Let Graph Be the Go Board: Gradient-Free Node Injection Attack for Graph Neural Networks via Reinforcement Learning

Mingxuan Ju[1], **Yujie Fan**[2], **Chuxu Zhang**[3], **Yanfang Ye**[1]
[1] University of Notre Dame, Notre Dame, IN 46556
[2] Case Western Reserve University, Cleveland, OH 44106
[3] Brandeis University, Waltham, MA 02453
[1]{mju2, yye7}@nd.edu; [2]yxf370@case.edu; [3]chuxuzhang@brandeis.edu

Abstract

Graph Neural Networks (GNNs) have drawn significant attentions over the years and been broadly applied to essential applications requiring solid robustness or vigorous security standards, such as product recommendation and user behavior modeling. Under these scenarios, exploiting GNN's vulnerabilities and further downgrading its performance become extremely incentive for adversaries. Previous attackers mainly focus on structural perturbations or node injections to the existing graphs, guided by gradients from the surrogate models. Although they deliver promising results, several limitations still exist. For the structural perturbation attack, to launch a proposed attack, adversaries need to manipulate the existing graph topology, which is impractical in most circumstances. Whereas for the node injection attack, though being more practical, current approaches require training surrogate models to simulate a white-box setting, which results in significant performance downgrade when the surrogate architecture diverges from the actual victim model. To bridge these gaps, in this paper, we study the problem of black-box node injection attack, without training a potentially misleading surrogate model. Specifically, we model the node injection attack as a Markov decision process and propose **G**radient-free **G**raph **A**dvantage **A**ctor **C**ritic, namely **G^{2}A2C**, a reinforcement learning framework in the fashion of advantage actor critic. By directly querying the victim model, G^{2}A2C learns to inject highly malicious nodes with extremely limited attacking budgets, while maintaining a similar node feature distribution. Through our comprehensive experiments over eight acknowledged benchmark datasets with different characteristics, we demonstrate the superior performance of our proposed G^{2}A2C over the existing state-of-the-art attackers. Source code is publicly available at: https://github.com/jumxglhf/G2A2C.

Introduction

Graph neural networks (GNNs), a class of deep learning methods designed to perform inference on graph data, have achieved outstanding performance in various real-world applications, such as recommendation system (Ying et al. 2018), user behavior modeling (Pal et al. 2020) and drug discovery (Jiang et al. 2021). The success of GNNs relies on their powerful capability of integrating the graph structure and node features simultaneously for node representation learning. Specifically, the majority of popular GNNs (Kipf and Welling 2016; Veličković et al. 2017) follow a neural message-passing scheme to learn node embeddings via recursively aggregating and propagating neighbor information. Along with their great success, the robustness of GNNs has also attracted increasing attentions in recent years, and it has been proved that such a message-passing scheme is vulnerable to adversarial attacks (Zügner, Akbarnejad, and Günnemann 2018; Chen et al. 2022).

Existing research efforts on graph adversarial attack mainly concentrate on graph structure perturbations via modifying edges (Dai et al. 2018; Wang and Gong 2019; Zügner, Akbarnejad, and Günnemann 2018). Despite their promising performance, these attack strategies have narrow applications since the adversaries are required to manipulate the existing graph topology, which is impractical under most circumstances. Besides graph structural perturbations, another trend of research focuses on the node injection attacks (Tao et al. 2021; Zou et al. 2021; Wang et al. 2020). They explore a more practical setting where attacks are launched by injecting new nodes into the existing graphs, and hence the authorities of modifying the existing graph structures are unnecessary. Considering spam detection in the social networks as an example, where adversaries aim at tricking the victim model into misclassifying the spam accounts (i.e., target nodes). In many circumstances, they do not have permission to add or remove the friendships already formed among the existing users (i.e., modifying connections between existing nodes). However, adversaries can easily create accounts with new profiles and establish new links with the existing users to fulfill the attack purposes (i.e., injecting new malicious nodes to deceive the victim model). Apparently, the node injection attacks are more feasible compared with the attacks via graph perturbations.

Nevertheless, node injection attacks are challenging, and the adversaries should consider: (i) *how to generate imperceptible yet malicious features for the injected nodes?* and (ii) *how to establish links between an injected node and the existing nodes?* Current related works (Tao et al. 2021; Zou et al. 2021; Wang et al. 2020; Sun et al. 2019) fulfill these purposes according to the gradient from the victim model in the white-box setting or the simulated gradient from the surrogate model in the black-box setting. Though promis-

ing, white-box approaches are usually not practical, and the performance of black-box ones can be deteriorated when the surrogate architecture and the actual victim model diverge.

In this work, we consider the most challenging and practical scenario, i.e., black-box evasion attack through the node injection, where only adjacency matrix, node attributes and model queries are available. To tackle these aforementioned challenges, we propose **G**radient-free **G**raph **A**dvantage **A**ctor **C**ritic, namely **G^{2}A2C**, a reinforcement learning framework in the fashion of advantage actor critic. Different from the existing counterparts, G^{2}A2C does not require the adversaries to train a surrogates model since the vulnerabilities of the victim model are learned according to the queries from the victim model instead of the surrogate gradient. Thus, G^{2}A2C makes no prior assumption about the victim model, which eliminates possible performance downgrade introduced by the divergence between the assumption and the actual victim model. Specifically, we formulate the node injection attack as a Markov Decision Process (MDP), where the attack is decoupled into node generation and edge wiring. During the node generation phase, imperceptible yet malicious features are attributed to the adversarial node. To guarantee the imperceptibility, besides the regularization on the similarities to the real nodes from the original graph, we design separate strategies to tackle both discrete and continuous feature spaces, so that G^{2}A2C can inject nodes according to the real feature space. And during the edge wiring phase, edges between the injected node and the remaining graph are wired according to a learnable conditional probability distribution. These two steps are gradient-free because they are guided by the rewards calculated from the model feedback instead of the surrogate gradients. The key contributions of this paper are summarized as follows:

- This is the first work that studies black-box node injection attack for GNNs without using the surrogate gradient, eliminating the performance downgrade entailed by the inaccurate approximation of the victim model.
- We carefully formulate the black-box node injection attack as an MDP and propose G^{2}A2C to launch effective yet imperceptible attacks against GNNs trained on graphs with either discrete or continuous node features.
- With comprehensive experiments over eight acknowledged benchmark datasets, we demonstrate G^{2}A2C's superior attack effectiveness with different attack budgets by comparison with the state-of-the-art attack models.

