Received: 13 July 2022 | Revised: 18 July 2022 | Accepted: 24 August 2022 | Published online: 28 October 2022

RESEARCH ARTICLE

Journal of Computational and Cognitive Engineering
2022, Vol. 1(4) 165–173
DOI: 10.47852/bonviewJCCE2202321

BON VIEW PUBLISHING

# Fake Node Attacks on Graph Convolutional Networks

**Xiaoyun Wang[1,*], Minhao Cheng[1], Joe Eaton[2], Cho-Jui Hsieh[1] and S. Felix Wu[1]**

[1]*University of California, USA*

[2]*NVIDIA, USA*

**Abstract:** In this paper, we study the robustness of graph convolutional networks (GCNs). Previous works have shown that GCNs are vulnerable to adversarial perturbation on adjacency or feature matrices of existing nodes; however, such attacks are usually unrealistic in real applications. For instance, in social network applications, the attacker will need to hack into either the client or server to change existing links or features. In this paper, we propose a new type of "fake node attacks" to attack GCNs by adding malicious fake nodes. This is much more realistic than previous attacks; in social network applications, the attacker only needs to register a set of fake accounts and link to existing ones. To conduct fake node attacks, a greedy algorithm is proposed to generate edges of malicious nodes and their corresponding features aiming to minimize the classification accuracy on the target nodes. In addition, we introduce a discriminator to classify malicious nodes from real nodes and propose a Greedy-generative adversarial network attack to simultaneously update the discriminator and the attacker, to make malicious nodes indistinguishable from the real ones. Our non-targeted attack decreases the accuracy of GCN down to 0.03, and our targeted attack reaches a success rate of 78% on a group of 100 nodes and 90% on average for attacking a single target node.

**Keywords:** neural networks, adversarial attacks, graph convolutional networks

## 1. Introduction

Graph data have been widely used in many real-world applications, such as social networks (Facebook and Twitter), biological networks (protein or gene interactions), as well as attribute graphs (PubMed and Arxiv) (Grover & Leskovec, 2016; Rhee et al., 2017; Ying et al., 2018). Node classification is one of the most important tasks on graphs—given a graph with labels associated with a subset of nodes, predict the labels for the rest of the nodes. For this node classification task, deep learning models on graphs, such as graph convolutional networks (GCNs), have achieved state-of-the-art performance (Kipf & Welling, 2016). Moreover, GCNs have wide applications in cyber security, where they can learn close-to-correct node labeling semi-autonomously. This reduces the load on security experts and helps to manage networks that add or remove nodes dynamically, such as WiFi networks in universities and web services in companies.

The power of GCN relies on the graph convolution operation, which constructs each node's embedding based on its neighbors' embeddings. However, this also leads to the concern about the robustness—Is it possible to perturb the links and embeddings of a small subset of nodes to affect the GCN's behavior on other nodes? To answer this question, Dai et al. (2018) and Zügner et al. (2018) develop algorithms to attack GCNs, showing that by altering a small amount of edges and features, the classification accuracy of a GCN can be reduced to chance level. However, changing edges or features associated with existing nodes is usually impractical. For example, in social network applications, an attacker has to login to the users' accounts to change existing connections and features, and gaining login access is almost impossible. In comparison, adding fake nodes that correspond to fake accounts or users can be much easier in practice. But the key question is that *can we interfere with the classification results of existing nodes by adding fake nodes to the network?* We answer this question affirmative by introducing novel algorithms to design fake nodes that successfully reduce GCN's performance on existing nodes.

To design the adjacency and feature matrices associated with fake nodes, we have to address two challenges. First, the edges and features are usually discrete 0/1 variables. Although there have been many algorithms proposed for attacking image classifiers, such as Fast Gradient Sign Method (FGSM), C&W, and PGD attacks (Carlini & Wagner, 2017; Goodfellow et al., 2014; Madry et al., 2017), they all assume continuous input space and cannot be directly applied to problems with discrete input space. Second, it is not easy to make the fake nodes "look" like the real ones. There are two aspects for realness of fake nodes,

*Corresponding author: Xiaoyun Wang, University of California, USA. Email: xiywang@ucdavis.edu

(i) in graph structure and (ii) in features. For example, if we add a fake node that connects to all existing nodes, the system can easily detect and disable such a fake node, and if a fake node features are all 1s or very large or small values in continuous cases, the system can also easily detect it. In this paper, we propose two algorithms, Greedy attack and Greedy-generative adversarial network (GAN) attack, to address these two challenges. Our contributions can be summarized below:

- To the best of our knowledge, this is the first paper studying how to add fake nodes to attack GCNs. We do not need to manipulate existing nodes' adjacency and feature matrices.
- We propose a Greedy attack algorithm to address the discrete input space problem in designing fake nodes' adjacency and feature matrices.
- We introduce a discriminator to classify fake nodes from real nodes and propose a Greedy-GAN algorithm to simultaneously optimize the discriminator and the attacker. Despite a slightly lower success rate, this approach can make fake nodes harder to detect.
- We conduct experiments on several real datasets. For non-targeted attacks, we are able to degrade the classifier's accuracy to less than 10% on both Cora and Citeseer datasets, and for the targeted attacks, we can achieve a $>70\%$ success rate when attacking either a single node or a group of nodes.

