Non-target-specific Node Injection Attacks on Graph Neural Networks: 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 have achieved immense success for node classification with its power to explore the topological structure in graph data 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 performance, e.g., on the node classification task. 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 our experiments show that NIPA is consistently more effective than the baseline node injection attack methods for poisoning graph data used to train GNN on several benchmark data sets. We further show that the graphs poisoned by NIPA are statistically similar to the original (clean) graphs, thus enabling the attacks to evade detection.

WWW '20, April 20–24, 2020, Taipei, Taiwan
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-7023-3/20/04.
https://doi.org/10.1145/3366423.3380149

KEYWORDS

Adversarial Attack; Graph Poisoning; Reinforcement learning;

ACM Reference Format:
Yiwei Sun, Suhang Wang, Xianfeng Tang, Tsung-Yu Hsieh, and Vasant Honavar. 2020. Non-target-specific Node Injection Attacks on Graph Neural Networks: 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, 34, 48, 49], e-commerce[16, 46], and FinTech [25, 32]. 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 [25, 40]. Graph Neural Networks (GNNs) [13, 23], currently offer the state-of-the art approach to node classification in graph-structured data.

However, recent studies [12, 37, 43, 50] have shown that GNNs are vulnerable to adversarial attacks that perturb or *poison* the graphs used for training the GNNs. For example, Nettack [50] involves perturbing the targeted node's attributes and connectivity so as to adversely impact the performance of the node classifier. Such an attack can be target-specific [43, 50] so as to ensure that the GNN is fooled into misclassifying the targeted node; or non-target-specific [12], so as to reduce the overall accuracy of node classification in a graph. In this paper, we focus on the latter. However, the success of such attack strategy requires that the adversary is able to control the nodes targeted by the attack and manipulate their connectivity. In other words, poisoning the real-world graphs such as Facebook and twitter requires 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.

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

Thus, we need a more efficient way to poison the graphs to increase the node mis-classification 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 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, 30], 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. However, there has been little work on NIPA.

Therefore, in this paper, we investigate NIPA, a novel form of adversarial attack on GNN. Specifically, we address two key challenges: (i) How to 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) seek to inject two fake nodes into the clean graph in (a). Obviously, the "smart attacker" who carefully designs the links and labels of the injected nodes (shown using dashed lines) could more effectively poison the training graph than the "naive attacker" who randomly establishes the links between the fake node and other nodes and randomly generates the attributes of the fake node and links between it and other nodes; and (ii) How to efficiently solve the resulting nonlinear discrete optimization problem. To address these two challenges, we propose NIPA, a novel family of attacks on GNN. We observe that the sequential addition the adversarial connections and the design of adversarial labels for the injected fake nodes can be naturally formulated as a Markov decision process (MDP). Hence, NIPA uses Q-learning algorithms [35], which has proven effective for solving such problems [7, 42]. Our Q-learning formulation of NIPA naturally addresses the underlying discrete optimization problem by mapping the addition of an edge to a discrete action within the reinforcement learning framework. To effectively cope with the large search space, NIPA adopts hierarchical Q-learning to take advantage of a hierarchical decomposition of the set of candidate actions.. To cope with the non-linearity of the mapping between states and actions, NIPA uses a deep Q network and GNN based encoding of the states into their low-dimensional latent representations. The key contributions of the paper are as follows:

- We study node injection poisoning attacks, a novel family of adversarial attacks on GNNs that are designed to adversely impact the node classification accuracy of GNNs *without manipulating the link structure of the original graph.*

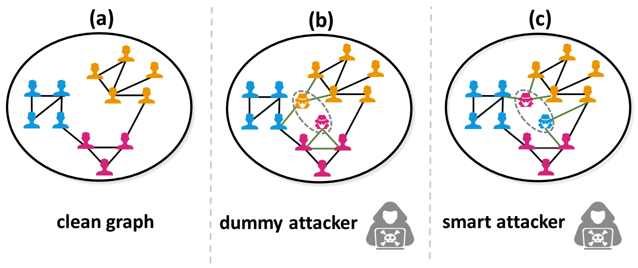

Figure 1: (a) is the toy graph where the color of a node represents its label; (b) shows the node injection poisoning attack performed by a naive attacker; (c) shows the node injection poisoning attack performed by a smart attacker using a smart strategy. The injected nodes are circled with dashed line.

- We propose NIPA, a novel hierarchical Q-learning based framework for launching such node injection poisoning attacks against GNNs. The resulting framework effectively addresses several non-trivial challenges presented by the resulting reinforcement learning problem.
- We present the results of experiments with several real-world graphs data that show that NIPA outperforms the state-of-the-art non-target-specific attacks on GNNs (as measured by the reduction in accuracy of node classification of a GNN trained on the poisoned graph).

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, 24, 28], unsupervised [44], and reinforcement [18, 22, 26] learning. Specifically, recent work has shown that deep neural networks are particularly vulnerable to data poisoning attacks [8, 19, 20, 36]. However, our focus is on such attacks on classifiers trained on graph-structured data.