Preliminary

Let $G = (V, E)$ denote a graph, where V is the set of $|V| = N$ nodes and $E \subseteq V \times V$ is the set of $|E|$ edges between nodes. Adjacency matrix is denoted as $\mathbf{A} \subseteq \{0,1\}^{N\times N}$, where a_{ij} at i-th row and j-th column equals to 1 if there exists an edge between nodes v_i and v_j, and 0 otherwise. We further denote the node feature matrix as $\mathbf{X} \in \mathbb{R}^{N\times F}$ where node v_i is associated with a feature vector $\mathbf{x}_i \in \mathbb{R}^F$ of dimension F. $\mathbf{Y} \subseteq \{0,1\}^{N\times C}$ denotes the label matrix of a graph, where C is the number of total classes. For M labeled nodes ($0 < M \ll N$) with label $\mathbf{Y}^L$ and $N-M$ unlabeled nodes with missing label $\mathbf{Y}^U$, the objective of GNNs for node classification is to predict $\mathbf{Y}^U$ given $\mathbf{Y}^L$, $\mathbf{A}$ and $\mathbf{X}$.

Graph Neural Network

GNNs generalize neural networks into graph-structured data (Kipf and Welling 2016; Veličković et al. 2017; Klicpera, Bojchevski, and Günnemann 2019; Ju et al. 2022). The key operation is graph convolution where information is routed between nodes with some pre-defined deterministic rules (e.g., adjacency matrices, Laplacian matrices, and attention). For example, the graph convolution layer (GCL) of GCN (Kipf and Welling 2016) is formulated as: $\mathbf{H}^{(l+1)} = f^{(l)}_{GCL}(\hat{\mathbf{A}}, \mathbf{H}^{(l)}, \mathbf{W}^{(l)}) = \sigma(\hat{\mathbf{A}}\mathbf{H}^{(l)}\mathbf{W}^{(l)})$, where $\hat{\mathbf{A}}$ denotes the normalized adjacency matrix with self-loop, $\sigma(.)$ denotes the non-linearity function, and $\mathbf{W}^{(l)}$ and $\mathbf{H}^{(l)}$ are the learnable parameters and node representations at l^{th} layer respectively. Normally, at K-th layer, with the last dimension of $\mathbf{W}^{(K)}$ and $\sigma(.)$ set to C and softmax respectively, the loss for node v_i is calculated as:

$$\mathcal{L}(v_i, G, \mathbf{y}_i) = CE\big(f^{(K)}_{GCL}(\hat{\mathbf{A}}, \mathbf{H}^{(K)}, \mathbf{W}^{(K)})[i], \mathbf{y}_i\big), \quad (1)$$

where $[\cdot]$ is the indexing operation and $CE(\cdot,\cdot)$ refers to the cross entropy loss function.

Graph Adversarial Attack

For a trained GNN model $f(\cdot,\cdot) : V \times G \to \mathbf{Y}$, the attacker $g(\cdot,\cdot) : G \times f \to G$ is asked to modify the graph $G = (V, E)$ into $G' = (V', E')$ such that:

$$\begin{aligned} \max_{G'} \quad & \mathbb{I}(f(V^U, G') \neq \mathbf{Y}^U) \\ s.t. \quad & G' = g(G, f) \ \text{ and } \ \mathcal{I}(G, G') = 1. \end{aligned} \quad (2)$$

Here V^U can be the testing set or nodes of interest, $\mathbb{I}(\cdot)$ is an indicator function that returns the number of true conditions, and $\mathcal{I}(\cdot,\cdot) : G \times G \to \{0,1\}$ is an indicator function, which returns one if two graphs are equivalent under the classification semantics. There exist two approaches to fulfill the attack purposes. The first is edge modification in G, also known as structural perturbation, which changes the entries in $\mathbf{A}$. Whereas the second approach tampers the nodes via adding, modifying, or deleting nodes in G, resulting in not only entry-level but also dimensional changes to both $\mathbf{A}$ and $\mathbf{X}$. In this work, we focus on the black-box node injection evasion attack, a special case of the second approach, where three attack budgets need to be considered: the number of adversarial nodes injected per attacking one node, denoted as β_n, the degree of each injected node β_e, and lastly the feature distribution shift β_f. Hence, $\mathcal{I}(\cdot,\cdot)$ is defined as

$$\begin{aligned} \mathcal{I}(G, G') = \mathbb{I}(|V'| - |V| \leq \beta_n) \cdot \prod_{i=N}^{N+\beta_n} \mathbb{I}(\|\mathbf{a}'_i\| \leq \beta_e) \\ \cdot \prod_{i=N}^{N+\beta_n} \mathbb{I}(\langle \mathbf{x}'_i, \mathbf{X} \rangle \leq \beta_f), \end{aligned} \quad (3)$$

where $\langle \cdot,\cdot \rangle$ refers to the metric measuring the similarities between the generated feature and the original features (e.g., Kullback–Leibler divergence for continuous features, and norm difference for discrete features).

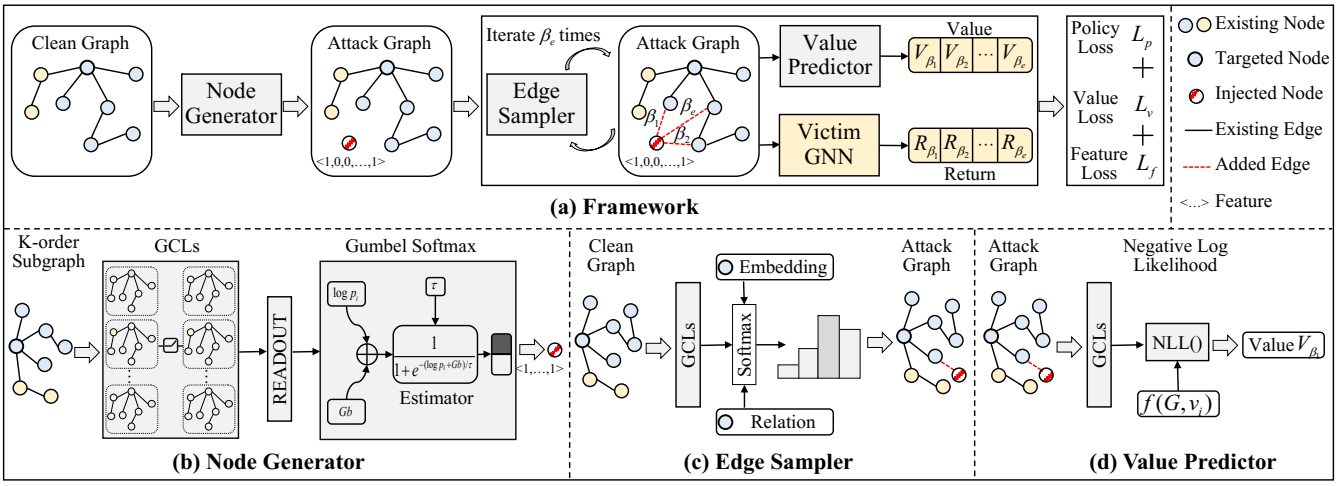

Figure 1: System overview of G^{2}A2C.