## 2. Related Work

### 2.1. Adversarial attacks

Adversarial examples for computer vision have been studied extensively. Goodfellow et al. (2014) discovered that deep neural networks are vulnerable to adversarial attacks—a carefully designed small perturbation can easily fool a neural network. Several algorithms have been proposed to generate adversarial examples for image classification tasks, including FGSM (Goodfellow et al., 2014), Iterative Fast Gradient Sign Method (IFGSM) (Kurakin et al., 2016), C&W attack (Carlini & Wagner, 2017), and PGD attack (Madry et al., 2017). In the black-box setting, it has also been reported that an attack algorithm can have high success rate using finite difference techniques (Chen et al. 2017; Cheng et al., 2018), and several algorithms are recently proposed to reduce query numbers (Ilyas et al., 2018; Suya et al., 2017). Transfer attack (Papernot et al., 2016) and ensemble attack (Tramèr et al., 2017) can also be applied to black-box setting, with lower successful rate but less number of queries. Besides attacking image classification, CNN-related attacks have also been explored. A typical usage is attacking semantic segmentation and object detection (Arnab et al. 2017; Cheng et al., 2018; Eykholt et al. 2017; Lu et al. 2017; Metzen et al. 2017; Xie et al. 2017). Image captioning (Chen et al., 2017) and visual QA (Xu et al., 2017) could also be attacked.

Most of the above-mentioned works are focusing on problems with continuous input space (such as images). When the input space is discrete, attacks become discrete optimization problems and are much harder to solve. This happens in most natural language processing applications. For text classification problem, FGSM is firstly applied by Papernot et al. (2016). The follow-up works include deleting important words (Li et al., 2016), replacing or inserting words with typos and synonyms (Liang et al., 2017; Samanta & Mehta, 2017). For black-box setting, Gao et al. (2018) develop score functions to find out the words to modify. Jia and Liang (2017) add misleading sentences to fool the reading comprehension system. Zhao et al. (2017) use GAN to generate natural adversarial examples. Ebrahimi et al. (2017) and Cheng et al. (2018) attack machine translation system Seq2Seq by changing words in text.

### 2.2. Graph convolutional neural networks

Graph convolutional networks (GCNs) have been widely used for node classification and other graph-based applications. The main idea of GCN is to aggregate associated information from a node and its neighbors using some aggregation functions. They train the aggregation steps and the final prediction layer end-to-end to achieve better performance than traditional approaches. There are several variations of GCNs proposed recently (Chen et al., 2018; Defferrard et al., 2016; Hamilton et al., 2017; Kipf & Welling, 2016; Pham et al., 2016; Ying et al., 2018).

The wide applicability of GCNs motivates recent studies about their robustness. Dai et al. (2018) and Zügner et al. (2018) recently proposed algorithms to attack GCNs by changing existing nodes' links and features. Zügner et al. (2018) developed an FGSM-based method that optimizes a surrogate model to choose the edges and features that should be manipulated. Dai et al. (2018) proposed several attacking approaches including gradient ascent, genetic algorithm, and reinforcement learning. They employed a gradient ascent method to change the graph structure in the white-box setting. However, their settings are different from ours in the following aspects: (1) Our methods have different constraint than the previous works. They considered altering edges or features of existing nodes while we develop algorithms to add fake nodes to interfere the performance of GCN. In their scenarios, keeping graph structure perturbations unnoticeable is important. While in our adding fake nodes scenario, only keeping structure perturbations unnoticeable is far from enough. Keeping fake nodes features realness is very important, if a fake nodes features are very unreal, the system can also easily detect it and disable this fake node. (2) They attacked nodes one-by-one while our algorithms can find a small perturbation to attack a group of nodes.

## 3. Preliminary

We consider the node classification problem using GCN. Given an adjacency matrix $A \in R^{n\times n}$, feature matrix $X \in R^{n\times d}$, and a subset of labeled nodes, the goal is to predict the labels of all the nodes in the graph. We use $n$ to denote the number of nodes, $P$ to denote the number of labels, and $d$ to denote the dimension of features (each row of $X$ corresponds to features of a node). There are several variations of GCNs, but we consider one of the most common approaches introduced in Kipf and Welling (2016). Starting from $H^0 = X$, GCN uses the following rule to iteratively aggregate features from neighbors:

$$H^{(k+1)} = \sigma(\hat{A}H^{(k)}W^{(k)}), \tag{1}$$

where $\hat{A}$ is the normalized adjacency matrix and $\sigma$ is the activation function. There are two common ways for normalizing $A$. The original GCN paper applied **symmetric normalization**, which sets $\tilde{A} = A + I$ (adding 1 to all the diagonal entries) and then normalizes it by $\hat{A} = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}$ where $\tilde{D}$ is a diagonal matrix with $\tilde{D}_{i,i} = \sum_j \tilde{A}_{ij}$. Another choice for $\hat{A}$ is to normalize the adjacency matrix by rows, leading to $\hat{A} = \tilde{D}^{-1}\tilde{A}$, which is referred to as **row-wise normalization**.

We set $\sigma(x) = \mathrm{ReLU}(x) = \max(0, x)$, which is the most common choice in practice. For a GCN with $K$ layers, after getting the top-layer feature $H^{(K)}$, a fully connected layer with softmax loss is

used for classification. Note that two types of normalization have been used in GCN.

A commonly used application is to apply two-layer GCN to semi-supervised node classification on graphs (Kipf & Welling, 2016). The model could be simplified as:

$$\begin{aligned} Z &= \text{softmax}(\hat{A}\sigma(\hat{A}XW^{(0)})W^{(1)}) \\ &:= \text{softmax}(f(X,A)). \end{aligned} \tag{2}$$

Note that each row of $f(X, A) \in R^{n\times P}$ is the logit output for a node over $P$ labels, and the softmax is taken on each row of the matrix.

For simplicity, we will assume our target network is structured as (2), but in general, our algorithm can be used to attack GCNs with more layers.