2.1 Adversarial Attacks on Graph Neural Networks

Recent work has highlighted the vulnerability of graph neural networks to adversarial attacks [12, 43, 50]. As already noted, such attacks can be target-specific (where the goal is to reduce the accuracy of classification of a specific target node) or non-target specific (where the goal is to reduce the overall accuracy of node classification across the graph). Both kinds of attacks can be executed by selectively adding adversarial edges or removing existing edges between the existing nodes in the graph [12, 43, 50].

Nettack [50] perturbs the graph data to poison the graph data used to train graph convolutional neural networks (GCN) [23]. RL-S2V [12] uses reinforcement learning to perform an attack aimed at evading detection during classification. Others focus on poison attacks that exploit the gradient of the loss function of the neural network with respect to the node labels or encoding of the link structure of the input graph [10, 43]; or the use of meta learning to perform a data poisoning attack [51]. Yet others formulate the adversarial attack on a graph as an optimization problem solve it using approximation techniques [38].

All of the preceding adversarial attacks on graphs focus on manipulating links among the existing nodes in the graph being attacked. In real-world settings, such attack strategies are impractical because the nodes in question are not typically under the control of the attacker. We consider a more realistic attack scenario that does not assume that the attacker can manipulate the links among existing nodes. Instead, we consider NIPA, a novel poisoning attack on graph data, that injects fake nodes (e.g., fake accounts in a social networks) into the graph, and uses carefully crafted labels for the fake nodes together with links between them and other (fake as well as genuine) nodes in the graph to poison the graph data.

2.2 Reinforcement Learning in Graph

Reinforcement learning(RL) offers a powerful approach to solving challenging problems in a variety of domains including robot control [33], game playing [29], code retrieval [45], among others.

More recently, reinforcement learning has begun to find applications that involve graph data. For example, NerveNet [41] uses a graph representation of the body of a robot to learn to control policy for the robot using a Graph Neural Network; Graph Convolutional Policy Network (GCPN) [47] uses graph representations of molecular structures to learn to generate molecular structures; Other work [14] learns to predict the products of chemical reactions.

In work that is most closely related to ours, RL-S2V [12] uses reinforcement learning to accomplish target evasion (a form of target-specific graph data poisoning attack) *by manipulating the links among existing nodes*. However, there are several main differences between RL-S2V and NIPA, our proposed model: (1) RL-S2V is a target-specific attack, whereas NIPA is non-target-specific; (2) Whereas RL-S2V learns to attack the targeted nodes in the graph by modifying the link structure of the original graph, NIPA instead establishes adversarial connections and labels for the fake nodes injected into the network; (3) There are major differences in the reward functions used by RL-S2V and NIPA.

3 NODE INJECTION POISONING ATTACKS ON GRAPH DATA

We first introduce the semi-supervised node classification problem before proceeding to formulate the non-target-specific node injection poisoning attacks on graph data aimed at decreasing the node classification accuracy of a predictive model trained on the graph data to perform node classification.

3.1 Semi-Supervised Node Classification

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

In the semi-supervised node classification task, the node classifier $C(G)$ learns the mapping $V \mapsto \tilde{L}$ to label nodes $v_j \in \mathcal{U}$ by exploiting the structure and the attributes of the nodes in the graph G, e.g., v_j and its neighbors. The classifier C is parameterized by θ and hence denote the classifier by C_θ. We use $C_\theta(G)_i$ to denote the label predicted by the classifier for node v_i and $\mathcal{T}_i$ as the ground truth label of v_i. During training, we aim to learn an optimal classifier C with the corresponding parameters θ_L:

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

where $\mathcal{L}$ is an appropriately chosen loss function such as cross entropy. In this paper, we focus on non-targeted graph poisoning attack problem where the attacker $\mathcal{A}$ poisons the the training data used to train the classifier C so as to reduce the overall performance of the resulting classifier on the unlabeled nodes $\mathcal{U}$.

Definition 3.2. (Non-Target-Specific Node Poisoning Attack) Given the attributed graph $G = (V, E, X)$, the labeled node set $\mathcal{T}$, the unlabeled node set $\mathcal{U}$ and an algorithm for learning C from the graph G, the attacker $\mathcal{A}$ aims to modify the graph G within a specified attack budget Δ (which controls the amount of change in the poisoned graph relative to the original graph so as to ensure that the change is virtually un-noticeable) so as to reduce the accuracy of the classifier C learned from the poisoned data on $\mathcal{U}$.

We proceed to propose a novel node injection poisoning attack to inject a set of adversarial nodes $V_\mathcal{A}$ to modify the node set V to realize a non-target-specific node poisoning attack.