Methodology

Given a clean graph $G = (V, E)$, the attacker g injects a set of adversarial nodes $V^{\mathcal{A}}$ with generated features $\mathbf{X}^{\mathcal{A}}$ into the clean node set V. After injecting $V^{\mathcal{A}}$, attacker g creates adversarial edges $E^{\mathcal{A}} \subseteq V^{\mathcal{A}} \times V \cup V^{\mathcal{A}} \times V^{\mathcal{A}}$ to evade the detection of GNN f for nodes V^U. $G' = (V', E')$ is the attacked graph in which $V' = V \cup V^{\mathcal{A}}$, $E' = E \cup E^{\mathcal{A}}$, and $\mathbf{X}' = \mathbf{X} \oplus \mathbf{X}^{\mathcal{A}}$, where $\oplus$ is the vertical concatenation.

Injecting node involves generating discrete graph data, such as adjacency matrices or feature matrices, that gradient-based approaches handle poorly in many circumstances. This phenomenon could be further aggravated by the black-box setting where gradient information from the surrogate model might not be accurate. Moreover, generating node and assigning edges are naturally sequential and reinforcement learning fits for such Markov Decision Process (MDP). Hence, to perform the optimization task in Eq. (2), we propose to explore deep reinforcement learning. Specifically, we utilize an on-policy A2C reinforcement learning framework, adapted from (Mnih et al. 2016), instead of the off-policy algorithms such as deep Q-learning. Since A2C circumvents the need to calculate the expected value for every possible action, which is intractable.

The overview of G^{2}A2C is shown in Figure 1. Given a graph G and a target node v_i, the node generator g_n creates the adversarial node according to v_i's sub-graph. Then edge sampler g_e is forwarded for β_e consecutive times to connect the injected node to the existing graph. Previous two processes iterate until the label of v_i has been successfully changed or the attack budget is depleted. A detailed definition of our proposed MDP is defined as follows:

State. $s_t \in S$ contains the intermediate modified graph $G'_t = (V'_t, E'_t)$ as well as the generated features $\mathbf{X}^{\mathcal{A}}_t$ at the timestamp t. To efficiently interpret s_t, edge sampler g_e and node generator g_n attend to the K-hop sub-graph $G'_t(v_i)$ entailed by the target node v_i, where K is a hyper-parameter for the number of stacked GCLs. We restrain our scope on the neighbors within K hops since normally the victim GNN f has a shallow receptive field. However, without loss of generality, g_e can be extended to the full graph G'_t for the optimal performance.

Action. Node injection attack can be decoupled into two components: creating the injected node and wiring it to the existing graph. We model this process as an MDP where G^{2}A2C starts with the node generation and then wires the generated node to the existing graph for β_e consecutive times. The MDP terminates if the attacker g successfully evades the detection from f or the attack budget is depleted. Formally, at time t, the node generation action is denoted as $a_t^{(n)}$, and the node wiring action is denoted as $a_t^{(e)}$. The trajectory of our proposed MDP is $(s_0, a_0^{(n)}, s_1, a_1^{(e)}, r_1, s_2, a_2^{(e)}, r_2, s_2, \ldots, a_t^{(n)}, s_t, \ldots, s_T)$, where s_T refers to the terminal state, and r_t is the reward for the action a_t.

Reward. In our curated setup, generating an isolated node does not entail a reward value, as an isolated node brings no perturbation to f; instead, the impact is reflected later when links are wired to the existing graph. Hence, reward values are only assigned to the edge wiring actions. During the intermediate phase of attacking node v_i at timestamp t, the reward for edge wiring action $a_t^{(e)}$ is calculated as:

$$r(v_i, a_t^{(e)}, G'_t) = \mathcal{L}\big(v_i, G'_{t+1}, f(v_i, G)\big) - \mathcal{L}\big(v_i, G'_t, f(v_i, G)\big) \text{ if } s_{t+1} \neq s_T, \tag{4}$$

where G'_{t+1} is the resulted graph after applying $a_t^{(e)}$ to G'_t. The reward function measures the difference between the classification losses before and after the edge wiring, which encourages G^{2}A2C to imperil the correct decision of the victim model. Besides, to further motivate our model to actively evade the detection, we give extra rewards if the prediction of v_i is flipped at the end of one attacking episode (i.e., $\mathbb{I}\big(f(G'_T, v_i) \neq f(v_i, G)\big)$).

Node Injection Attack via Actor Critic

Adversarial Node Generator To deceive f into misclassifying the target node v_i, given its K-order sub-graph $G'_t(v_i)$, the node generator g_n aims to create an adversarial node v_a with an imperceptible yet malicious feature vector $\mathbf{x}_a$. Specifically, the generated $\mathbf{x}_a$ should follow the same characteristic conventions as $\mathbf{X}$. We should not expect a continuous $\mathbf{x}_a$ when all other feature vectors are discrete, and vice versa. Moreover, $\mathbf{x}_a$ should not diverge too much from nodes features in $G'_t(v_i)$, as restrained by the distribution shift budget β_f. To tackle the aforementioned challenges, g_n is equipped with K-stacked GCLs, conducts message propagation on $G'_t(v_i)$, summarizes $G'_t(v_i)$ by a readout function (Xu et al. 2018), and with Gumbel-Softmax (Jang, Gu, and Poole 2016), generates $\mathbf{x}_a$ tailoring the vulnerability of v_i as well as the imperceptibility, as shown in Figure 1 (b). Formally, the K-th convolution layer of g_n can be described as:

$$\mathbf{H}_n^{(K+1)} = f_{GCL}^{(n,K)}\big(\hat{\mathbf{A}}(v_i), \mathbf{H}_n^{(K)}, \mathbf{W}_n^{(K)}\big), \tag{5}$$

