposed method and present the reinforcement learning environment design, the deep Q network used to estimate the policy and the training algorithm of the proposed NIPA.

## 4.1 Attacking Environment

We model the proposed poisoning attack procedure as a Finite Horizon Markov Decision Process $(S, A, P, R, \gamma)$. The definition of the MDP contains state space $S$, action set $A$, transition probability $P$, reward $R$, discount factor $\gamma$.

*4.1.1 State.* The state $s_t$ contains the intermediate poisoned graph $G'_t$ and labels information $\mathcal{T}_{\mathcal{A}_t}$ of the injected nodes at the time $t$. To capture the highly non-linear information and the non-Euclidean structure of the poisoned graph $G'_t$, we embed $G'_t$ as $e(G'_t)$ with aggregating the graph structure information via designed graph neural networks. $e(\mathcal{T}_{\mathcal{A}_t})$ encodes the adversarial label information $L_{\mathcal{A}_t}$ with neural networks. The details of the state representation is described in following subsection. Since in the injection poisoning

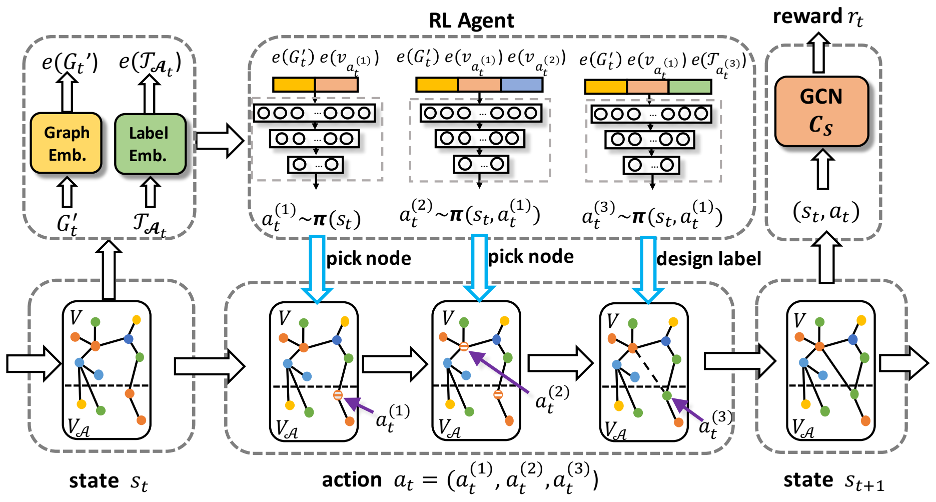
RL Agent
reward $r_t$
$e(G_t')$ $e(\mathcal{T}_{\mathcal{A}_t})$
Graph Emb.
Label Emb.
$G_t'$ $\mathcal{T}_{\mathcal{A}_t}$
$a_t^{(1)} \sim \boldsymbol{\pi}(s_t)$
$a_t^{(2)} \sim \boldsymbol{\pi}(s_t, a_t^{(1)})$
$a_t^{(3)} \sim \boldsymbol{\pi}(s_t, a_t^{(1)})$
GCN $C_S$
$(s_t, a_t)$
pick node
pick node
design label
state $s_t$
action $a_t = (a_t^{(1)}, a_t^{(2)}, a_t^{(3)})$
state $s_{t+1}$

**Figure 2: An overview of the Proposed Framework NIPA for Node Injection Attack on Graphs**

environment, the node set $V'$ remains identical thus the DRL agent performs poisoning actually on the edge set $E_t'$.

*4.1.2 Action.* In the poisoning attack environment, the agent is allowed to (1) add the adversarial edges within the injected nodes $V_{\mathcal{A}}$ or between the injected nodes and the clean nodes; (2) designing the adversarial labels of the injected nodes. However, directly adding one adversarial edge has $O(|V_{\mathcal{A}}|^2 + |V_{\mathcal{A}}| * |V|)$ possible choices and modifying the adversarial label of one injected node requires $O(|L|)$ space where $|L|$ is the number of label categories. Thus, performing one action that contains both adding an adversarial edge and changing the label of a node has search space as $O(|V_{\mathcal{A}}| * |V'| * |L|)$, which is extremely expensive especially in large graphs. Thus, we adopt hierarchical action to decompose such action and reduce the action space to enable efficient exploration inspired by previous work [12].

As shown in Figure 2, in NIPA, at time $t$, the agent first performs an action $a_t^{(1)}$ to select one injected node from $V_{\mathcal{A}}$. The agent then picks another node from the whole node set $V'$ via action $a_t^{(2)}$. After performing the actions $a_t^{(1)}$ and $a_t^{(2)}$, the agent connects these two selected nodes to forge the adversarial edge as the dashed line in the Figure 2. Finally, the agent designs the label of the selected fake node through action $a_t^{(3)}$. By such hierarchical action $a_t = (a_t^{(1)}, a_t^{(2)}, a_t^{(3)})$, the action space is reduced from $O(|V_{\mathcal{A}}| * |V'| * |L|)$ to $O(|V_{\mathcal{A}}| + |V'| + |L|)$. With the hierarchy action $a = (a^{(1)}, a^{(2)}, a^{(3)})$, the trajectory of the proposed MDP is $(s_0, a_0^{(1)}, a_0^{(2)}, a_0^{(3)}, r_0, s_1, \dots, s_{T-1}, a_{T-1}^{(1)}, a_{T-1}^{(2)}, a_{T-1}^{(3)}, r_{T-1}, s_T)$.

*4.1.3 Policy network.* As both of previous work [12] and our preliminary experiments show that Q-learning works more stable than other policy optimization methods such as Advantage Actor Critic, we focus on modeling policy network with Q-learning. Q-learning finds a policy that is optimal in the sense that it maximizes the expected value of the total reward over any and all successive steps, starting from the current state. Q-learning is an off-policy optimization which fits the Bellman optimality equation as:

$$Q^*(s_t, a_t) = r(s_t, a_t) + \gamma \max_{a_t'} Q^*(s_{t+1}, a') \quad (6)$$

The greedy policy to select the action $a_t$ with respect to $Q^*$ is:

$$a_t = \pi(s_t) = \arg\max_a Q^*(s_t, a) \quad (7)$$

As we explain in the above subsection that performing one poisoning action requires searching in $O(|V_{\mathcal{A}}| * |V'| * |L|)$ space and we perform hierarchical actions other than one action, we cannot directly follow the policy network in Eq.(6) and Eq.(7). Here, we adopt hierarchical Q function for the actions and we propose the hierarchical framework which integrates three DQNs. The details of the proposed DQNs are presented in following section.