Definition 3.3. (Node Injection Poisoning Attack (NIPA). Given a clean graph $G = (V, E, X)$, the attacker $\mathcal{A}$ injects the adversarial node set $V_\mathcal{A}$ with its adversarially established features $X_\mathcal{A}$ and labels $\mathcal{T}_\mathcal{A}$ to augment 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, yielding a poisoned graph $G' = (V', E', X')$ 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 $\mathcal{T}' = \mathcal{T} \cup \mathcal{T}_\mathcal{A}$. The graph classifier is then trained on poisoned graph G'.

With the preceding definitions in place, we are ready to specify an objective function for the non-target-specific node injection poisoning attack:

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

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

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

Here $\mathbb{1}(s)$ is the indicator function such that $\mathbb{1}(s) = 1$ if s is True and 0 otherwise, and $\mathcal{U}_j$ denotes the label of the unlabeled node

v_j. If the attacker has access to the ground truth label for the a node that is unlabeled from the perspective of the end user, then $\mathcal{U}$ corresponds to the 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 first constraint (3) ensures that the classifier is learned from the poisoned graph G'. and the second constraint (4) ensures that the modifications of adversarial edges by the attacker stay within the attack budget Δ. However, if attacker doesn't have the access to the ground truth labels of the unlabeled nodes, the attacker cannot not directly use the objective function in Eq.(2). In this case, we can consider two alternative solutions [51]: The first is to maximize the loss of the classifier on the labeled (training) nodes; the second is to adopt self-learning, i.e. use the predicted labels to compute the loss of the classifier on the unlabeled nodes.

3.2 Graph Convolution Network

In this paper, we use a Graph Convolution Network (GCN) [23], a widely used state-of-the-art model for node classification, as our graph classifier C. The convolutional layer of a GCN considers the topological structure of the input graph and aggregates the attributes of the node and its neighbors followed by the non-linear transformation such as ReLU. The equation for a two-layer GCN is given by:

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

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 adjacency matrix A. $\hat{D}$ denotes the diagonal matrix with diagonal elements $\hat{D}_{ii} = \sum_j \tilde{A}_{ij}$. $W^{(0)}$ and $W^{(1)}$ denote the weights of first and second layer of GCN, respectively. We use ReLU(0, a) = max$(0, a)$ to introduce nonlinearity into GCN. We use cross-entropy as the loss function $\mathcal{L}$.

For the convenience of the reader, the notations used in the paper are summarized in Table 1.

4 NODE INJECTION POISONING ATTACK (NIPA) ALGORITHM

Deep reinforcement learning offers a promising approach for realizing non-target-specific node injection poisoning attacks on graphs. Specifically, deep reinforcement learning offers two key advantages over methods that directly optimize the adjacency matrix with traditional matrix optimization methods: (**i**) Addition of edges between fake nodes and other (fake or genuine) nodes and the assignment of labels to fake nodes are naturally modeled by actions in a reinforcement learning model [35]; (**ii**) The mapping between the underlying graph structure and its low-dimensional representation is typically non-linear [39]. Such non-linearity is easily modeled by a deep neural network. Hence, a deep reinforcement learning framework that combines incorporates non-linear mappings of states to actions via deep neural networks offers an attractive framework for solving the optimization problem specified by Eq.(2) to accomplish non-target-specific node injection poisoning attacks.

The proposed deep reinforcement learning framework for non-target-specific adversarial attacks on graphs is shown in Figure 2. The key idea behind our proposed framework is to use a deep reinforcement learning (DRL) agent to iteratively perform actions

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_θ
$C_\theta(G')$	Prediction of classifier C on G'
L	Label sets
Δ	Attack 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

aimed at poisoning the graph. The actions involve addition of adversarial edges and modification of the labels of the nodes injected into the graph. More specifically, the DRL agent needs to pick a node from injected nodes set $V_{\mathcal{A}}$ and then select another node from poisoned node set V' to add an adversarial edge, and modify the labels of the injected nodes so as to reduce the accuracy of the node classifier C. We now proceed to describe the DRL environment and the reward function designed to accomplish this goal by optimizing the objective function specified by Eq.(2).

4.1 Attack Environment

We model the proposed poisoning attack by a Finite Horizon Markov Decision Process (S, A, P, R, γ) where S denotes the set of states, A the set of actions, P the matrix of state transition probabilities, R is the reward function, and $\gamma < 1$ is a discounting factor that discounts delayed reward relative to immediate reward.

4.1.1 States and Actions. The state s_t contains the intermediate poisoned graph G'_t and labels $\mathcal{T}_{\mathcal{A}_t}$ 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 into a latent space as $e(G'_t)$ by aggregating the graph structure and attributes using a graph neural network. We use $e(\mathcal{T}_{\mathcal{A}_t})$ to encode the adversarial labels $L_{\mathcal{A}_t}$ using the neural network. Since in the injection poisoning environment, the node set V' remains identical the DRL agent performs poisoning actually on the edge set E'_t.