where $\mathbf{H}_n^{(0)} = \mathbf{X}'_t$, $\hat{\mathbf{A}}(v_i)$ refers the normalized adjacency matrix of $G'_t(v_i)$ and $\mathbf{W}_n^{(K)}$ is the parameter matrix of g_n's K-th convolution layer. To consider the holistic representation of $G'_t(v_i)$ and the unique characteristics tailored by v_i, we formulate the feature distribution $\mathbf{z}_n$ as:

$$\mathbf{z}_n = \sigma\Big(\big(\mathrm{READOUT}(\mathbf{H}_n^{(K+1)})||\mathbf{H}_n^{(K+1)}(v_i)\big)\cdot\mathbf{W}_n^f\Big),$$

where $\mathrm{READOUT}(\cdot)$ refers to the graph pooling function such as column-wise summation (Xu et al. 2018), $\mathbf{H}_n^{(K+1)}(v_i)$ denotes v_i's node embedding after the propagation, and $\mathbf{W}_n^f \in \mathbb{R}^{2d\times F}$ is the learnable parameter matrix that combines the target node's characteristics with the local neighborhood information.

To inject nodes in the discrete feature space, we directly utilize $\mathbf{z}_n$ as the logits of a relaxed Bernoulli probability distribution, a binary special case of the Gumbel-Softmax reparameterization trick which is soft and differentiable (Jang, Gu, and Poole 2016). Utilizing the relaxed sample, we apply a straight-through gradient estimator (Bengio, Léonard, and Courville 2013) that rounds the relaxed sample in the forward phase. In the backward propagation, actual gradients are directly passed to the relaxed samples instead of the previously rounded values, making g_n trainable. Formally, the discrete version of $\mathbf{x}_a$ is generated by:

$$\mathbf{x}_a[i] = \left\lfloor \frac{1}{1+e^{-(\log \mathbf{z}_n[i]+Gb)/\tau}} + \frac{1}{2} \right\rfloor, \tag{6}$$

where $[\cdot]$ is the indexing operation, $Gb \sim Gumbel(0,1)$ is a Gumbel random variable and τ is the temperature for the Gumbel-Softmax distribution. To maintain the imperceptibility of $\mathbf{x}_a$, we propose a feature loss function for the discrete feature space:

$$\mathcal{L}_f(\mathbf{x}_a) = \big(\|\mathbf{x}_a\|/\|\mathbf{X}\| - \beta_f\big)^2. \tag{7}$$

Whereas to inject nodes in the continuous feature space, we transform $\mathbf{z}_n$ into μ_n and σ_n by two learnable weight matrices $\mathbf{W}_\sigma$ and $\mathbf{W}_\mu \in \mathbb{R}^{d\times d}$, and utilize them as parameters of a normal distribution to generate the feature, formulated as:

$$\begin{gathered}\mathbf{x}_a \sim \mathcal{N}(\mu_n, \sigma_n^2) \\ \text{where} \quad \mu_n = \mathbf{z}_n\cdot\mathbf{W}_\mu \text{ and } \sigma_n = \mathbf{z}_n\cdot\mathbf{W}_\sigma.\end{gathered} \tag{8}$$

For nodes in the continuous feature space, we enforce the imperceptibility by minimizing the KL-divergence between the feature generation distribution and the real node feature distribution (i.e., represented by the mean value of the real features $\mu_{\mathbf{x}}$ and their standard deviation $\sigma_{\mathbf{x}}$), as follows:

$$\begin{gathered}\mathcal{L}_f(\mathbf{x}_a) = p(\mathbf{x}_a,\mu_n,\sigma_n)\log\frac{p(\mathbf{x}_a,\mu_n,\sigma_n)}{p(\mathbf{x}_a,\mu_{\mathbf{x}},\sigma_{\mathbf{x}})} - \beta_f, \\ \text{where} \quad p(\mathbf{x}_a,\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{1}{2}(\frac{(\mathbf{x}_a)-\mu}{\sigma})^2}.\end{gathered} \tag{9}$$

Adversarial Edge Sampler For the target node v_i, given the generated node features $\mathbf{x}_a$ from g_n, the adversarial edge sampler g_e aims at connecting v_a to the current graph G'_t. Similar to g_n, g_e is equipped with a K-stacked GCLs, formulated as $\mathbf{H}_e^{(K+1)} = f_{GCL}^{(e,K)}(\hat{\mathbf{A}}(v_i), \mathbf{H}_e^{(K)}, \mathbf{W}_e^{(K)})$, where $\mathbf{H}_e^{(0)} = \mathbf{X}'$, and $\mathbf{W}_e^{(K)}$ is the parameter matrix of g_e's K-th convolution layer. Then, we concatenate $\mathbf{x}_a$ with each row of $\mathbf{H}_e^{(K+1)}$ to obtain $\mathbf{Z}_e \in \mathbb{R}^{|V'|\times(d+F)}$. The probability vector of remaining nodes connecting to v_a is calculated as:

$$\mathbf{p}_e = \mathrm{softmax}(\mathbf{Z}_e\cdot\mathbf{W}_e + \mathbf{A}[v_i]), \tag{10}$$

where $\mathbf{W}_e \in \mathbb{R}^{(d+F)}$ is the learnable parameter matrix and $\mathbf{A}[v_i]$ denotes v_i's row in the adjacency matrix of G'_t. We add $\mathbf{A}[v_i]$ to the probability logits because in order for the adversarial perturbation to be perceived by f, the introduced edge must enable $\mathbf{v}_a$ to lie in the receptive field of f. Then, we sample an edge from an one-hot categorical distribution parameterized by $\mathbf{p}_e$, merge the sampled edge into G'_t and get G'_{t+1}. For the next edge sampling operation, G'_{t+1} is fed into g_e, and this process iterates until v_i is successfully evaded or the number of wired edges reaches to β_e, as shown in Figure 1 (c).

Value Predictor Along with the policy learners g_n and g_e we have proposed, the value predictor g_v is the other component of A2C that aims at predicting the expected accumulated rewards at the end of the MDP. Given the dedicated reward function we have defined in Eq. (4), g_v predicts the final accumulated loss score of targeted node based on the current G'_t. We formulate this process as a regression task, where g_v predicts the negative log likelihood between the class log probabilities in current graph G'_t and $f(G, v_i)$. Specifically, a GNN model with K-stacked layers is utilized to capture the node topological information, similar to g_e, formulated as: $\mathbf{H}_v^{(K+1)} = f_{GCL}^{(v,K)}(\hat{\mathbf{A}}, \mathbf{H}_v^{(K)}, \mathbf{W}_v^{(K)})$, where $\mathbf{W}_v^{(K)}$ is the parameter matrix of g_v's K-th convolution layer. As shown in Figure 1 (d), we extract node v_i's embedding and concatenate it with f's output to predict the value score, formulated as:

$$\begin{aligned}&g_v(v_i, G'_t) = \\ &NLL\Big(\big(\mathbf{H}_v^{(K+1)}(v_i)||f(v_i,G'_t)\big)\cdot\mathbf{W}_v, f(v_i,G)\Big)\end{aligned} \tag{11}$$

where $NLL(\cdot,\cdot)$ is the negative log likelihood function, and $\mathbf{W}_v \in \mathbb{R}^{(d+C)\times C}$ is the learnable parameter.