*4.1.4 Reward.* As the RL agent is trained to enforce the misclassification of the graph classifier $C$, we need to design the reward accordingly to guide the agent. The reasons why we need to design novel reward function other than using the widely adopted binary sparse rewards are two folds: (1) as our trajectory in the attacking environment is usually long, we need the intermediate rewards which give feedback to the RL agent on how to improve its performance on each state; (2) different from the target attack that we know whether the attack on one targeted node is success or not, we perform the non-target attack over graph thus accuracy is not binary. To tackle theses two challenges, we design the reward of the current state and actions for the agent is designed according to the poisoning objective function shown in Eq. (2). For each state $s_t$, we firstly define the attack success rate $\mathcal{A}_t$ as:

$$\mathcal{A}_t = \sum_{v_j \in \mathcal{T}} \mathbb{1}(\mathcal{T}_j \neq C_{\theta_S}(G_t')_j)/|\mathcal{V}| \quad (8)$$

$$\theta_S = \arg\min_{\theta} \sum_{v_i \in \mathcal{T}'} \mathcal{L}(\mathcal{T}_i', C_\theta(G')_i) \tag{9}$$

Here $\mathcal{T}$ is the training set used to compute the reward as we discussed for the Eq.(2). Note that the $C_{\theta_S}$ is not the graph classifier $C$ that evaluates the final classification accuracy by end-user. As the attacker usually doesn't know the model that end-user is using, it represents the simulated graph classifier designed by attacker to acquire the state and actions reward. However, directly using the success rate $\mathcal{A}_t$ as the reward would increase the instability of training process since the accuracy might not differ a lot for two consecutive state. In this case, we design the guiding binary reward $r_t$ to be one if the action $a_t = (a_t^{(1)}, a_t^{(2)}, a_t^{(3)})$ could reduce the accuracy of attacker's simulated graph classifier $C_{\theta_S}$ at time $t$, and to be negative one vice versa. The proposed guiding reward $r_t$ is defined as follows:

$$r_t(s_t, a_t^{(1)}, a_t^{(2)}, a_t^{(3)}) = \begin{cases} 1; & \text{if } \mathcal{A}_{t+1} > \mathcal{A}_t \\ -1; & \text{otherwise.} \end{cases} \tag{10}$$

Our preliminary experimental results show that such guiding reward is effective in our case.

*4.1.5 Terminal.* In the poisoning attacking problem, the number of allowed adding adversarial edges is constrained by the budget $\Delta$ for the unnoticeable consideration. So in the poisoning reinforcement learning environment, once the agent adds budget number of edges ($T = \Delta$), it stops taking actions. In terminal state $s_T$, the poisoned graph $G'$ contains $T$ more adversarial edges compared to the clean graph $G$.

## 4.2 State Representation

As mentioned above, the state $s_t$ contains the poisoned graph $G_t'$ and injected nodes labels $\mathcal{T}_{\mathcal{A}_t}$ at time $t$. It is important to explore the high nonlinear structure of the state as the Q function is scoring the nodes in the poisoned graph $G_t'$. As shown in Figure 2, NIPA represents the state $s_t$ by the $e(G_t')$ and $e(\mathcal{T}_{\mathcal{A}_t})$ via graph embedding and label embedding methods. Here, in details, to represent the non-Euclidean structure of the poisoned graph $G_t'$ with vector $e(G_t')$, the latent embedding $e(v_i)$ of the each node $v_i$ in $G_t'$ is firstly learned by struct2vec [11] using the discriminative information. Then the state vector representation $e(G_t')$ is obtained by aggregating the embedding of nodes as:

$$e(G_t') = \sum_{v_i \in V'} e(v_i)/|V'| \tag{11}$$

To represent the label of the injected fake nodes, we use the two layer neural networks to encode the one-hot embedding of the nodes labels $\mathcal{T}_{\mathcal{A}_t}$ as:

$$e(\mathcal{T}_{\mathcal{A}_t}) = \sigma(W_l^{(2)}(\sigma(W_l^{(1)} z_{\mathcal{A}_t} + b_1) + b_2) \tag{12}$$

Here, $z_{\mathcal{A}_t}$ represents the one-hot embedding of the labels $T_{\mathcal{A}_t}$, $\sigma$ is the non-linear activation function and $\{W_l^{(1)}, W_l^{(2)}, b_1, b_2\}$ are the parameters of neural networks.

Actually, more complex graph embedding and label embedding methods could replace the adopted modules outlined here and we leave exploring feasible graph embedding and label embedding methods as a future direction. Note that for the notation compact and consistency consideration, $e(s)$ represents embedding of the state, and $e(v_a)$ and $e(\mathcal{T}_a)$ are the embeddings of the node selected by action $a$ and label selected by action $a$ respectively in the following paper.

## 4.3 Hierarchical Q Network

In Q learning process, given the state $s_t$ and action $a_t$, the action-value function $Q(s_t, a_t)$ is supposed to give the scores of current state and selected actions to steer the RL agent. However, as the action $a$ is decomposed into three hierarchical actions $\{a^{(1)}, a^{(2)}, a^{(3)}\}$ for the efficiency searching consideration, it would be hard to directly design the $Q(s_t, a_t^{(1)}, a_t^{(2)}, a_t^{(3)})$ and apply one policy network to select hierarchical actions.

To overcome this problem, we adopt hierarchical deep Q networks $Q = \{Q^{(1)}, Q^{(2)}, Q^{(3)}\}$ which integrates three DQNs to model the Q values over the actions. Figure 2 illustrates the selection of action $a_t = \{a_t^{(1)}, a_t^{(2)}, a_t^{(3)})\}$ at time $t$ performed by our proposed NIPA. After obtaining the state representations $e(s_t)$, the first DQN $Q^{(1)}$ guides the policy to pick a node from injected node set $V_{\mathcal{A}}$; Based on $a_t^{(1)}$, the second DQN $Q^{(2)}$ learns the policy to pick the second node from the node set $V'$, which completes an edge injection by connecting the two nodes; The third DQN $Q^{(3)}$ learns the policy to design the label of the first selected injected fake node.