In the poisoning attack environment, the DRL agent is allowed to (1) add the adversarial edges between the injected nodes $V_{\mathcal{A}}$ or between the injected nodes and the original (genuine) nodes; (2) craft the adversarial labels of the injected nodes. However, directly adding one adversarial edge presents $O(|V_{\mathcal{A}}|^2 + |V_{\mathcal{A}}| * |V|)$ possible choices and modifying the adversarial label of an injected node requires $O(|L|)$ space where $|L|$ is the number of possible labels.

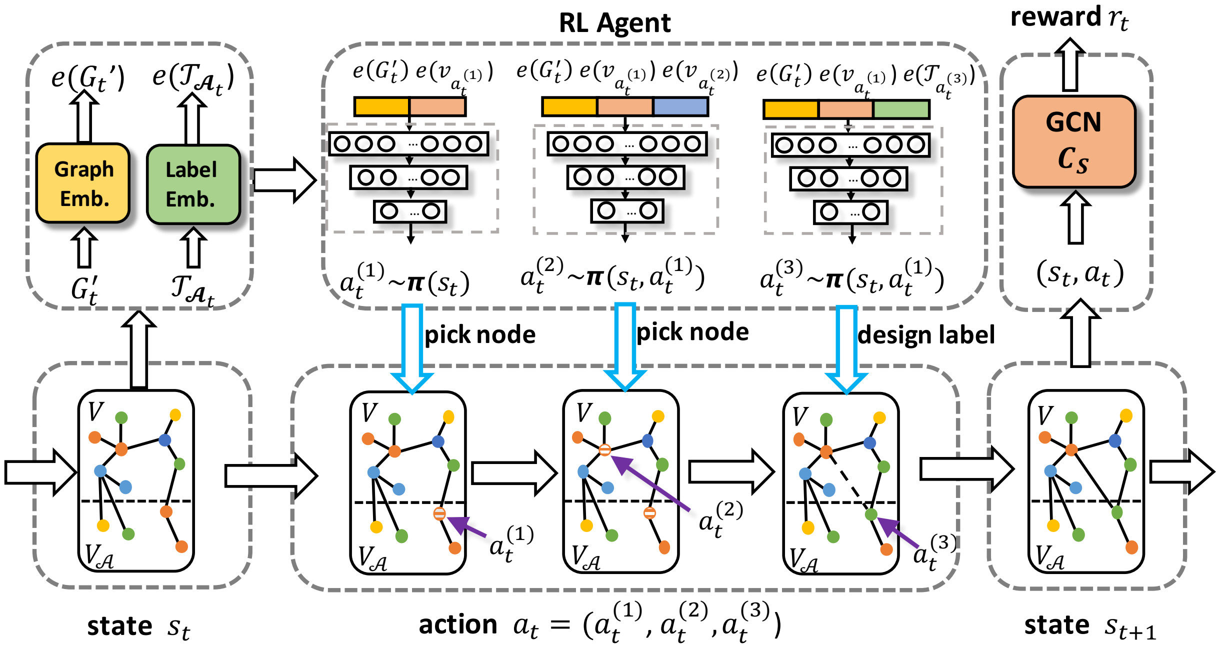
RL Agent
reward r_t
$e(G_t')$ $e(\mathcal{T}_{\mathcal{A}_t})$
Graph Emb.
Label Emb.
G_t' $\mathcal{T}_{\mathcal{A}_t}$
$a_t^{(1)} \sim \pi(s_t)$
$a_t^{(2)} \sim \pi(s_t, a_t^{(1)})$
$a_t^{(3)} \sim \pi(s_t, a_t^{(1)})$
GCN C_S
(s_t, a_t)
pick node
pick node
design label
V
$V_{\mathcal{A}}$
state s_t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

Figure 2: Overview of NIPA

Hence, performing an action that simultaneously adds an adversarial edge and changes the label of a node presents a search space of size $O(|V_{\mathcal{A}}| * |V'| * |L|)$, which is infeasible, especially in the case of large graphs. Thus, after [12], we adopt a hierarchical decomposition of actions to reduce the action space and hence the resulting state space.

As shown in Figure 2, in NIPA, at time t, the DRL agent first performs an action $a_t^{(1)}$ to select an injected node from $V_{\mathcal{A}}$. The agent then picks a second node from the node set V' using action $a_t^{(2)}$. After performing the actions $a_t^{(1)}$ and $a_t^{(2)}$, the agent connects these two selected nodes to establish the adversarial edge which is denoted by a dashed line in the Figure 2. Finally, the agent crafys the label of the selected fake node using action $a_t^{(3)}$. The resulting hierarchical decomposition of the actions $a_t = (a_t^{(1)}, a_t^{(2)}, a_t^{(3)})$, yields a reduction in the size of the action space 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 $(s_0, a_0^{(1)}, a_0^{(2)}, a_0^{(3)}, r_0, s_1, \ldots, s_{T-1}, a_{T-1}^{(1)}, a_{T-1}^{(2)}, a_{T-1}^{(3)}, r_{T-1}, s_T)$.