Training Algorithm

To train G^{2}A2C = $\{g_n, g_e, g_v\}$, we explore the experience replay technique with memory buffer $\mathcal{M}$. Intuitively, we simulate the selection process to generate the training data and store the experience in the memory buffer during the forward runs of training phase. An instance in $\mathcal{M}$ is in the format of triplet (G'_t, a_t, R_t) with return $R_t = \sum_{j=t}^{j=T} r(v_i, a_j, G'_j) \cdot \gamma^{(j-t)}$, where γ refers to the discount factor. During the back-propagation, three losses are involved: policy loss $\mathcal{L}_p$, value loss $\mathcal{L}_v$ and feature loss $\mathcal{L}_f$. Given a triplet $(G'_t, a_t, R_t) \in \mathcal{M}$, $\mathcal{L}_p$ is calculated as:

$$\mathcal{L}_p(G'_t, a_t, R_t) = -\log\big(p(a_t|G'_t)\big) \cdot \big(R_t - g_v(v_i, G'_t)\big),$$

where $p(a_t|G'_t)$ denotes the probability of conducting action a_t under the graph G'_t. In $\mathcal{L}_p$, the second term $(R_t - g_v(v_i, G'_t))$ is also known as the advantage score (Mnih et al. 2016), which depicts how much better of selecting action a_t over the other actions. $\mathcal{L}_p$ enforces G^{2}A2C to deliver better actions with higher probabilities. On the other hand, value loss $\mathcal{L}_v$ enforces the value predictor g_v to correctly deliver the actual accumulated reward, calculated as:

$$\mathcal{L}_v(G'_t, R_t) = |g_v(v_i, G'_t) - R_t|. \tag{12}$$

The final loss for G^{2}A2C is formulated as:

$$\mathcal{L} = \sum_{\mathcal{M}} \big(\mathcal{L}_v(G'_t, R_t) + \mathcal{L}_p(G'_t, a_t, R_t)\big) + \sum_{\mathbf{x}_a \in \mathbf{X}_t^{\mathcal{A}}} \mathcal{L}_f(\mathbf{x}_a).$$

Experiment

In this section, we aim at answering the following four research questions: (**RQ1**) Can our proposed G^{2}A2C effectively evade target nodes given a well trained GNN for various datasets? (**RQ2**) Can the "gradient-free" property enhance attack performance when inaccurate victim model architecture is approximated? (**RQ3**) What is the attack performance of G^{2}A2C under different budgets? (**RQ4**) How does G^{2}A2C conduct the node injection attack in real cases?

Dataset. We conduct experiments on eight acknowledged benchmark datasets, namely `Cora`, `Citeseer`, `Pubmed` (Sen et al. 2008), `Amazon Photo`, `Amazon Computers` (McAuley, Pandey, and Leskovec 2015), `Wiki. CS` (Mernyei and Cangea 2020), `Reddit` (Hamilton, Ying, and Leskovec 2017), and `OGB-Products` (Hu et al. 2020). These datasets cover a broad range of fields, such as social networks, merchandise networks and citation networks. Besides, node features of these datasets cover both discrete and continuous spaces, to validate the attack performance of all baselines under various scenarios. We use public splits for training and evaluation on `Cora`, `Citeseer` and `Pubmed`, the random splits of 10%/10%/80% for `Amazon Photo`, `Amazon Computers` and `Wiki. CS`. For `OGB-Products` and `Reddit`, we explore the sub-graphs and splits shared by G-NIA (Tao et al. 2021) for fair comparison. The dataset statistics are shown in Table 1.

Dataset	Node	Edge	Class	Dim.
Datasets with Discrete Feature Space				
Cora*	2,708	5,429	7	1,433
Citeseer*	3,327	4,732	6	3,703
Am. Photo	7,650	119,043	8	745
Am. Comp.	13,752	245,778	10	767
Datasets with Continuous Feature Space				
Pubmed*	19,717	44,338	3	500
Wiki. CS	11,701	216,123	10	300
OGB-Prod.*	10,494	77,656	35	100
Reddit*	10,004	37,014	41	602

Table 1: Dataset statistics. For the datasets with $*$, we explore the public splits (Kipf and Welling 2016). And for datasets with $\star$, we use the largest connect component subgraphs acquired from Tao et al. (2021).

Baselines. Since black-box node injection attack is an emerging and far less researched area, only few methods focus on this topic, such as NIPA (Sun et al. 2019), G-NIA (Tao et al. 2021), AFGSM (Wang et al. 2020), and TDGIA (Zou et al. 2021). To sufficiently demonstrate the effectiveness of G^{2}A2C, besides these methods, we also compare G^{2}A2C with the adaptions of the state-of-the-art structural perturbation method Nettack (Zügner, Akbarnejad, and Günnemann 2018). Accordingly, our baselines include: NIPA (Sun et al. 2019) and its variant (i.e., Node+NIPA), two variants of Nettack (Zügner, Akbarnejad, and Günnemann 2018) (i.e., Rand.+Nettack and Node+Nettack), G-NIA (Tao et al. 2021), AFGSM (Wang et al. 2020), and TDGIA (Zou et al. 2021). "Rand" and "Node" refer to the random-generated and the G^{2}A2C-generated node features, respectively. To compare G^{2}A2C with white-box (i.e., Nettack, AFGSM and G-NIA) or grey-box (i.e., NIPA) approaches requiring gradient from the victim model, we train a 2-layer GCN as the surrogate model.

Experimental Setup

For the baselines, we explore DeepRobust (Li et al. 2020) and open-source code with the default settings. We set the hyper-parameters in G^{2}A2C as following: the number of GCLs K to 2, the temperature of Gumbel-Softmax to 1.0, hidden dimension d to 256, and the discount factor to 0.95. We utilize Adam optimizer with learning rate 10^{-4}. Besides, we adopt the early stopping with a patience of 3 epochs. All experiments are conducted for 10 times with mean and deviation reported.