The agent firstly selects one node from the injected node set $V_{\mathcal{A}}$ and calculate the $Q$ value based on the action-value function $Q^{(1)}$ as:

$$Q^{(1)}(s_t, a_t^{(1)}; \theta^{(1)}) = W_1^{(1)} \sigma(W_2^{(1)}[e(s_t) \parallel e(v_{a_t^{(1)}})]) \tag{13}$$

where $\theta^{(1)} = \{W_1^{(1)}, W_2^{(1)}\}$ represents the trainable weights of the first DQN and $\parallel$ is the concatenation operation. The action-value function $Q^{(1)}$ estimates the Q value of each injected fake node given the state representation $s_t$ and action embedding $e(v_{a_t^{(1)}})$. The greedy policy which selects the action $a_t^{(1)}$ based on optimal action-value function $Q^{(1)*}$ in Eq.(13) is defined as follows:

$$a_t^{(1)} = \pi(s_t) = \arg\max_{a \in V_{\mathcal{A}}} Q^{(1)}(s_t, a; \theta^{(1)}); \tag{14}$$

With the first action $a_t^{(1)}$ selected, the agent picks the second action $a_t^{(2)}$ hierarchically based on $Q^{(2)}$ as:

$$Q^{(2)}(s_t, a_t^{(1)}, a_t^{(2)}; \theta^{(2)}) = W_1^{(2)} \sigma(W_2^{(2)}[e(s_t) \parallel e(v_{a_t^{(1)}}) \parallel e(v_{a_t^{(2)}})]) \tag{15}$$

where $\theta^{(2)} = \{W_1^{(2)}, W_2^{(2)}\}$ is the trainable weights. The action value function $Q^{(2)}$ scores the second nodes based on the state $s_t$, and the selected action $a_t^{(1)}$. The greedy policy to make the second action $a_t^{(2)}$ with the optimal $Q^{(2)*}$ in Eq.(15) is defined as follows:

$$a_t^{(2)} = \pi(s_t, a_t^{(1)}) = \arg\max_{a \in V'} Q^{(2)}(s_t, a_t^{(1)}, a; \theta^{(2)}); \tag{16}$$

Note that the agent only modifies the label of the selected injected fake node $a_t^{(1)}$, thus the action-value function for the third action is not directly related to the action $a_t^{(2)}$. The action-value function $Q^{(3)}$ which scores the injected fake node label designing is defined as follows:

$$Q^{(3)}(s_t, a_t^{(1)}, a_t^{(3)}; \theta^{(3)}) = W_1^{(3)} \sigma(W_2^{(3)}[e(s_t) \parallel e(v_{a_t^{(1)}}) \parallel e(\mathcal{T}_{a_t^{(3)}})]) \tag{17}$$

In Eq.(17), $\theta^{(3)} = \{W_1^{(3)}, W_2^{(3)}\}$ represents the trainable weights in $Q^{(3)}$. The action value function $Q^{(3)}$ models the score of changing the label of the injected node $a_t^{(1)}$. The greedy policy to such action is defined as follows:

$$a_t^{(3)} = \pi(s_t, a_t^{(1)}) = \arg\max_{a \in L} Q^{(3)}(s_t, a_t^{(1)}, a; \theta^{(3)}); \tag{18}$$

With the proposed hierarchical deep Q networks $Q = \{Q^{(1)}, Q^{(2)}, Q^{(3)}\}$ in Eq.(13), Eq.(15) and Eq.(17), NIPA integrates hierarchical action-value functions to model the Q values over the hierarchical actions $a = \{a^{(1)}, a^{(2)}, a^{(3)}\}$.

## 4.4 Training Algorithm

To train the proposed hierarchical DQNs $Q = \{Q^{(1)}, Q^{(2)}, Q^{(3)}\}$ and the parameters in states representation methods, we adopt the experience replay technique with a certain size memory buffer $\mathcal{M}$ [30]. The high level idea to use the experience replay is to reduce bias caused by correlation between samples. We simulate the selection process to generate training data and store them in memory buffer $\mathcal{M}$. During training, a batch of experience $(s, a, s')$ where $a = \{a^{(1)}, a^{(2)}, a^{(3)}\}$ is drawn uniformly at random from the stored memory buffer $\mathcal{M}$. The Q-learning loss function is defined as:

$$\mathbb{E}_{(s,a,s') \sim \mathcal{M}}[(r + \gamma \max_{a'} \hat{Q}(s', a'|\theta^-) - Q(s, a|\theta))^2] \tag{19}$$

where $\hat{Q}$ represents the target action-value function and its parameters $\theta^-$ are updated with $\theta$ every C steps. To improve the stability of the algorithm, we clip the error term between −1 and +1. The agent adopts $\epsilon$-greedy policy that select a random action with probability $\epsilon$. The overall training framework is summarized in Algorithm 1.

In the proposed model, we use two layer multi-layer perceptrons to implement all the trainable parameters $\theta$ in action-value functions $Q = \{Q^{(1)}, Q^{(2)}, Q^{(3)}\}$ and structure2vec. Actually, more complex deep neural networks could replace the models outlined here. We leave exploring feasible deep neural networks as a future direction.