4.1.2 Policy network. After [12] we use Q-learning to find 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 aims to satisfy the the Bellman optimality equation as follows:

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

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

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

However, because we have to accommodate a hierarchical decomposition of actions, we cannot directly use the policy network in Eq.(6) and Eq.(7). Hence, we adopt a hierarchical Q function for the actions and propose the hierarchical framework which integrates three deep Q networks (see below)

4.1.3 Reward Function. The design of a reward function for DRL for NIPA presents some non-trivial challenges. (1) Because the learning trajectory for NIPA in the attack environment is usually quite long, it is beneficial to introduce intermediate rewards to provide feedback to the DRL agent on how to improve its performance at various states along the trajectory; (2) Unlike in the case of a target-specific data poisoning attack where the success or failure of an attack on the targeted node can be used to derive the reward, in the case of a non-target-specific data poisoning attack, we need to aggregate the node classification results over the entire network to determine the reward based on the objective function Eq. (2). For each state s_t, we firstly define the attack success rate $\mathcal{A}_t$ as:

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

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

Here $\mathcal{T}$ is the training set used to compute the reward as specified by the Eq.(2). It is important to note that the C_{θ_S} is not the graph classifier C which is used to estimate the node classification accuracy from the perspective of the end user. Because the attacker usually does not have the model being used by the end user, it represents a simulated graph classifier constructed by the attacker to drive DRL. However, direct use of the success rate $\mathcal{A}_t$ as the reward could slow down training since the accuracy might not differ significantly between two consecutive states. Hence, we design a guiding binary reward r_t to be 1 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_{θ_S} at time t, and to be -1 otherwise. 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 experiments show that such a guiding reward is effective in our setting.

4.1.4 Terminal. In NIPA, the number of allowed adversarial edges added is constrained by the budget Δ to ensure that the poisoned graph is hard to distinguish from the original graph. Hence, once the DRL agent adds a maximum number of edges ($T = \Delta$), it stops taking further actions. In terminal state s_T, the poisoned graph G' contains T (adversarial) edges in excess of those present in the clean graph G.

4.2 Embedding of States

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. As shown in Figure 2, NIPA represents the state s_t by non-linear embeddings $e(G'_t)$ and $e(\mathcal{T}_{\mathcal{A}_t})$. Specifically, to encode the non-Euclidean structure of the poisoned graph G'_t with vector $e(G'_t)$, the latent embedding $e(v_i)$ of each node v_i in G'_t is first learned using struct2vec [11]. Then the state vector representation $e(G'_t)$ is obtained by aggregating the node embeddings as follows:

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

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

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

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

To avoid notational clutter, we use $e(s)$ to denote the embedding of the state, and $e(v_a)$ and $e(\mathcal{T}_a)$ to denote embeddings of the node selected by action a and label selected by action a respectively.

4.3 Hierarchical Q Network

Given the state s_t and action a_t, the action-value function $Q(s_t, a_t)$ scores the current state and selected actions to guide a Q learning agent. However, because we have hierarchically decomposed each action a into three component actions $\{a^{(1)}, a^{(2)}, a^{(3)}\}$ because of efficiency considerations, it is hard to design a single Q function $Q(s_t, a_t^{(1)}, a_t^{(2)}, a_t^{(3)})$ to optimize the policy.

Hence, we adopt a hierarchical deep Q networks $Q = \{Q^{(1)}, Q^{(2)}, Q^{(3)}\}$ which integrate three distinct Q networks to model the Q values for each of the three 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. Based on the state representation $e(s_t)$, the first deep Q network (DQN) $Q^{(1)}$ guides the policy to choose a node from the 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', thus introducing an adversarial edge; The third DQN $Q^{(3)}$ learns the policy for crafting the label of the fake node injected by the first DQN.

The DRL agent first selects one node from the injected node set $V_{\mathcal{A}}$ and calculates the Q value of the state-action pair based on the action-value function $Q^{(1)}$ as follows:

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

where $\theta^{(1)} = \{W_1^{(1)}, W_2^{(1)}\}$ denotes 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 for selecting the action $a_t^{(1)}$ based on optimal action-value function $Q^{(1)*}$ in Eq.(13) is given by:

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

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

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

where $\theta^{(2)} = \{W_1^{(2)}, W_2^{(2)}\}$ denote the trainable weights. The action value function $Q^{(2)}$ scores the candidate nodes for establishing an edge based on the state s_t, and the selected action $a_t^{(1)}$. The greedy policy for the second action $a_t^{(2)}$ with the optimal $Q^{(2)*}$ in Eq.(15) is given by:

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

Note that the DRL agent only modifies the label of the selected injected fake node $a_t^{(1)}$. Hence, 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 candidate labels for an injected fake node is given by:

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

In Eq.(17), $\theta^{(3)} = \{W_1^{(3)}, W_2^{(3)}\}$ denote the trainable weights in $Q^{(3)}$. The action value function $Q^{(3)}$ models the score of each candidate

[2]Note that the embedding methods that can be used to encode states and the nodes and edges are not limited to those proposed here

label for the injected node $a_t^{(1)}$. The greedy policy for the corresponding action is given by:

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

Thus, the proposed hierarchical deep Q networks $Q = \{Q^{(1)}, Q^{(2)}, Q^{(3)}\}$ in Eq.(13), Eq.(15) and Eq.(17), NIPA integrate hierarchical action-value functions to model the Q values for 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 memory buffer $\mathcal{M}$ [29]. The key idea behind experience replay is to randomizes the order of data used by Q learning, so as to remove the correlations in the observation sequence. We simulate action selection and store the resulting data in a 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 given by:

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

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

In the proposed DRL, we use two layer neural networks to encompass the trainable parameters θ in action-value functions $Q = \{Q^{(1)}, Q^{(2)}, Q^{(3)}\}$ and structure2vec [3].

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 AND RESULTS

We proceed to experiments that compare NIPA with several baseline methods for non-target-specific graph data poisoning attacks. poisoning attack methods using several benchmark data sets. Our experiments are designed to answer the following research questions:

- (**RQ1**) Can the NIPA algorithm effectively perform a non-target-specific node injection poisoning attack on the graph data and hence a GCN model trained on the graph data for node classification?
- (**RQ2**) How closely does the poisoned graph resemble the original graph (based on the the key statistics of the two graphs)?
- (**RQ3**) How does the effectiveness of NIPA vary as a function of factors such as sparseness of the graph being attacked, the average degree of fake nodes, etc.?

5.1 Experimental Setup

5.1.1 Datasets. We report results of experiments with three widely used benchmark data sets for node classification, which include CORA-ML [5, 27], CITESEER [17] and DBLP [31]. After [51], we consider only the largest connected component (LCC) from each graph data set. The statistics of the data sets computed from the LCC are summarized in Table 2. For each data set, we randomly split the nodes into two disjoint sets: (20%) of the labeled nodes are used for training the node classifier and (80%) of the unlabeled nodes nodes as test set to evaluate the model. The labeled nodes are further split into training and validation sets of equal size. We report results averaged over 5 different random splits.

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. Recall that the attack executed by NIPA is quite distinct from those executed by Nettack [50] and RL-S2v [12], because the latter, unlike the former, can manipulate links between existing nodes in the graph. Consequently it does not make sense to use Nettack and RL-S2v as baselines for comparison with NIPA. Because NIPA is a novel form of attack, there are few baseline methods that we can compare it with. Hence, we consider the

[3]The proposed framework admits more complex deep neural network architectures

following four baselines, two of which are based on classical graph attack models, one is a variant of fast gradient attack, and one is 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 proportional to the average degree of the clean graph $G(V, E)$. This ensures that the density of the injected graph G_A is similar to that of the clean graph. The attacker then randomly adds adversarial edges that link the nodes in the injected graph G_A with those in the clean graph G until the attack budget Δ is used up.
- Preferential attack [2]: The attacker $\mathcal{A}$ iteratively adds the adversarial edges based on a preferential attachment mechanism. The probability of connecting the injected node $v_i \in V_{\mathcal{A}}$ to any other node $v_j \in |V \cup V_{\mathcal{A}}|$ is proportional to the degree of the node. As before, the number of adversarial edges is constrained by the budget Δ.
- A variant of the Gradient Attack (FGA) [10]: The attacker $\mathcal{A}$ attacks the graph by introducing or removing adversarial edges guided so as to maximize the node mis-classification rate as a function of addition or deletion of adversarial edges until the attack budget Δ is used up. Because the original Fast Gradient Attack (FGA) [10] is not directly applicable in the node injection poisoning setting where the injected nodes are isolated from the rest of the graph and can be easily filtered out as fake by the node classifier. Hence, we apply FGA to the graph poisoned by preferential attack. The resulting attack uses the gradient $\nabla_{ij} L_{GCN}$ with $v_i \in V_{\mathcal{A}}$ and $v_j \in V'$, to add or remove the adversarial edges between (v_i, v_j) to maximize the gradient of the GCN loss $\nabla_{ij} L_{GCN}$. The number of adversarial edges added or deleted is constrained by budget Δ. The attacker is allowed to perform 20Δ modifications in total as suggested by [10].
- NIPA-w/o: This attack is a variant of NIPA where we do not optimize the objective function with respect to the labels of fake nodes, i.e., it randomly labels to the fake nodes. In all other respects, NIPA-w/o behaves like NIPA.

5.2 Performance of NIPA Compared with the Baseline Methods

To answer **RQ1**, we assess the decrease in accuracy of node classification on the poisoned graph as compared with that on the clean graph. The larger the decrease in node classification performance on the poisoned graph relative to the clean graph, the more effective the attack.