## 4. Attack Algorithms

In this section, we will describe our "fake node" attacks. We will describe both a non-targeted attack, which aims to lower the accuracy of all existing nodes uniformly, and a targeted attack, which attempts to force the GCN to give the desired label to nodes. Instead of manipulating the feature and adjacency matrices of existing nodes, we insert $m$ fake nodes with corresponding fake features into the graph. After that, the adjacency matrix is $A' = \begin{Bmatrix} A & B^T \\ B & C \end{Bmatrix}$ and the feature matrix becomes: $X' = \begin{Bmatrix} X \\ X_{\text{fake}} \end{Bmatrix}$. Note that $A$ is the original adjacency matrix and $X$ is the original feature matrix. Starting from $B = 0, C = I$, our goal is to design $B, C, X_{\text{fake}}$ to achieve the desired objective (e.g., lower the classification accuracy on existing nodes).

### 4.1. Non-targeted attack

The goal of the non-targeted attacks is to lower the classification accuracy on all the existing nodes by designing features and links of fake nodes. We use the accuracy of GCN to measure the effectiveness of attacks. We will present two different algorithms to attack GCNs: A greedy attack that updates links and features one by one, and Greedy-GAN attack that uses a discriminator to generate unnoticeable features of fake nodes.

#### *4.1.1. Greedy attack*

We define the objective function of the attack as

$$J(A', X') = \sum_{i\in S}\Big(\max\left([f(X', A')]_{i,:}\right) - [f(X', A')]_{i,y_i}\Big), \tag{3}$$

where $y_i$ is the correct label of node $i$; $S$ is a group of $m$ target nodes $S = \{v_1, \ldots, v_m\}$. In this objective function, if the largest logit of node $i$ is not the correct label $y_i$, it will encounter a positive score; otherwise, the score will be zero. We then try to solve the following optimization problem to form the fake nodes:

$$\arg\max_{B,C,X_{\text{fake}}} J(A', X') \quad \text{s.t. } \|B\|_0 + \|C\|_0 + \|X_{\text{fake}}\|_0 \le T, \tag{4}$$

where $\|\cdot\|_0$ denotes the number of nonzero elements in the matrix. Also, we assume $B, C, X_{\text{fake}}$ can only be 0/1 matrices. Unlike images, graphs have discrete values in the adjacency matrix, and in many applications, the feature matrix comes from indicators of different categories. For some attribute graphs, feature matrices are generated using word count; in practice, feature matrices are from categorical features, and thus they are often discrete.

**Algorithm 1**
**Greedy attack**

**Input:** Adjacency matrix $A$; feature matrix $X$; A classifier $f$ with loss function $J$; number of iterations $T$.
**Output:** Modified graph and features $G' = (A', X')$ after adding fake nodes.
**for:** $t = 0$ to $T - 1$ **do**
  **Let** $e^* = (u^*, v^*) \leftarrow \arg\max \nabla_{B,C} J(A', X')$
    $G_{B,C}^{(t+1)} \leftarrow G_{B,C}^{(t)} + e^*$
  **Let** $f^* = (u^*, i^*) \leftarrow \arg\max \nabla_{X_{\text{fake}}} J(A', X')$
    $G_{X_{\text{fake}}}^{(t+1)} \leftarrow G_{X_{\text{fake}}}^{(t)} + f^*$
**return:** $G^{(t)}$

Therefore, gradient-based techniques such as FGSM and PGD cannot be directly applied. Instead, we propose a greedy approach—starting from $B, X_{\text{fake}}$ all being zeros, and $C$ being identity matrix $I$, we add one feature and one edge at each step. To add a feature, we find the maximal element in $\nabla_{X_{\text{fake}}} J(A', X')$ and turn it into nonzero. Similarly, we find the maximal element in $\nabla_{B,C} J(A', X')$ and add the entry to the adjacency matrix. The Greedy attack is presented in Algorithm 1.

In the algorithm, when adding links and features, we make sure that there is no such a link or feature before adding. Also, although in Algorithm 1 we conduct one feature update and one edge update at each iteration, in practice we can adjust the frequency according to the data distribution. For example, if the original adjacency matrix has twice the nonzero elements as the feature matrix, we can update two elements in the adjacency matrix and one element in the feature matrix at every iteration.

#### *4.1.2. Greedy-GAN attack*

Unlike changing existing graph methods (Dai et al., 2018; Zügner et al., 2018), adding fake nodes on graphs is faced with different "unnoticeable" definitions, and it has different constraints from the existing works on node features. Making edges perturbation unnoticeable is needed, but it is far from enough for adding fake nodes setting. For example, if add nodes' features are very different from the existing nodes, it will be detected by data cleaning easily. Facing such a different constraint and ensuring fake nodes' features are realistic become crucial. Next, we will present the attack based on the idea of GAN.

The main idea is to add a discriminator to generate fake features that are similar to the original ones. To achieve this goal, we first design a neural network with two fully connected layers plus a softmax layer as the discriminator, which can be written as

$$D(X') = \text{softmax}(\sigma(X'W^{(0)})W^{(1)}), \tag{5}$$

where softmax works on each row of the output matrix. Each element in $D(X')$ indicates whether the discriminator classifies the node as real or fake.