**Algorithm 1:** The training algorithm of framework NIPA

```
Input: clean graph $G(V, E, X)$, labeled node set $\mathcal{T}$, budget $\Delta$,
       number of injected nodes $|V_{\mathcal{A}}|$, training iteration $K$
Output: $G'(V', E', X')$ and $L_{\mathcal{A}}$
1  Initialize action-value function Q with random parameters $\theta$ ;
2  Set target function $\hat{Q}$ with parameters $\theta^- = \theta$;
3  Initialize replay memory buffer $\mathcal{M}$;
4  Randomly assign Adversarial label $L_{\mathcal{A}}$;
5  while episode $\leq K$ do
6      while $t \leq \Delta$ do
7          Compute state representation according to Eq.(11) and Eq.(12);
8          With probability $\epsilon$ select a random action $a_t^{(1)}$,
             otherwise select $a_t^{(1)}$ based on Eq.(14);
9          With probability $\epsilon$ select a random action $a_t^{(2)}$,
             otherwise select $a_t^{(2)}$ based on Eq.(16);
10         With probability $\epsilon$ select a random action $a_t^{(3)}$,
             otherwise select $a_t^{(3)}$ based on Eq.(18);
11         Compute reward $r_t$ according to Eq.(8) and Eq.(10);
12         Set $s_{t+1} = \{s_t, a_t^{(1)}, a_t^{(2)}, a_t^{(3)}\}$;
13         Update edges as $E_{\mathcal{A}_{t+1}} \leftarrow E_{\mathcal{A}} \cup (a_t^{(1)}, a_t^{(2)})$ and
             labels as $l_{a_{t+1}^{(1)}} \leftarrow a_t^{(3)}$ ;
14         Store $\{s_t, a_t^{(1)}, a_t^{(2)}, a_t^{(3)}, r_t, s_{t+1}\}$ in memory buffer $\mathcal{M}$;
15         Sample minibatch transition randomly from $\mathcal{M}$;
16         Update parameter according to Eq.(19);
17         Every C steps update $\theta^- = \theta$;
18     end
19 end
```

## 5 EXPERIMENTS

In this section, we introduce the experiment setting including baseline datasets and comparing poisoning attack methods. Moreover, we conduct experiments and present results to answer the following research questions:

- (**RQ1**) Can the NIPA effectively poison the graph data and attack the GCN via node injection?
- (**RQ2**) Whether the poisoned graph remains the key statistics after the poison attack?
- (**RQ3**) How the proposed framework performances under different scenarios?

Next, we first introduce the experimental settings followed by experimental results to answer the three questions.

### 5.1 Experiment Setup

*5.1.1 Datasets.* We conduct experiments on three widely used benchmark datasets for node classification, which include CORA-ML [5, 28], CITESEER [17] and DBLP [32]. Following [52], we only consider the largest connected component (LCC) of each graph data. The statistics of the datasets with the LCC are summarized in Table 2. For each dataset, we randomly split the nodes into (20%) labeled nodes for training procedure and (80%) unlabeled nodes as test set to evaluate the model. The labeled nodes are further equally split into training and validation sets. We perform the random split five times and report averaged results.

**Table 2: Statistics of benchmark datasets**

| Datasets | $N_{LCC}$ | $E_{LCC}$ | \|L\| |
|---|---|---|---|
| CITESEER | 2,110 | 3,757 | 6 |
| CORA-ML | 2,810 | 7,981 | 7 |
| PUBMED | 19,717 | 44,324 | 3 |

*5.1.2 Baseline Methods.* Though there are several adversarial attack algorithms on graphs such as Nettack [51] and RL-S2v [12], most of them are developed for manipulating links among existing nodes in graph, which cannot be easily modified in our attacking setting for node injection attack. Thus, we don't compare with them. Since node injection attack on graphs is a novel task, there are very few baselines that we can compare with. We select following four baselines, with two from classical graph generation models, one by applying the technique of fast gradient attack and a variant of NIPA.

- Random Attack [15]: The attacker $\mathcal{A}$ first adds adversarial edges between the injected nodes according to classic Erdős–Rényi model $G(V_{\mathcal{A}}, p)$, where the probability $p = \frac{2|E|}{|V|^2}$ is the average degree of the clean graph $G(V, E)$ to make sure the density of the injected graph $G_A$ is similar to the clean graph. The attacker then randomly add adversarial edges connecting the injected graph $G_A$ and clean graph $G$ until the budget $\Delta$ is used ups.
- Preferential attack [2]: The attacker $\mathcal{A}$ iteratively adds the adversarial edges according to preferential attachment mechanism. The probability of connecting the injected node $v_i \in V_{\mathcal{A}}$ to the other node $v_j \in |V \cup V_{\mathcal{A}}|$ is proportional to the node degrees. The number of adversarial edges is constrained by the budget $\Delta$.
- Fast Gradient Attack (FGA) [10]: FGA is a gradient based method which is designed to attack the graph data by perturbing the gradients. In FGA, the attacker $\mathcal{A}$ removes/adds the adversarial edges guided by edge gradient.
- NIPA-w/o: This is a variant of the proposed framework NIPA where we don't optimize w.r.t the label of fake nodes, i.e., the labels of the fake nodes are randomly assigned.

The Fast Gradient Attack (FGA) [10] is not directly applicable in node injection poisoning setting, since the injected nodes are isolated at the beginning and would be filtered out by graph classifier. Here we modify the FGA for fair comparison. The FGA method is performed on the graph poisoned by preferential attack. After calculating the gradient $\nabla_{ij} L_{GCN}$ with $v_i \in V_{\mathcal{A}}$ and $v_j \in V'$, the attack $\mathcal{A}$ adding/remove the adversarial edges between $(v_i, v_j)$ according to the largest positive/negative gradient. The attack only add and remove one feasible adversarial edge are each iteration so that the number of the adversarial edges is still constrained by budget $\Delta$. The attacker is allowed to perform 20*$\Delta$ times modifications in total suggested by [10].

## 5.2 Attack Performance Comparison

To answer **RQ1**, we evaluate how the node classification accuracy degrades on the poisoned graph compared with the performance on the clean graph. The larger decrease the performance is on the poisoned graph, the more effective the attack is. To compare the decrease the of different models, we firstly show the node classification results on clean graph.

*5.2.1 Node Classification on Clean Graph.* As the Nettack [51] points out that "poisoning attacks are in general harder and match better the transductive learning scenario", we follow the same poisoning transductive setting in this paper. The parameters of GCN are trained according to Eq. (1). We report the averaged node classification accuracy over five runs in Table. 3 to present the GCN node classification accuracy on clean graph. Note that if the poisoning nodes are injected with the budget $\Delta = 0$, such isolated nodes would be filtered out by GCN and the classification results remain the same as in Table. 3.