5.2.1 Node Classification on Clean Graph. After [50] we adopt the transductive learning setting. The parameters of GCN are trained according to Eq. (1). We report the node classification accuracy of the GCN trained on the clean graph averaged over the five independent runs (see above) 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 the Poisoned Graph. The attack budget Δ controls the number of added adversarial edges impacts the effectiveness of the attack. On the one hand, if the budget is too small eg., $\Delta < |V_{\mathcal{A}}|$, then at least $|V_{\mathcal{A}}| - \Delta$ injected nodes are isolated. Such isolated nodes have no effect on the node classification accuracy since they can be easily detected as fake by the classifier. On the other hand, if the attack budget is too large, the density of the poisoned graph will be different from that of the clean graph, thus putting the injected nodes at risk of being detected by methods designed to defend the network against attacks. Hence, to simulate real-world attack scenarios where the attacker has every incentive to ensure that the attack goes undetected, we ensure that the density of the poisoned graph is similar to that of the clean graph by setting $\Delta = r|V|deg(V)$ where r is the fraction of the injected nodes relative to the number of nodes in the clean graph and $deg(V)$ is the average degree of the clean graph G. Hence, the number of injected nodes is $|V_{\mathcal{A}}| = r|V|$. We also evaluate the effectiveness of the attack as a function of the degree of the injected nodes (See Section 5.4.1). For a comprehensive comparison of the different attacks, we experiment with different choices of $r \in \{0.01, 0.02, 0.05, 0.10\}$. We do not consider values of $r > 0.10$ because higher values of r make it difficult for the attack to be unnoticed. Similar considerations dictate that the features of the injected fake nodes be similar to those of the genuine nodes in the clean graph. For each injected node, we set its feature values to be those obtained by perturbing the mean $\bar{X}$ of the features of the nodes in the clean graph with zero mean Gaussian noise $\mathcal{N}(0, 1)$ [4]. Because the baseline methods cannot modify the adversarial labels of the injected nodes, we randomly generate the adversarial labels in the case of all baseline methods, including NIPA-w/o. In both NIPA and NIPA-w/o, we set the discount factor $\gamma = 0.9$ and the injected nodes $V_{\mathcal{A}}$. Needless to day, in all cases, the graph is poisoned only prior to training the node classifier.

The mean and the standard deviation of the accuracy of node classification (obtained from 5 independent runs - see above) are reported in Table 4. The results included in Table 3 and 4, show that:

- As expected, regardless of the attack method chosen, the effectiveness of the attack improves with higher values of r which translate to greater numbers of injected nodes.
- NIPA significantly outperforms all baseline methods.
- The FGA variant marginally outperforms the Random Attack and the Preferential Attack. This is explained by the fact that unlike the FGA variant which adaptively inserts fake nodes and links to maximize the decrease in node classification accuracy, the Random and the Preferential attack simply insert fake nodes according to a predefined strategy.
- NIPA significantly outperforms the FGA variant. We attribute the greater effectiveness of NIPA to its effective use of hierarchical deep reinforcement learning to optimize the effectiveness of the attack.

[4]It is possible to further optimize the node features to maximize the effectiveness of the attack while minimizing the likelihood of being detected

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

- NIPA out performances NIPA-w/o, which shows the benefits of optimizing the labels of the injected nodes labels in node injection attacks.

5.3 Key Statistics of the Poisoned Graphs

To answer **RQ2**, we analyze several key statistics of the poisoned graphs. A desired property of the poisoning attack is that the poisoned graph is statistically similar to the clean graph. After [6], we report the graph statistics of the poisoned graphs for all three benchmark data sets in Table 5. The statistics considered include the Gini coefficient, characteristic path length, distribution entropy, power law exponent and the triangle counts. The detailed equations and descriptions are available in [6]. The graph statistics included in Table 5 show that:

- For smaller values of r, the graphs poisoned by NIPA generally are statistically very similar to the corresponding clean graph.
- For larger values of r, the graphs poisoned by NIPA become more distinguishable from the corresponding clean graph.
- Larger values of r lead to an increase in the number of triangles in the graph poisoned by NIPA. Thus, NIPA results in not only fake nodes getting connected to other nodes, but also in the introduced edges leading to the formation of triangles, causing the edges to affect additional nodes in the graph.

5.4 Effectiveness of NIPA Under Different Scenarios

In this subsection, we report the results of experiments designed to answer **RQ3**, i.e., how does the performance of NIPA vary as as function of different parameters of NIPA and the graph being attacked.