We aim to generate fake nodes with features similar to the real ones to fool the discriminator. Since the output of the discriminator is binary, we use binary cross entropy loss defined by $L(p,y) = -(y\ \log(p)+(1-y)\log(1-p))$, where $y$ is a binary indicator of the groundtruth (real or fake nodes), and $p$ is the predicted probability by our discriminator. Then, we solve the following optimization problem:

**Algorithm 2**
**Greedy-GAN attack**

**Input:** Adjacency matrix $A$; feature matrix $X$; A classifier $f$ with loss function $J$; Discriminator $D$ with loss function $L$; number of outer iterations $T_{\text{outer}}$, and inner iterations $T_{\text{inner}}$.
**Output:** Modified graph $G' = (A', X')$ after adding fake nodes.
**for:** $t_1 = 0$ to $T_{\text{outer}} - 1$ **do**
  **for:** $t_2 = 0$ to $T_{\text{inner}} - 1$ **do**
    **Let** $e^*_{\text{add}} = (u^*_{\text{add}}, v^*_{\text{add}}) \leftarrow \arg\max \nabla_{B,C} Q$
      $e^*_{\text{drop}} = (u^*_{\text{drop}}, v^*_{\text{drop}}) \leftarrow \arg\min \nabla_{B,C} Q$
    **if** $|\nabla_{B,C} Q_{e^*_{\text{add}}}| > |\nabla_{B,C} Q_{e^*_{\text{drop}}}|$:
      $G_{B,C} \leftarrow G_{B,C} + e^*_{\text{add}}$
    **else**:
      $G_{B,C} \leftarrow G_{B,C} - e^*_{\text{drop}}$
    **Let** $f^*_{\text{add}} = (u^*_{\text{add}}, i^*_{\text{add}}) \leftarrow \arg\max \nabla_{X_{\text{fake}}} Q$
      $f^*_{\text{drop}} = (u^*_{\text{drop}}, i^*_{\text{drop}}) \leftarrow \arg\min \nabla_{X_{\text{fake}}} Q$
    **if** $|\nabla_{X_{\text{fake}}} Q_{f^*_{\text{add}}}| > |\nabla_{X_{\text{fake}}} Q_{f^*_{\text{drop}}}|$:
      $G_{X_{\text{fake}}} \leftarrow G_{X_{\text{fake}}} + f^*_{\text{add}}$
    **else**:
      $G_{X_{\text{fake}}} \leftarrow G_{X_{\text{fake}}} - f^*_{\text{drop}}$
  **end for**
  **update** discriminator by $u$ times.
**end for**
**return**: $G$

$$\arg\max_{B,C,X_{fake}} \min_{D} (\underbrace{J(A', X') + c * L(D(X'), Y)}_{Q}), \tag{6}$$

where $Y$ is the groundtruth (real/fake) indicator for nodes and $c$ is the parameter determined by the weight of the discriminator and the GCN performance. For example, if $c$ is large, the objective function is dominated by the discriminator, so the node features generated will be very close to real ones but with a lower attack success rate. We will discuss more in Section 5.7.

We adopt the GAN-like framework to train both features/adjacency matrices and discriminator parameters iteratively. In experiments, at each epoch we conduct 100 greedy updates for $B, C, X_{\text{fake}}$ and then 10 iterations of $D$ updates. The Greedy-GAN algorithm is presented in Algorithm 2. Greedy-GAN supports both adding and dropping links and features. In the algorithm, we add or drop elements according to the absolute gradient of elements, and the one with larger absolute value will be chosen.

The time complexity of Greedy and Greedy-GAN per iteration for updating edges of fake nodes costs $O(|V|*k+k^2)$, where $k$ is the number of added fake nodes; updating features of fake nodes costs $O(k * N_{\text{features}})$, $N_{\text{features}}$ is the number of features each node.

## 4.2. Targeted attack

Next, we extend the proposed algorithms to conduct targeted attacks. Given an adjacency matrix and a feature matrix, the goal is to make nodes classified as the desired class by adding fake nodes. Assume our goal is to attack a subset of nodes $S$ and $y_i$* is the target label for node $i$, the attack objective function can be defined as:

$$J(A', X') = \sum_{i \in S} \left( [f(X', A')]_{i, y_i^*} - \max \left( [f(X', A')]_{i,:} \right) \right), \tag{7}$$

In this objective function, if the largest logit of node $i$ is the target label $y_i$*, the objective function value will be 0; otherwise, the value is negative. Similar to the non-targeted attacks, we would like to find $B$, $C$, and $X_{\text{fake}}$ using Greedy attack to solve the optimization problem (4) and Greedy-GAN attack to solve the optimization problem (6).

In our experiments, we consider two cases: attacking a group of randomly sampled nodes ($S = \{v_i, \ldots, v_m\}$) and attacking one node (only one element in $S$). For attacking a group of nodes, the labels of the fake nodes are given in two ways: (1) uniform distribution, which is the same as in the non-targeted attack setting and (2) using the target labels $y_i$*. For attacking a single target node, we add three fake nodes with the target label.

## 5. Experimental Results

We use Cora (2708 nodes, 5429 edges, 1433 features) and Citeseer (3312 nodes, 4732 edges, 3703 features) attribute graphs as benchmarks. In all the experiments, we split the data into 10% for training, 20% for validation, and 70% for testing. For non-targeted attacks, the results are evaluated by GCN classification accuracy (the lower the better), while for targeted attacks the results are evaluated by the attack success rates (the higher the better). Also, we conduct two kinds of experiments: attacking a single node and attacking a group of nodes. For attacking a single node, we add three fake nodes; for attacking a group of 100 nodes, we add 2.5% nodes as the fake nodes. To show unnoticeable perturbations on graph structures, we use not only 95% confidence interval of nodes degrees distribution but also number of connected components of the graphs. To show our features are like the real ones, we use F1 score of a classifier.

We include the following algorithms into comparisons:

- Greedy and Greedy-GAN: our proposed algorithm that attacks GCN by adding fake nodes.
- GradArgmax (Dai et al., 2018): white box GCN attack by only changing links of graphs.[1]
- Nettack (Zügner et al., 2018): state-of-the-art data poisoning GCN attack by changing existing nodes' attributes and links.
- Random: randomly generating features of fake nodes, and randomly adding links between fake nodes and real ones. We add this method as a baseline.

## 5.1. Non-targeted attack

We compare the effectiveness of our methods with random and GradArgmax algorithms (Dai et al., 2018).