**Table 3: Node classification results on clean graph**

| Dataset | CITESEER | CORA-ML | Pubmed |
|---|---|---|---|
| Clean data | 0.7730 ± 0.0059 | 0.8538 ± 0.0038 | 0.8555 ± 0.0010 |

*5.2.2 Node Classification on Poisoned Graph.* In poisoning attacking process, the attacking budget $\Delta$ which controls the number of added adversarial edges is one important factor. On the one hand, if the budget is limited, eg., $\Delta < |V_{\mathcal{A}}|$, then at least $|V_{\mathcal{A}}| - \Delta$ injected nodes are isolated. Clearly, isolated nodes have no effect on the label prediction as they are not really injected into the environment. On the other hand, if the budget is large, the density of the injected graph is different from the clean graph and such injected nodes might be detected by the defense methods. Here, to make the poisoned graph has the similar density with the clean graph and simulates the real world poisoning attacking scenario, we set $\Delta = r * |V| * deg(V)$ where $r$ is the injected nodes ratio compared to the clean graph and $deg(V)$ is the average degree of the clean graph $G$. The injected nodes number is $|V_{\mathcal{A}}| = r * |V|$. We will evaluate how effective the attack is when the injected nodes can have different number of degrees in Section 5.4.1. To have comprehensive comparisons of the methods, we vary $r$ as $r = \{0.01, 0.02, 0.05, 0.10\}$. We don't set $r > 0.10$ since we believe that too much injected nodes could be easily noticed in real-world scenarios. For the same unnoticeable consideration, the feature of the injected nodes is designed to be similar to that of the clean node features. For each injected node, we calculate the mean of the features as $\bar{X}$ and apply the Gaussian noise $\mathcal{N}(0, 1)$ on the averaged features $\bar{X}$. The features of the injected node are similar to the features in clean graph. We leave the generation of node features as future work. As the other baselines method could not modifies the adversarial labels of the injected nodes, we also provide the variant model NIPA-w/o which doesn't manipulate the adversarial labels for fair comparison. The adversarial labels are randomly generated within $|L|$ for the baseline methods. In both NIPA and NIPA-w/o, we set the discount factor $\gamma = 0.9$ and the injected nodes $V_{\mathcal{A}}$ are only appear in training phase in all of the methods.

The averaged results with standard deviation for all methods are reported in Table 4. From Table 3 and 4, we make the following observations

- In all attacking methods, more injected nodes could better reduce the node classification accuracy, which satisfy our expectation;
- Compared with Random and Preferential attack, FGA is relatively more effective in attacking the graph, though the performance gap is marginal. This is because random attack and preferential attack don't learn information from the clean graph and just insert fake nodes following predefined

**Table 4: Classification results after adversarial attack on graphs**

| Dataset | Methods | $r = 0.01$ | $r = 0.02$ | $r = 0.05$ | $r = 0.10$ |
|---|---|---|---|---|---|
| CITESEER | Random | 0.7582 ± 0.0082 | 0.7532 ± 0.0130 | 0.7447 ± 0.0033 | 0.7147 ± 0.0122 |
| | Preferrential | 0.7578 ± 0.0060 | 0.7232 ± 0.0679 | 0.7156 ± 0.0344 | 0.6814 ± 0.0131 |
| | FGA | 0.7129 ± 0.0159 | 0.7117 ± 0.0052 | 0.7103 ± 0.0214 | 0.6688 ± 0.0075 |
| | NIPA-wo(ours) | 0.7190 ± 0.0209 | 0.6914 ± 0.0227 | 0.6778 ± 0.0162 | 0.6301 ± 0.0182 |
| | NIPA (ours) | **0.7010** ± 0.0123 | **0.6812** ± 0.0313 | **0.6626** ± 0.0276 | **0.6202** ± 0.0263 |
| CORA-ML | Random | 0.8401 ± 0.0226 | 0.8356 ± 0.0078 | 0.8203 ± 0.0091 | 0.7564 ± 0.0192 |
| | Preferrential | 0.8272 ± 0.0486 | 0.8380 ± 0.0086 | 0.8038 ± 0.0129 | 0.7738 ± 0.0151 |
| | FGA | 0.8205 ± 0.0044 | 0.8146 ± 0.0041 | 0.7945 ± 0.0117 | 0.7623 ± 0.0079 |
| | NIPA-w/o (ours) | 0.8042 ± 0.0190 | 0.7948 ± 0.0197 | 0.7631 ± 0.0412 | 0.7206 ± 0.0381 |
| | NIPA (ours) | **0.7902** ± 0.0219 | **0.7842** ± 0.0193 | **0.7461** ± 0.0276 | **0.6981** ± 0.0314 |
| PUMBED | Random | 0.8491 ± 0.0030 | 0.8388 ± 0.0035 | 0.8145 ± 0.0076 | 0.7702 ± 0.0126 |
| | Preferrential | 0.8487 ± 0.0024 | 0.8445 ± 0.0035 | 0.8133 ± 0.0099 | 0.7621 ± 0.0096 |
| | FGA | 0.8420 ± 0.0182 | 0.8312 ± 0.0148 | 0.8100 ± 0.0217 | 0.7549 ± 0.0091 |
| | NIPA-w/o(ours) | 0.8412 ± 0.0301 | 0.8164 ± 0.0209 | 0.7714 ± 0.0195 | 0.7042 ± 0.0810 |
| | NIPA (ours) | **0.8242** ± 0.0140 | **0.8096** ± 0.0155 | **0.7646** ± 0.0065 | **0.6901** ± 0.0203 |

rule. Thus, both of the methods are not as effective as FGA which tries to inject nodes through a way to decrease the performance;

- The proposed framework outperforms the other methods. In particular, both FGA and NIPA are optimization based approach while NIPA significantly outperforms FGA, which implies the effectiveness of the proposed framework by designing hierarchical deep reinformcent learning to solve the decision making optimization problem; and
- NIPA out performances NIPA-w/o, which shows the necessity of optimizing w.r.t to labels for node injection attack.

## 5.3 Key Statistics of the Poisoned Graphs

To answer **RQ2**, we analyze several key statistics of the poisoned graphs, which helps us to understand the attacking behaviors. One desired property of the poisoning graph is that the poisoned graph has similar graph statistics to the clean graph. Here, we adopt the important graph statistics as that used in [6] to measure the poisoned graphs for the three datasets. More specifically, we presents the Gini coefficient, characteristic path length, distribution entropy, power law exponent and numbers of triangle counts to carefully analysis the poisoned graph statistics such as graph distribution and graph density. The detailed equations and descriptions could be found [6]. The results are reported in Table 5. It could be concluded from the graph statistics that

- Poisoned graph has very similar graph distribution to the clean graph. For example, the similar exponent of the power law distribution in graph indicates that the poisoned graph and the clean graph shares the similar distribution;
- More injected nodes would make the poisoning attack process noticeable. The results show that with the increase of $r$, the poisoned graph becomes more and more diverse from the origin graph.
- The number of triangles increases, which shows that the attack not just simply connect fake nodes to other nodes, but also connect in a way to form triangles so each connection could affects more nodes.