5.4.1 *Average Degrees of Injected Nodes.* Recall that $\Delta = r|V|\deg(v_{\mathcal{A}})$ which plays a role in the effectiveness of NIPA. Hence, we investigate impact of the average degree of injected nodes on the effectiveness of NIPA. Specifically, we experiment with different values of

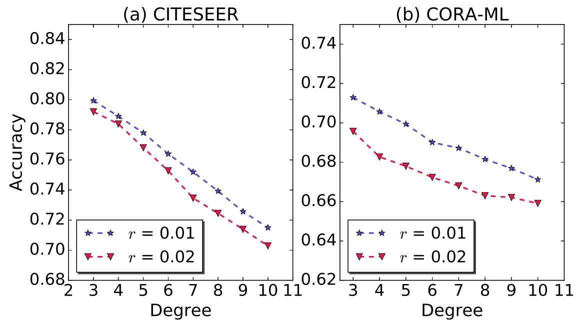

Figure 3: Node classification performance on a graph poisoned by NIPA, in the case of (a) CITESEER and (b) CORA-ML, as a function of average node degree of injected nodes

$\deg(v_{\mathcal{A}}) \in \{3, \ldots 10\}$. We do not consider values of $\deg(v_{\mathcal{A}}) > 10$ since injecting 'celebrity' or 'hub' nodes into the network is generally not an effective graph poisoning attack strategy.

The performance of NIPA 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. We observe that the node classification accuracy on the graph poisoned by NIPA decreases or the effectiveness of NIPA increases as the average degree of the injected nodes increases. This observation is consistent with the fact that the greater the number of links a fake node can have, the greater its ability to poison the graph.

5.4.2 *Sparsity of the Clean Graph.* We examine the effect of sparsity of the graph on the effectiveness of NIPA. In this experiment, we ensure that the average degree of injected nodes to be same as the the average degree of genuine nodes. To simulate a sparse network, we randomly remove $S_p = \{0, 10\%, \ldots, 90\%\}$ edges from the clean graph and execute NIPA on the resulting sparsified 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. We observe that as the graph more and more sparse, NIPA becomes

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

more and more effective in attacking the graph. This is because as the graph becomes sparser, each node in the clean graph has fewer neighbors, which makes the it easier for fake nodes to impact the labels of unlabeled nodes.

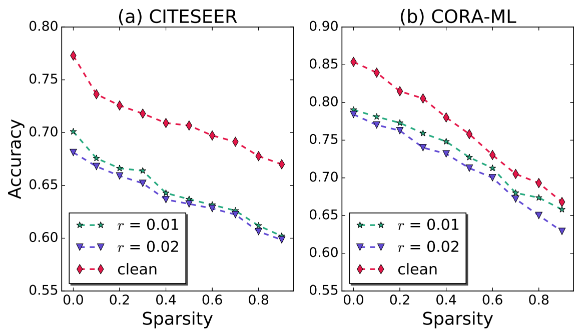

Figure 4: Node classification performance on (a) CITESEER and (b) CORA-ML with varying graph sparsity

6 SUMMARY AND DISCUSSION

We introduce a novel family of non-target-specific node injection poisoning attack on graphs. We propose NIPA, a deep hierarchical reinforcement learning based method to execute such attacks. 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. NIPA injects fake nodes into the graph, and sequentially adds adversarial edges and adversarially chooses labels for the injected fake nodes. NIPA uses a novel hierarchy deep Q learning networks to efficiently reduce the state-action combinations to be explored. It uses graph neural networks to learn compact graph embeddings to encode states. We report the results of poisoning the graph data used to train graph convolutional network for node classification which show the effectiveness of NIPA in poisoning the graph in a non-target-specific manner. Our results show that the graph poisoned by NIPA statistically very similar to the original clean graph, making it easy for the attack to evade detection. Our experiments further show, as expected, that the larger the attack budget, the more effective the attack; and sparser the graph being attacked, the more effective is the attack.

There are several interesting directions that need further investigation. In this paper, 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. In this paper, 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. 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.

ACKNOWLEDGEMENTS

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. The content is solely the responsibility of the authors and does not necessarily represent the official views of the sponsors.

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] Bo Li, Yining Wang, Aarti Singh, and Yevgeniy Vorobeychik. 2016. Data poisoning attacks on factorization-based collaborative filtering. In *Advances in neural information processing systems*. 1885–1893.

[25] 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.

[26] 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.

[27] 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.

[28] 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*.

[29] 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.

[30] 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.

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

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

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

[34] 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).

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

[36] 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).

[37] Xianfeng Tang, Yandong Li, Yiwei Sun, Huaxiu Yao, Prasenjit Mitra, and Suhang Wang. 2019. Robust graph neural network against poisoning attacks via transfer learning. *arXiv preprint arXiv:1908.07558* (2019).

[38] 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).

[39] 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.

[40] 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.

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

[42] 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.

[43] 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*.

[44] Huang Xiao, Battista Biggio, Gavin Brown, Giorgio Fumera, Claudia Eckert, and Fabio Roli. 2015. Is feature selection secure against training data poisoning?. In *International Conference on Machine Learning*. 1689–1698.

[45] 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.

[46] 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.

[47] 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.

[48] 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.

[49] 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.

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

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