Performance Comparison

We perform single injection attack, the most extreme setting where only one injected node with one edge is allowed to attack a target node. The results of G^{2}A2C and all baselines are reported in Table 2. We can observe that all baselines cause a performance downgrade to the victim model, and the introduced perturbation on the datasets with con-

Attacker	Discrete Feature Space				Continuous Feature Space			
	Cora	Citeseer	Am. Comp.	Am. Photo	OGB-Prod.	Reddit	Pubmed	Wiki. CS
Clean	18.4	21.1	24.37	17.8	24.3	8.5	21.9	21.3
NIPA	$18.6_{\pm 0.1}$	$21.1_{\pm 0.}$	$25.0_{\pm 0.2}$	$17.8_{\pm 0.}$	$25.9_{\pm 0.2}$	$12.5_{\pm 0.7}$	$21.9_{\pm 0.}$	$25.2_{\pm 0.4}$
Node+NIPA	$25.3_{\pm 0.4}$	$33.5_{\pm 0.6}$	$32.6_{\pm 0.7}$	$27.2_{\pm 1.0}$	$65.3_{\pm 0.4}$	$44.2_{\pm 0.2}$	$45.0_{\pm 0.1}$	$56.0_{\pm 0.3}$
Rand+Nettack	$24.3_{\pm 0.3}$	$32.1_{\pm 1.1}$	$30.5_{\pm 1.4}$	$22.4_{\pm 1.2}$	$63.3_{\pm 0.5}$	$31.2_{\pm 0.9}$	$46.7_{\pm 0.6}$	$53.9_{\pm 1.2}$
Node+Nettack	$27.4_{\pm 0.4}$	$37.2_{\pm 1.3}$	$\underline{39.9_{\pm 1.3}}$	$27.7_{\pm 1.4}$	$78.6_{\pm 0.3}$	$63.2_{\pm 0.5}$	$53.6_{\pm 0.7}$	$78.3_{\pm 0.8}$
AFGSM	$26.3_{\pm 4.2}$	$38.6_{\pm 3.2}$	$37.5_{\pm 1.9}$	$32.3_{\pm 1.1}$	$74.9_{\pm 0.7}$	$45.8_{\pm 0.7}$	$65.8_{\pm 0.9}$	$77.4_{\pm 0.6}$
TDGIA	$\underline{29.5_{\pm 2.8}}$	$\underline{44.2_{\pm 2.2}}$	$39.4_{\pm 1.1}$	$\underline{32.5_{\pm 0.7}}$	$93.3_{\pm 0.2}$	$91.8_{\pm 0.5}$	$67.2_{\pm 0.4}$	$\underline{84.2_{\pm 1.1}}$
G-NIA	$24.3_{\pm 2.5}$	$36.5_{\pm 3.1}$	$34.4_{\pm 1.4}$	$25.2_{\pm 1.4}$	$\underline{95.0_{\pm 0.4}}$	$\underline{94.6_{\pm 1.2}}$	$\underline{68.3_{\pm 1.0}}$	$81.1_{\pm 1.2}$
G^2A2C (ours)	$\mathbf{36.3}_{\pm 2.7}$	$\mathbf{49.4}_{\pm 2.8}$	$\mathbf{42.2}_{\pm 1.1}$	$\mathbf{33.6}_{\pm 1.2}$	$\mathbf{97.4}_{\pm 0.4}$	$\mathbf{98.7}_{\pm 0.6}$	$\mathbf{74.1}_{\pm 0.8}$	$\mathbf{86.6}_{\pm 0.8}$
Avg. ↑	6.8	5.2	2.3	1.1	2.4	4.1	5.8	2.4

Table 2: Miclassification rate (%) of a trained two-layer GCN model after the single node injection attack (i.e., $\beta_e = 1$, $\beta_n = 1$, and $\beta_f = 0$) launched by the different attackers. Rate in bold indicates the best and rate in underline is the second best. The results reported above are averaged over 10 independent runs with different initialization seeds.

tinuous feature space is more severe than those with discrete feature space, demonstrating the obstacle from the unsmooth feedback during the attack phase. Compared with all baselines, on average, G^2A2C increases the misclassification rate by 3.85 on discrete datasets and by 3.68 on continuous datasets. We further conduct a t-test on the results of G^2A2C and the best performing baseline in each dataset, and the performance improvement brought by G^2A2C is significant with 95% confidence, demonstrating the superior and stable performance of G^2A2C. By comparing perturbation models of random features with those utilizing the generated adversarial features from G^2A2C (e.g., Rand+Nettack vs. Node+Nettack), we can clearly observe that the latter greatly imperils the performance of the victim model, and all baselines can cause considerable downgrade with the generated features from G^2A2C, which further demonstrates the legitimacy of the node generator in G^2A2C. To answer **RQ1**: under the black-box setting, G^2A2C outperforms all baselines and by a significant margin across all datasets in the most extreme setting. Besides, the performance of weak baselines can be significantly boosted by the adversarial features generated by G^2A2C, as demonstrated by Node+NIPA and Node+Nettack. By comparing Node+Nettack with G^2A2C, we observe a higher performance for G^2A2C, indicating the outstanding edge wiring capability of the edge sampler.

To further validate the performance of G^2A2C, we change the backbone GNN of the victim model to GAT (Veličković et al. 2017), APPNP (Klicpera, Bojchevski, and Günnemann 2019), and SGC (Wu et al. 2019) and launch attacks by the best-performing baselines as well as G^2A2C. In this setting, we do not accordingly modify the surrogate model architecture or the backbone GNN of G^2A2C to intentionally create an extreme setting where the attacker possesses an inaccurate approximation of the victim model. The results are shown in Table 3. We can observe that the performances of all attackers are significantly enhanced on shallow models such as SGC, indicating the vulnerabilities of the shallow and less-parametrized models. While attacking the over-parametrized GAT model, G-NIA delivers downgraded performances, but TDGIA and G^2A2C show stronger performance compared with the performance of attacking other shallow models (e.g., GCN and SGC). This phenomenon demonstrates that the vulnerabilities of GAT is dependent on the different datasets. Overall, G^2A2C outperforms all best-performing baselines over these three exemplary datasets. To answer **RQ2**, when the attacker's assumptions on the architecture of the victim model are incorrect, the effectiveness of G^2A2C is barely downgraded, demonstrating the advantages brought by the "gradient-free" property.