For attacking a single node, three fake nodes with an average five edges are added. Table 1 shows that Greedy method achieves lower accuracy than GradArgmax method in both types of normalization. In our experiments, the number of added links is small; thus, the fake nodes' degrees are all fell in 95% confidence interval of power-law distribution. To further illustrate that our method perturbs the graph very slightly, we use number of connected components which is a stronger criterion than node degree distributions. After adding fake nodes and edges, the number of connected components only changes within 3% for both datasets. Also to be noted that we did not perform Greedy-GAN attack for one node, because the numbers of real and fake nodes are so unbalanced and the discriminator will over-fit to the added three nodes.

[1] The original GradArgmax does not work for adding fake nodes, but we modify them to do so by (1) initializing fake nodes with random features and (2) adding the constraints to their optimization to restrict them to modify only those links.

**Table 1**
**Accuracy for non-targeted attacks on a single node with different normalization in Section 3. The last two rows are average number of connected components for each attacking case**

| Dataset | Cora | | Citeseer | |
|---|---|---|---|---|
| Normalization | Row-wise | Symmetric | Row-wise | Symmetric |
| Greedy | 0.09 | 0.08 | 0.11 | 0.09 |
| GradArgmax | 0.10 | 0.56 | 0.15 | 0.55 |
| CC of graph | 78 | 78 | 438 | 438 |
| Avg CC of Greedy | 77.01 | 76.96 | 435.11 | 434.67 |
| Avg CC of GradArgmax | 77.14 | 76.55 | 435.72 | 428.62 |

**Table 2**
**Accuracy of GCN before and after non-targeted attacks for attacking random 100 nodes in each graph. Each graph added 2.5% fake nodes. Each fake nodes is with on average 10 fake links. Note that the final two rows are F1 score of a classifier—lower values indicate added fake nodes are harder to be detected**

| Dataset | Cora | | Citeseer | |
|---|---|---|---|---|
| Normalization | Row-wise | Symmetric | Row-wise | Symmetric |
| Clean | 0.84 | 0.81 | 0.76 | 0.73 |
| Random | 0.77 | 0.79 | 0.68 | 0.69 |
| GradArgmax | 0.53 | 0.61 | 0.55 | 0.47 |
| Greedy | 0.08 | 0.03 | 0.03 | 0.04 |
| Greedy-GAN | 0.05 | 0.09 | 0.07 | 0.03 |
| CC of GradArgmax | 74 | 118 | 439 | 457 |
| CC of Greedy | 71 | 72 | 418 | 401 |
| CC of Greedy-GAN | 72 | 70 | 419 | 420 |
| F1 score for Greedy | 0.73 | 0.67 | 0.64 | 0.77 |
| F1 score for Greedy-GAN | 0.35 | 0.47 | 0.43 | 0.65 |

Next we consider attacking a group of nodes together rather than a single node. In the experiment, we randomly sampled 100 nodes as our attack target and added 2.5% labeled fake nodes with average 10 links per fake node. The results in Table 2 show that our Greedy and Greedy-GAN methods perform better than GradArgmax. Moreover, even directly compared with the results from the Nettack paper (Zügner et al., 2018), which changing edges and features on existing nodes simultaneously, our methods could reach comparable result in Tables 1 and 2.

For the Greedy-GAN method, we choose the parameter $c = 0.1$ in Equation(6), in the way that it is more focusing on attacking than generating real features. We notice that there are some cases that Greedy-GAN performances better than Greedy algorithm in terms of accuracy of GCN, which is a little bit against people intuition. Intuitively, Greedy should always be better than Greedy-GAN, since Greedy-GAN limits the feature spaces. We observe that Greedy method is easier to stuck in local optimum during the optimization procedure (e.g., keep adding/dropping the same edge), while Greedy-GAN has more smoothed optimization curve, because of the added discrimination loss. We will discuss more about how different $c$ influences efficiency of attacking in Section 5.7.

To check whether the attacks change the graph structure significantly, we check the number of connected components. Number of connected components is a widely used parameter in graph theory when describing graphs structures. It is a stronger criterion than node degree distributions.

We observed that GradArgmax attack produces a larger number of components, while both of our methods stay relatively unchanged. Furthermore, we train a classifier using randomly generated nodes and then feed it with fake features generated by Greedy and Greedy-GAN algorithms. The f1 scores of the classifier indicate that it is much harder to differentiate the features of a node between real and fake under attacks by Greedy-GAN, as compared to Greedy. This means that Greedy-GAN attack, although sometimes having lower success rate, can produce fake nodes that are harder to be detected.

## 5.2. Targeted attack

The targeted attack tries to fool the GCN to classify specific nodes to target classes. We perform two different experiments for targeted attacks: (1) attack only a single node and (2) attack a group of 100 nodes on test data. In our experiments, we use the same setting as non-targeted attack for the number of fake nodes and edges.

When attacking only a single node, we add three fake nodes with target labels. Table 3 shows that the success rate of our algorithm is around 80% on different target labels. The addition of three nodes produced no noticeable change in label distributions.

We also conduct targeted attacks on groups of nodes. In Figures 1 and 2, we notice that attacking nodes to certain target classes are more difficult than others. For example, in Figure 2, targeted attack has a lower success rate with a target class 0 (249 nodes) from Citeseer dataset (3312 nodes in 6 classes). Success rate of targeted attack is positively correlated with the percentage of that class in the groundtruth. The reason might be that if there are less nodes belonging to a class, the attacked nodes only have weak linkage to that class, and the classification is more influenced by nodes in other classes. This makes it harder to turn the predicted label to class with few nodes. For fake nodes labels, we do experiments on both randomly and targeted labels. Targeted labeling generally gives better result. Table 4 shows targeted attacks success rate for Greedy and Greedy-GAN algorithms using targeted labeling.