## 5.4 Attack Effects Under Different Scenarios

In this subsection, we conduct experiments to answer **RQ3**, i.e., how effective the attack by NIPA is under different scenarios. We carefully study the effective of different parameters to NIPA.

*5.4.1 Average Degrees of Injected Nodes.* As we discussed that the budget $\Delta = r * |V| * \deg(v_{\mathcal{A}})$ is essential to the poisoning attack, we investigate the node classification accuracy by varying the average degree of injected nodes as $\deg(v_{\mathcal{A}}) = \{3, \ldots 10\}$. We don't set $\deg(v_{\mathcal{A}}) > 10$ since injecting 'celebrate' or 'hub' nodes into the network is not common in graph poisoning attack.

The experiment results with injected node ratio $r = 0.01$ and $r = 0.02$ on CITESEER and CORA-ML are shown in Fig. 3(a) and Fig. 3(b), respectively. From the figures, we observe that as the increase of the average degree of the injected nodes, the node classification accuracy decrease sharply. Such observation satisfies our expectation because the more links a fake node can have, the more likely it can propagate the adversarial information and poison the graph.

*5.4.2 Sparsity of the Origin Graph.* We further investigate how the proposed framework works under different sparsity of the network. Without loss of generality, we set average degree of injected node as the average degree of the real node. To simulate the sparsity of the network, we randomly remove $S_p = \{0, 10\%, \ldots, 90\%\}$ edges from the original graph and perform NIPA on the modified graph. The results with injected node ratio $r = 0.01$ and $r = 0.02$ on CITSEER and CORA-ML are shown in Fig.4(a) and Fig.4(b) respectively.

The results show that as the graph becomes more spare, the proposed framework is more effective in attacking the graph. This is because as the graph becomes more sparse, each node in the clean graph has less neighbors, which makes the it easier for fake nodes to change the labels of unlabeled nodes.

**Table 5: Statistics of the clean graph ($r = 0.00$) and the graphs poisoned by NIPA averaged over 5 runs.**

| Dataset | $r$ | Gini Coefficient | Characteristic Path Length | Distribution Entropy | Power Law Exp. | Triangle Count |
|---|---|---|---|---|---|---|
| CITESEER | 0.00 | 0.4265 ± 0.0000 | 9.3105 ± 0.0000 | 0.9542 ± 0.0000 | 2.0584 ± 0.0000 | 1083.0 ± 0.0 |
| | 0.01 | 0.4270 ± 0.0012 | 8.3825 ± 0.3554 | 0.9543 ± 0.0001 | 2.0296 ± 0.0024 | 1091.2 ± 6.6 |
| | 0.02 | 0.4346 ± 0.0007 | 8.3988 ± 0.2485 | 0.9529 ± 0.0005 | 2.0161 ± 0.0007 | 1149.8 ± 32.4 |
| | 0.05 | 0.4581 ± 0.0026 | 8.0907 ± 0.7710 | 0.9426 ± 0.0009 | 1.9869 ± 0.0073 | 1174.2 ± 42.8 |
| | 0.10 | 0.4866 ± 0.0025 | 7.3692 ± 0.6818 | 0.9279 ± 0.0012 | 1.9407 ± 0.0088 | 1213.6 ± 61.8 |
| CORA-ML | 0.00 | 0.3966 ± 0.0000 | 6.3110 ± 0.0000 | 0.9559 ± 0.0000 | 1.8853 ± 0.0000 | 1558.0 ± 0.0 |
| | 0.01 | 0.4040 ± 0.0007 | 6.0576 ± 0.1616 | 0.9549 ± 0.0004 | 1.8684 ± 0.0016 | 1566.2 ± 7.4 |
| | 0.02 | 0.4075 ± 0.0002 | 6.1847 ± 0.1085 | 0.9539 ± 0.0002 | 1.8646 ± 0.0006 | 1592.0 ± 17.4 |
| | 0.05 | 0.4267 ± 0.0014 | 5.8165 ± 0.1018 | 0.9458 ± 0.0009 | 1.8429 ± 0.0027 | 1603.8 ± 12.8 |
| | 0.10 | 0.4625 ± 0.0005 | 6.1397 ± 0.0080 | 0.9261 ± 0.0007 | 1.8399 ± 0.0017 | 1612.4 ± 22.2 |
| PUBMED | 0.00 | 0.6037 ± 0.0000 | 6.3369 ± 0.0000 | 0.9268 ± 0.0000 | 2.1759 ± 0.0000 | 12520.0 ± 0.0 |
| | 0.01 | 0.6076 ± 0.0005 | 6.3303 ± 0.0065 | 0.9253 ± 0.0004 | 2.1562 ± 0.0013 | 12570.8 ± 29.2 |
| | 0.02 | 0.6130 ± 0.0006 | 6.3184 ± 0.0046 | 0.9213 ± 0.0004 | 2.1417 ± 0.0009 | 13783.4 ± 101.8 |
| | 0.05 | 0.6037 ± 0.0000 | 6.3371 ± 0.0007 | 0.9268 ± 0.0000 | 2.1759 ± 0.0001 | 14206.6 ± 152.8 |
| | 0.10 | 0.6035 ± 0.0003 | 6.2417 ± 0.1911 | 0.9263 ± 0.0010 | 2.1686 ± 0.0141 | 14912.0 ± 306.8 |

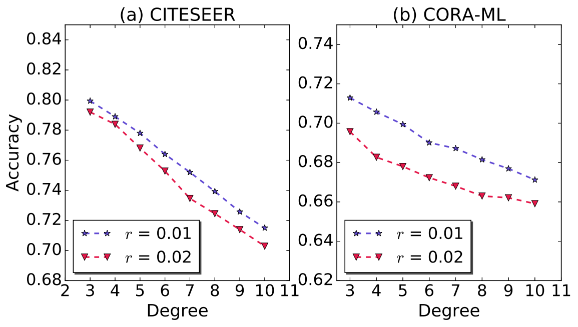

**Figure 3: Node classification performance on (a) CITESEER and (b) CORA-ML by varying average node degree of injected nodes**