Attacker	Backbone	Cora	Citeseer	Pubmed
G-NIA	GCN	$24.3_{\pm 2.5}$	$36.5_{\pm 3.1}$	$68.3_{\pm 1.0}$
	SGC	$27.6_{\pm 2.9}$	$40.1_{\pm 3.5}$	$70.2_{\pm 1.1}$
	GAT	$22.1_{\pm 2.1}$	$35.3_{\pm 1.8}$	$68.8_{\pm 0.7}$
	APPNP	$24.2_{\pm 2.0}$	$37.2_{\pm 3.4}$	$69.7_{\pm 0.8}$
TDGIA	GCN	$29.5_{\pm 2.8}$	$44.2_{\pm 2.2}$	$67.2_{\pm 0.4}$
	SGC	$35.2_{\pm 2.1}$	$51.2_{\pm 1.1}$	$74.4_{\pm 1.5}$
	GAT	$28.3_{\pm 2.0}$	$52.1_{\pm 1.4}$	$81.2_{\pm 0.2}$
	APPNP	$29.1_{\pm 2.4}$	$43.8_{\pm 1.1}$	$69.6_{\pm 2.9}$
G^2A2C	GCN	$36.3_{\pm 2.7}$	$49.4_{\pm 2.8}$	$74.1_{\pm 0.8}$
	SGC	$41.6_{\pm 3.1}$	$53.8_{\pm 1.4}$	$76.3_{\pm 2.2}$
	GAT	$33.4_{\pm 3.3}$	$55.6_{\pm 1.6}$	$85.0_{\pm 0.3}$
	APPNP	$36.4_{\pm 2.7}$	$48.0_{\pm 1.3}$	$82.9_{\pm 3.2}$

Table 3: Misclassification rate (%) to different two-layer GNNs. The same setting as reported in Table 2 is explored.

Budget Analysis

In this section, we conduct experiments w.r.t. all the attack budgets in our setting: the edge budget β_e, the node budget β_n and the feature shift budget β_f, as shown in Figure 2. From these results, to answer **RQ3**, the most fruitful budget is β_n, and with 3 injected nodes per target, the performance of GCN on all datasets is downgraded to the range around

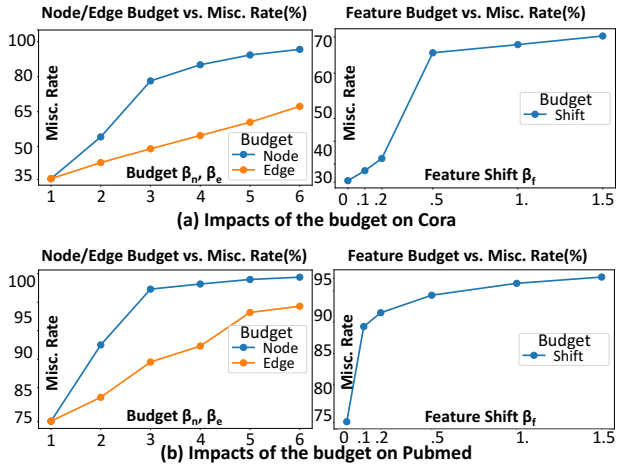

Figure 2: Effectiveness of G^{2}A2C under different budgets.

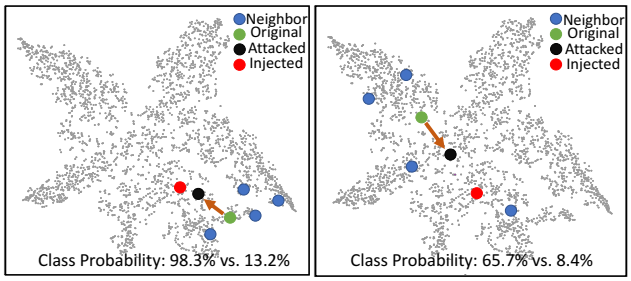

Figure 3: Visualization of the attack launched by G^{2}A2C.

10%. The second most impactful budget is β_f, and the performance of GCN all falls down below 10% with a distribution shift budget of 0.5 and its impact saturates around 0.5. The least impactful budget is β_e. We observe that its impact is relatively linear compared with other budgets.

Case Study

To further investigate how G^{2}A2C conducts node injection attack, we visualize two successful attacks on Citeseer. As shown in Figure 3, we visualize the attack process by plotting the hidden embedding of the involved nodes, extracted from the victim model, before and after the attack via T-SNE (Van der Maaten and Hinton 2008). In this figure, blue points are target node's original neighbors in the clean graph, green point is the target node before the attack, black point refers to the attacked target node, and red point refers to the injected adversarial node. To answer **RQ4**, as shown in these two cases, the injected node could effectively perturb the embedding of the target node, relocate it to a relatively intertwined position, and hence flip its prediction.

Related Work

GNNs have been proved to be sensitive to adversarial attacks (Dai et al. 2018; Ma et al. 2019; Wang and Gong 2019; Xu et al. 2019; Zügner, Akbarnejad, and Günnemann 2018; Sun et al. 2019; Tao et al. 2021; Wang et al. 2020). Most of them focus on perturbations on existing knowledge, such as topological structure (Xu et al. 2019; Wang and Gong 2019; Ma et al. 2019), node attributes (Zügner, Akbarnejad, and Günnemann 2018), and labels (Sun et al. 2019). However, in the real world, modifying existing edges or node attributes is not practical, due to limited access to the node of interests. Node injection attack aims at a more realistic scenario, which adds adversarial nodes in to the existing graph. State-of-the-art attackers (Sun et al. 2019; Tao et al. 2021; Wang et al. 2020; Zou et al. 2021) either explore the less practical white-box setting, or training a surrogate model to simulate a white-box setting, which might introduce performance downgrade when inaccurate approximations about the victim model are made. For the white-box setting, NIPA (Sun et al. 2019) creates a batch of random nodes and wiring them to the existing graph to fulfill the malicious intent. And for the black-box setting, AFGSM (Wang et al. 2020) utilizes a fast gradient sign method and G-NIA (Tao et al. 2021) explores a neural network to generalize the attacking process. TDGIA firstly selects topological defective edges to the injected node, and then generates the adversarial features for the injected nodes according to the surrogate model. To further leverage the practicality as well as effectiveness, we study the node injection attack under the black-box setting without training a surrogate model to acquire simulated gradient, eliminating the possibility of error propagation due to inaccurate approximations about the victim model.