## 5.3. Data poisoning

In industry, GCN is normally applied to longitudinal data. Due to the time-consuming process of training a new network, networks are periodically retrained instead. We thus consider the scenario where the GCN is retrained under the dataset with fake nodes. This is also known as the data poisoning scenario, while the previous discussed case (without retraining) is called test time attack.

Table 5 presents the accuracy after non-targeted data poisoning. The GCN is retrained with learning rate 0.01, 50 epochs after the data have been modified. The results show that both attacking and poisoning can effectively reduce the accuracy of GCN, and the symmetric normalization is more robust under poisoning attacks overall.

For specific node data poisoning, since only three fake nodes are added, we did not change existing edges in the training dataset; thus, the "good" nodes and edges in the training dataset are still dominating. While for a group of nodes it works quite well, we could reach as low as 0.01 accuracy which is as good as

**Table 3**
**Success rate of targeted attacks on a single node for different targeted classes using Greedy attack**

| | | Class index | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Dataset | Normalization | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| Cora | Row-wise | 0.90 | 0.87 | 0.74 | 0.78 | 0.81 | 0.89 | 0.80 |
| Cora | Symmetric | 0.90 | 0.86 | 0.75 | 0.80 | 0.87 | 0.86 | 0.81 |
| Citeseer | Row-wise | 0.80 | 0.81 | 0.80 | 0.83 | 0.75 | 0.80 | |
| Citeseer | Symmetric | 0.83 | 0.80 | 0.79 | 0.83 | 0.73 | 0.81 | |

**Figure 1**
**Row normalized Cora dataset. Right axis: Success rate of targeted attacking a group of 100 nodes with randomly and targeted labeling methods; left axis: Groundtruth and before attacking label distributions, with seven different classes**

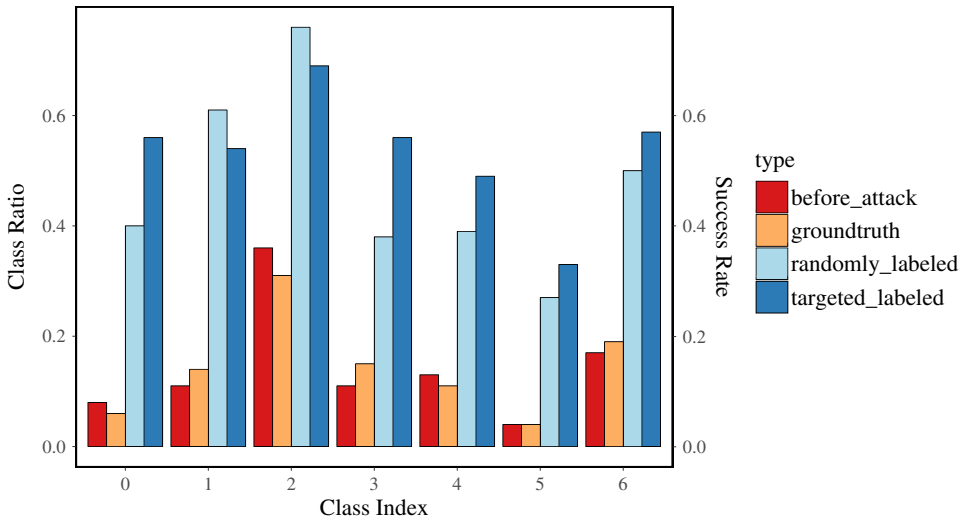


**Figure 2**
**Row normalized Citeseer dataset. Right axis: Success rate of targeted attacking a group of 100 nodes with randomly and targeted labeling methods; left axis: Groundtruth and before attacking label distributions, with six different classes**

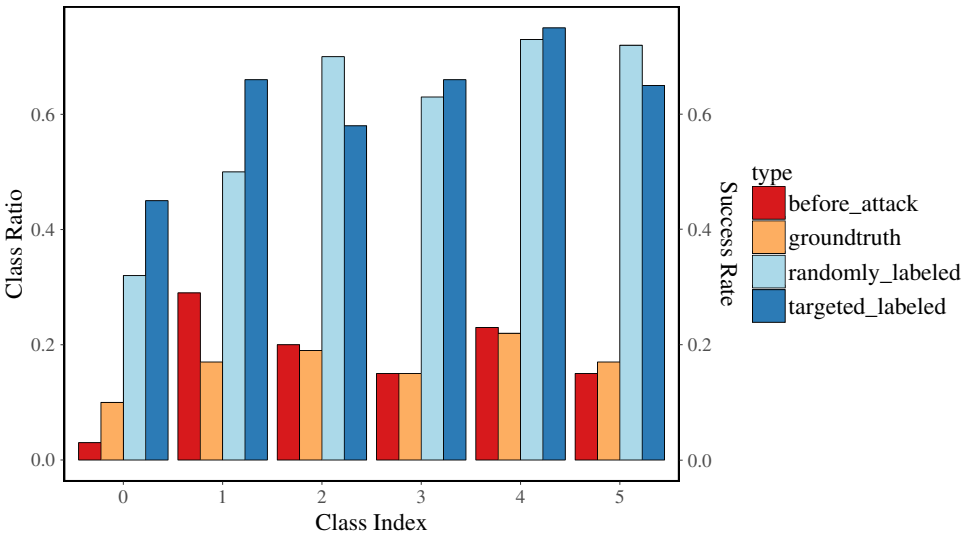

**Table 4**
**Success rate for targeted attacks on the 100 nodes on graph**

| Dataset | Cora | | Citeseer | |
|---|---|---|---|---|
| Normalization | Row-wise | Symmetric | Row-wise | Symmetric |
| Greedy | 0.69 | 0.66 | 0.75 | 0.68 |
| Greedy-GAN | 0.76 | 0.53 | 0.78 | 0.65 |