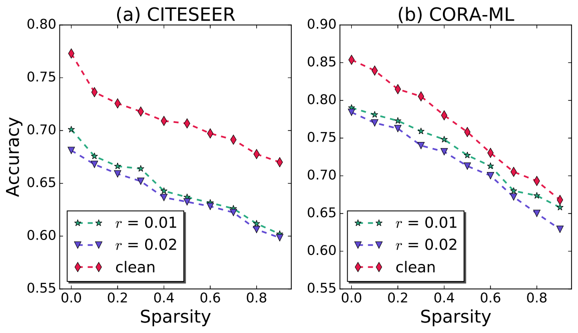

**Figure 4: Node classification performance on (a) CITESEER and (b) CORA-ML with varying graph sparsity**

## 6 CONCLUSION

In this paper, we study a novel problem of non-target graph poisoning attack via node injection. We propose a deep reinforcement learning based method named NIPA to solve the node injection poisoning attack task. NIPA simulates the attack process and sequentially adds the adversarial edges and designs labels for the injected fake nodes. Specifically, we adopt hierarchy deep Q networks to efficiently reduce the action spaces and use GNN based graph state representation method to cope with the graph topology. Experimental results of poisoning graph convolutional network on node classification demonstrate the effectiveness of the proposed framework for poisoning the graph. The poisoned graph has very similar properties as the original clean graph such as gini coefficient and distribution entropy. Further experiments are conducted to understand how the proposed framework works under different scenarios such as very sparse graph.

There are several interesting directions that need further investigation. First, we have used the mean of node features corrupted by Gaussian noise to set the features of fake nodes. It would be interesting to explore variants of NIPA that optimize the features of fake nodes to maximize the effectiveness of NIPA. Second, we have used a 2-layer graph neural networks to encode the states of the hierarchical deep Q learner. It would be interesting to explore more complex deep neural networks for this task. Moreover, it would be interesting to explore extensions of NIPA for carrying out node poisoning attacks on more complex graphs, e.g., heterogeneous graphs, multi-modal graphs, and dynamic graphs. Last, but not the least, it would be interesting to explore variants of NIPA that use more sophisticated optimization methods for reinforcement learning.

## ACKNOWLEDGMENTS

This work was funded in part by the NIH NCATS grant UL1 TR002014 and by NSF grants 1518732, 1640834, and 1636795, the Edward Frymoyer Endowed Professorship at Pennsylvania State University and the Sudha Murty Distinguished Visiting Chair in Neurocomputing and Data Science funded by the Pratiksha Trust at the Indian Institute of Science (both held by Vasant Honavar) and by Samsung GRO Award #225003 to Suhang Wang and Vasant Honavar.

## REFERENCES

[1] Charu C Aggarwal. 2011. An introduction to social network data analytics. In *Social network data analytics*. Springer, 1–15.

[2] Albert-László Barabási and Réka Albert. 1999. Emergence of scaling in random networks. *science* 286, 5439 (1999), 509–512.

[3] Battista Biggio, Blaine Nelson, and Pavel Laskov. 2012. Poisoning attacks against support vector machines. In *29th Int'l Conf. on Machine Learning (ICML)*.

[4] Battista Biggio and Fabio Roli. 2018. Wild patterns: Ten years after the rise of adversarial machine learning. *Pattern Recognition* 84 (2018), 317–331.

[5] Aleksandar Bojchevski and Stephan Günnemann. 2018. Deep Gaussian Embedding of Graphs: Unsupervised Inductive Learning via Ranking. In *International Conference on Learning Representations*. https://openreview.net/forum?id=r1ZdKJ-0W

[6] Aleksandar Bojchevski, Oleksandr Shchur, Daniel Zügner, and Stephan Günnemann. 2018. Netgan: Generating graphs via random walks. *arXiv preprint arXiv:1803.00816* (2018).

[7] Han Cai, Kan Ren, Weinan Zhang, Kleanthis Malialis, Jun Wang, Yong Yu, and Defeng Guo. 2017. Real-time bidding by reinforcement learning in display advertising. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*. ACM, 661–670.

[8] Nicholas Carlini and David Wagner. 2018. Audio adversarial examples: Targeted attacks on speech-to-text. In *2018 IEEE Security and Privacy Workshops (SPW)*. IEEE, 1–7.

[9] Christopher J Carpenter. 2012. Narcissism on Facebook: Self-promotional and anti-social behavior. *Personality and individual differences* 52, 4 (2012), 482–486.

[10] Jinyin Chen, Yangyang Wu, Xuanheng Xu, Yixian Chen, Haibin Zheng, and Qi Xuan. 2018. Fast gradient attack on network embedding. *arXiv preprint arXiv:1809.02797* (2018).

[11] Hanjun Dai, Bo Dai, and Le Song. 2016. Discriminative embeddings of latent variable models for structured data. In *International conference on machine learning*. 2702–2711.

[12] Hanjun Dai, Hui Li, Tian Tian, Xin Huang, Lin Wang, Jun Zhu, and Le Song. 2018. Adversarial attack on graph structured data. *arXiv preprint arXiv:1806.02371* (2018).

[13] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in neural information processing systems*. 3844–3852.

[14] Kien Do, Truyen Tran, and Svetha Venkatesh. 2019. Graph transformation policy network for chemical reaction prediction. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 750–760.

[15] Paul Erdős and Alfréd Rényi. 1960. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci* 5, 1 (1960), 17–60.

[16] Wenqi Fan, Yao Ma, Qing Li, Yuan He, Eric Zhao, Jiliang Tang, and Dawei Yin. 2019. Graph Neural Networks for Social Recommendation. In *The World Wide Web Conference*. ACM, 417–426.

[17] C Lee Giles, Kurt D Bollacker, and Steve Lawrence. 1998. CiteSeer: An Automatic Citation Indexing System.. In *ACM DL*. 89–98.

[18] Adam Gleave, Michael Dennis, Neel Kant, Cody Wild, Sergey Levine, and Stuart Russell. 2019. Adversarial Policies: Attacking Deep Reinforcement Learning. *arXiv preprint arXiv:1905.10615* (2019).

[19] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations*.