Conclusion

In this work, we study gradient-free node injection evasion attack for graphs under the black-box setting. Unlike other node injectors requiring gradient from the surrogate model, we propose G^{2}A2C, a gradient-free attacker without any assumption on the victim model, eliminating the possibility of error propagation due to inaccurate approximations about the victim model. We formulate such attack as an MDP and solve it through our designed graph reinforcement learning framework. Our node generator generates imperceptible yet malicious node features, followed by the edge sampler that wires the node to the remaining graph. Through comprehensive experiments with the state-of-the-art baselines, we demonstrate the promising performance of G^{2}A2C over eight acknowledged datasets with diverse characteristics. And by modifying the architectures of the victim model to four different GNNs, we empirically prove the advantage brought by the "gradient-free" property of G^{2}A2C.

Acknowledgments

This work is partially supported by the NSF under grants IIS-2209814, IIS-2203262, IIS-2214376, IIS-2217239, OAC-2218762, CNS-2203261, CNS-2122631, CMMI-2146076, and the NIJ 2018-75-CX-0032. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of any funding agencies.

References

Bengio, Y.; Léonard, N.; and Courville, A. 2013. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*.

Chen, Y.; Yang, H.; Zhang, Y.; KAILI, M.; Liu, T.; Han, B.; and Cheng, J. 2022. Understanding and Improving Graph Injection Attack by Promoting Unnoticeability. In *Procs. of ICLR*.

Dai, H.; Li, H.; Tian, T.; Huang, X.; Wang, L.; Zhu, J.; and Song, L. 2018. Adversarial attack on graph structured data. In *Procs. of ICML*.

Hamilton, W. L.; Ying, R.; and Leskovec, J. 2017. Inductive representation learning on large graphs. In *Procs. of NeurIPS*.

Hu, W.; Fey, M.; Zitnik, M.; Dong, Y.; Ren, H.; Liu, B.; Catasta, M.; and Leskovec, J. 2020. Open graph benchmark: Datasets for machine learning on graphs. *In Procs. of NeurIPS*.

Jang, E.; Gu, S.; and Poole, B. 2016. Categorical reparameterization with gumbel-softmax. *In Procs. of ICLR*.

Jiang, D.; Wu, Z.; Hsieh, C.-Y.; Chen, G.; Liao, B.; Wang, Z.; Shen, C.; Cao, D.; Wu, J.; and Hou, T. 2021. Could graph neural networks learn better molecular representation for drug discovery? A comparison study of descriptor-based and graph-based models. *Journal of cheminformatics*.

Ju, M.; Hou, S.; Fan, Y.; Zhao, J.; Ye, Y.; and Zhao, L. 2022. Adaptive kernel graph neural network. In *Procs. of AAAI*.

Kipf, T. N.; and Welling, M. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.

Klicpera, J.; Bojchevski, A.; and Günnemann, S. 2019. Predict then Propagate: Graph Neural Networks meet Personalized PageRank. *In Procs. of SIGKDD*.

Li, Y.; Jin, W.; Xu, H.; and Tang, J. 2020. Deeprobust: A pytorch library for adversarial attacks and defenses. *arXiv preprint arXiv:2005.06149*.

Ma, Y.; Wang, S.; Derr, T.; Wu, L.; and Tang, J. 2019. Attacking graph convolutional networks via rewiring. *arXiv preprint arXiv:1906.03750*.

McAuley, J.; Pandey, R.; and Leskovec, J. 2015. Inferring networks of substitutable and complementary products. In *Procs. of SIGKDD*.

Mernyei, P.; and Cangea, C. 2020. Wiki-cs: A wikipedia-based benchmark for graph neural networks. *arXiv preprint arXiv:2007.02901*.

Mnih, V.; Badia, A. P.; Mirza, M.; Graves, A.; Lillicrap, T.; Harley, T.; Silver, D.; and Kavukcuoglu, K. 2016. Asynchronous methods for deep reinforcement learning. In *Procs. of ICML*.

Pal, A.; Eksombatchai, C.; Zhou, Y.; Zhao, B.; Rosenberg, C.; and Leskovec, J. 2020. Pinnersage: Multi-modal user embedding framework for recommendations at pinterest. In *Procs. of SIGKDD*.

Sen, P.; Namata, G.; Bilgic, M.; Getoor, L.; Galligher, B.; and Eliassi-Rad, T. 2008. Collective classification in network data. *AI magazine*.

Sun, Y.; Wang, S.; Tang, X.; Hsieh, T.-Y.; and Honavar, V. 2019. Node injection attacks on graphs via reinforcement learning. *In Procs. of WWW*.

Tao, S.; Cao, Q.; Shen, H.; Huang, J.; Wu, Y.; and Cheng, X. 2021. Single Node Injection Attack against Graph Neural Networks. *In Procs. of CIKM*.

Van der Maaten, L.; and Hinton, G. 2008. Visualizing data using t-SNE. *JMLR*.

Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Lio, P.; and Bengio, Y. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903*.

Wang, B.; and Gong, N. Z. 2019. Attacking graph-based classification via manipulating the graph structure. In *Procs. of SIGKDD*.

Wang, J.; Luo, M.; Suya, F.; Li, J.; Yang, Z.; and Zheng, Q. 2020. Scalable attack on graph data by injecting vicious nodes. *Data Mining and Knowledge Discovery*.

Wu, F.; Souza, A.; Zhang, T.; Fifty, C.; Yu, T.; and Weinberger, K. 2019. Simplifying graph convolutional networks. In *Procs. of ICML*.

Xu, K.; Chen, H.; Liu, S.; Chen, P.-Y.; Weng, T.-W.; Hong, M.; and Lin, X. 2019. Topology attack and defense for graph neural networks: An optimization perspective. *arXiv preprint arXiv:1906.04214*.

Xu, K.; Hu, W.; Leskovec, J.; and Jegelka, S. 2018. How powerful are graph neural networks? *In Procs. of ICLR*.

Ying, R.; He, R.; Chen, K.; Eksombatchai, P.; Hamilton, W. L.; and Leskovec, J. 2018. Graph convolutional neural networks for web-scale recommender systems. In *Procs. of SIGKDD*.

Zou, X.; Zheng, Q.; Dong, Y.; Guan, X.; Kharlamov, E.; Lu, J.; and Tang, J. 2021. TDGIA: Effective injection attacks on graph neural networks. In *Procs. of SIGKDD*.

Zügner, D.; Akbarnejad, A.; and Günnemann, S. 2018. Adversarial attacks on neural networks for graph data. In *Procs. of SIGKDD*.