**Table 5**
**Accuracy after poisoning GCN**

| Dataset | Cora | | Citeseer | |
|---|---|---|---|---|
| Normalization | Row-wise | Symmetric | Row-wise | Symmetric |
| Nettack 100 nodes | 0.01 | 0.04 | 0.04 | 0.07 |
| Greedy one node | 0.08 | 0.09 | 0.14 | 0.20 |
| Greedy 100 nodes | 0.06 | 0.04 | 0.01 | 0.08 |
| Greedy-GAN 100 nodes | 0.01 | 0.11 | 0.03 | 0.03 |

**Table 6**
**A typical case for nodes with different degrees, under Greedy non-targeted attack and poisoning; Citeseer dataset with row normalization, seed = 42**

| Nodes degree | (0,5] | (5,10] | (10,20] | (20, ∞) | Total accuracy |
|---|---|---|---|---|---|
| Clean | 0.78 | 0.83 | 1.0 | 1.0 | 0.80 |
| Greedy attacking | 0.037 | 0.25 | 0.50 | 1.0 | 0.10 |
| Greedy poisoning | 0.025 | 0.417 | 0.667 | 1.0 | 0.12 |
| Greedy-GAN attacking | 0.037 | 0.25 | 0.67 | 1.0 | 0.11 |
| Greedy-GAN poisoning | 0.074 | 0.33 | 0.67 | 1.0 | 0.15 |

the original Nettack (Zügner et al., 2018) by changing features and edges directly and retraining the network until converge, and our result on Citeseer is even better.

## 5.4. Degree of nodes

Previous work (Zügner et al., 2018) found that nodes with smaller degrees are more vulnerable in data poisoning. Our experiments confirm that attacking a group of nodes and single node also follows that rule. Table 6 presents typical nodes accuracy by degree. When adding extra links between fake nodes and real nodes, higher degree nodes are more resistant to the impact of fake nodes. And this works for both attacking and poisoning.

## 5.5. Row-wise vs. symmetric normalization

Although row-wise or symmetric normalization could achieve similar performance in classification, we found their robustnesses vary. For row-wise normalization, when adding a new edge $e_{i,j}$ on graph, the two elements in adjacency matrix $A_{i,j}$ and $A_{j,i}$ change from 0 to 1. Therefore, we could get the corresponding two elements in the normalized matrix $\hat{A}$ changing from 0 to $\hat{A}_{i,j}^{(t+1)} = 1/(d_{ii}+1)$ and $A_{j,i}^{(t+1)} = 1/(d_{jj}+1)$. Also, other nonzero elements $\hat{A}_{i,.}^{(t)}, \hat{A}_{j,.}^{(t)}$ change to

$$\hat{A}_{i,.}^{(t+1)} = d_{ii}^{(t)}/(d_{ii}^{(t)}+1)\hat{A}_{i,.}^{(t)}$$
$$\hat{A}_{j,.}^{(t+1)} = d_{jj}^{(t)}/(d_{jj}^{(t)}+1)\hat{A}_{j,.}^{(t)}$$

For symmetric normalization, origin elements $\hat{A}_{i,j}^{(t)}$ and $\hat{A}_{j,i}^{(t)}$ in the original normalized matrix change from 0 to $1/\sqrt{(d_{ii}^{(t)}+1)(d_{jj}^{(t)}+1)}$ while other nonzero elements $\hat{A}_{i,.}^{(t)}$, $\hat{A}_{j,.}^{(t)}$ change to

$$\hat{A}_{i,.}^{(t+1)} = \hat{A}_{.,i}^{(t+1)} = \sqrt{d_{ii}^{(t)}}/\sqrt{d_{ii}^{(t)}+1}\hat{A}_{i,.}^{(t)}$$
$$\hat{A}_{j,.}^{(t+1)} = \hat{A}_{.,j}^{(t+1)} = \sqrt{d_{jj}^{(t)}}/\sqrt{d_{jj}^{(t)}+1}\hat{A}_{j,.}^{(t)}$$

To sum up, on one hand, the magnitude of change is greater in row normalized elements than in symmetric normalized elements. On the other hand, when adding or dropping an edge in the graph, only 2 rows are affected in the row normalized adjacency matrix. However for symmetric case, it affects both 2 rows and 2 columns.

Therefore, we find that when adding smaller amount of edges, number of affected rows/columns is dominated so that symmetric normalized GCN is more vulnerable. When adding larger amount of edges, the magnitude of change in elements plays a more important role so that the row normalized GCN is more vulnerable. For example, in Table 1, the average number of edges per fake node is 5, and accuracy of symmetric normalized GCN is smaller than row normalized one. While in Table 7 symmetric normalized GCN is more robust in all of the methods, where we add 5% fake nodes to attack the whole test dataset of graph (70% nodes of graph) by adding 10,000 fake edges (around 60–75 edges per fake node). For attacking a group of 100 nodes, the average number of edges per fake nodes is 10, which causes those two effects contribute equally. Therefore, there are no significant differences in terms of robustness.