[20] Qingyu Guo, Zhao Li, Bo An, Pengrui Hui, Jiaming Huang, Long Zhang, and Mengchen Zhao. 2019. Securing the Deep Fraud Detector in Large-Scale E-Commerce Platform via Adversarial Machine Learning Approach. In *The World Wide Web Conference*. ACM, 616–626.

[21] Kun He, Yiwei Sun, David Bindel, John Hopcroft, and Yixuan Li. 2015. Detecting overlapping communities from local spectral subspaces. In *2015 IEEE International Conference on Data Mining*. IEEE, 769–774.

[22] Kwang-Sung Jun, Lihong Li, Yuzhe Ma, and Jerry Zhu. 2018. Adversarial attacks on stochastic bandits. In *Advances in Neural Information Processing Systems*. 3640–3649.

[23] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

[24] Ziqi Liu, Chaochao Chen, Xinxing Yang, Jun Zhou, Xiaolong Li, and Le Song. 2018. Heterogeneous graph neural networks for malicious account detection. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*. ACM, 2077–2085.

[25] Yuzhe Ma, Kwang-Sung Jun, Lihong Li, and Xiaojin Zhu. 2018. Data poisoning attacks in contextual bandits. In *International Conference on Decision and Game Theory for Security*. Springer, 186–204.

[26] Yao Ma, Suhang Wang, Charu C Aggarwal, and Jiliang Tang. 2019. Graph convolutional networks with eigenpooling. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 723–731.

[27] Yao Ma, Suhang Wang, Lingfei Wu, and Jiliang Tang. 2019. Attacking graph convolutional networks via rewiring. *arXiv preprint arXiv:1906.03750* (2019).

[28] 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.

[29] Shike Mei and Xiaojin Zhu. 2015. Using Machine Teaching to Identify Optimal Training-Set Attacks on Machine Learners. In *The 29th AAAI Conference on Artificial Intelligence*.

[30] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529.

[31] Michele Nitti, Luigi Atzori, and Irena Pletikosa Cvijikj. 2014. Friendship selection in the social internet of things: challenges and possible strategies. *IEEE Internet of things journal* 2, 3 (2014), 240–247.

[32] Shirui Pan, Jia Wu, Xingquan Zhu, Chengqi Zhang, and Yang Wang. 2016. Tri-party deep network representation. *Network* 11, 9 (2016), 12.

[33] Thomas Puschmann. 2017. Fintech. *Business & Information Systems Engineering* 59, 1 (2017), 69–76.

[34] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. 2015. Trust region policy optimization. In *International conference on machine learning*. 1889–1897.

[35] Yiwei Sun, Suhang Wang, Tsung-Yu Hsieh, Xianfeng Tang, and Vasant Honavar. 2019. Megan: A generative adversarial network for multi-view network embedding. *arXiv preprint arXiv:1909.01084* (2019).

[36] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction.*

[37] 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).

[38] Xianfeng Tang, Yandong Li, Yiwei Sun, Huaxiu Yao, Prasenjit Mitra, and Suhang Wang. 2020. Transferring Robustness for Graph Neural Network Against Poisoning Attacks. In *ACM Internatioal Conference on Web Search and Data Mining (WSDM)*.

[39] Binghui Wang and Neil Zhenqiang Gong. 2019. Attacking Graph-based Classification via Manipulating the Graph Structure. *ACM Conference on Computer and Communications Security (CCS)* (2019).

[40] Daixin Wang, Peng Cui, and Wenwu Zhu. 2016. Structural deep network embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 1225–1234.

[41] Jianyu Wang, Rui Wen, Chunming Wu, Yu Huang, and Jian Xion. 2019. FdGars: Fraudster Detection via Graph Convolutional Networks in Online App Review System. In *Companion Proceedings of The 2019 World Wide Web Conference*. ACM, 310–316.

[42] Tingwu Wang, Renjie Liao, Jimmy Ba, and Sanja Fidler. 2018. Nervenet: Learning structured policy with graph neural networks. (2018).

[43] Zeng Wei, Jun Xu, Yanyan Lan, Jiafeng Guo, and Xueqi Cheng. 2017. Reinforcement learning to rank with Markov decision process. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, 945–948.

[44] Huijun Wu, Chen Wang, Yuriy Tyshetskiy, Andrew Docherty, kai Lu, and Liming Zhu. 2019. Adversarial Examples on Graph Data: Deep Insights into Attack and Defense. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*.

[45] Han Xu, Yao Ma, Haochen Liu, Debayan Deb, Hui Liu, Jiliang Tang, and Anil Jain. 2019. Adversarial attacks and defenses in images, graphs and text: A review. *arXiv preprint arXiv:1909.08072* (2019).

[46] Ziyu Yao, Jayavardhan Reddy Peddamail, and Huan Sun. 2019. CoaCor: Code Annotation for Code Retrieval with Reinforcement Learning. In *The World Wide Web Conference*. ACM, 2203–2214.

[47] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. 2018. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 974–983.

[48] Jiaxuan You, Bowen Liu, Zhitao Ying, Vijay Pande, and Jure Leskovec. 2018. Graph convolutional policy network for goal-directed molecular graph generation. In *Advances in Neural Information Processing Systems*. 6410–6421.

[49] Yanwei Yu, Huaxiu Yao, Hongjian Wang, Xianfeng Tang, and Zhenhui Li. 2018. Representation learning for large-scale dynamic networks. In *International Conference on Database Systems for Advanced Applications*. Springer, 526–541.

[50] Yiming Zhang, Yujie Fan, Wei Song, Shifu Hou, Yanfang Ye, Xin Li, Liang Zhao, Chuan Shi, Jiabin Wang, and Qi Xiong. 2019. Your Style Your Identity: Leveraging Writing and Photography Styles for Drug Trafficker Identification in Darknet Markets over Attributed Heterogeneous Information Network. In *The World Wide Web Conference*. ACM, 3448–3454.

[51] Daniel Zügner, Amir Akbarnejad, and Stephan Günnemann. 2018. Adversarial Attacks on Neural Networks for Graph Data. In *SIGKDD*. 2847–2856.

[52] Daniel Zügner and Stephan Günnemann. 2019. Adversarial Attacks on Graph Neural Networks via Meta Learning. In *International Conference on Learning Representations (ICLR)*.