## 5.6. Number of fake nodes

In this part, we explore how the number of fake nodes influences the efficiency of attacking. We attack a group of 100 nodes with different number of fake nodes. Figure 3 shows how the number of fake nodes influences the classification accuracy. We use use 1%, 1.5%, 2 %, 2.5%, and 3% of fake nodes and assign random labels to the fake nodes. As expected, more fake nodes yield more effective attacks. We notice that the number of added edges between fake nodes and real nodes also plays an important role. The blue line in Figure 3 ranges from 2% to 6% adding same total amount edges (as same as the experiment in Sections 5.1 and 5.2), with different number of fake nodes. This indicates even with very small amount of fake nodes, we can attack the accuracy of GCN to very low, for example, if we have only 1% fake nodes, with average degree of fake node 25, in

**Table 7**
**An extreme case of adding 5% fake nodes and 10,000 edges for non-targeted attacking the whole test dataset (70% nodes)**

| Dataset | Cora | | Citeseer | |
|---|---|---|---|---|
| Normalization | Row-wise | Symmetric | Row-wise | Symmetric |
| Clean | 0.84 | 0.81 | 0.76 | 0.75 |
| Random | 0.34 | 0.42 | 0.49 | 0.51 |
| Greedy | 0.11 | 0.15 | 0.02 | 0.16 |
| Greedy-GAN | 0.11 | 0.25 | 0.07 | 0.17 |

**Figure 3**
**Accuracy varies with different number of fake nodes and different number of added edges, using symmetric Citeseer dataset; red line: Fixed average degree of fake nodes as 10; blue line: Fixed total number of added edges**

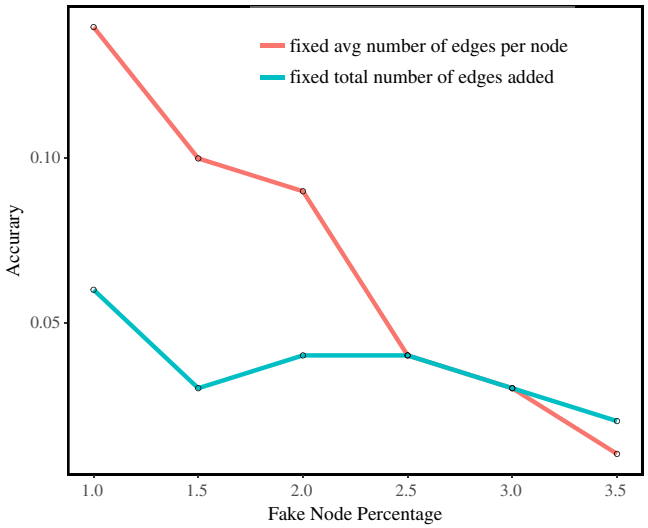


**Figure 4**
**Row normalized Cora dataset: Accuracy with difficult *c* in Algorithm 2**

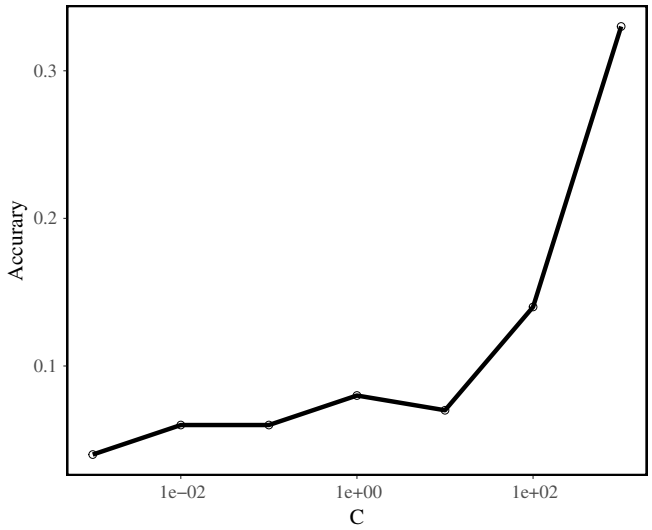


Figure 3, the accuracy is down to 0.06. On the other hand, if we fixed the average degree of fake nodes as 10 (also the same as Sections 5.1 and 5.2 average degree of fake nodes setting), the accuracy varies from 0.14 to 0.01 with the number of fake nodes increasing from 1% to 3%.

## 5.7. Different *c* in Greedy-GAN

We study how different $c$ in Algorithm 2 influences the attack efficiency. In Section 4.1, we claimed that if $c$ is large, the objective function is dominated by the discriminator, so the node features generated will be very close to real ones but with a lower attack successful rate (higher accuracy). If $c$ is greater than 1, we say realness of the nodes is more important, if $c$ is smaller than 1, then the system is more on attacking. In Figure 4, $c$ varies from very small value 0.001 to very large value 1000 increasing by 10 times each time; there is a positive correlation between $c$ and accuracy of GCN. We notice that when $c$ is smaller or equal to 10, the accuracy varies from 0.04 to 0.08, which means even if the system is more on the realness of the fake nodes features ($c > 1$), we could get efficient attacking using Greedy-GAN. Greedy-GAN algorithm could maintain the realness of fake nodes without losing attacking efficiency. Unless $c$ is extremely large, say 1000, the accuracy is 0.33, which is not as efficient as the result in Table 2 in terms of attacking.

## 5.8. Attacking large-scale graphs

When training large-scale graphs, original GCN will not work due to memory bottleneck. Thus, algorithms based on neighborhood sampling come to relieve this issue (Chen et al., 2018; Hamilton et al., 2017). Because there is a neighborhood sampling function in these algorithms, directly adding or deleting edges on the original graphs method becomes less effective than on GCN. The reason is when training GraphSAGE (or other large-scale graph neural networks), the sampled neighborhood could be viewed as dropping edges during training, Dai et al. (2018) show that use dropping edges while training is a cheap method to increase the robustness of GCN. As a result, attacking GraphSAGE (or other large-scale graph neural networks) is more difficult than attacking GCN.

In this section, we use a GCN aggregator for GraphSAGE for nodes prediction. We use Reddits dataset (232,965 nodes, 11,606,919 edges,41 classes and 602 features) as a benchmark. For labeling rate, we use the same setting as GraphSAGE (Hamilton et al., 2017) paper. When considering the attacking single node scenario, for changing edges directly on the existing graphs, the accuracy of GraphSAGE drops from 0.98 to 0.95. For adding fake nodes, the accuracy drops to 0.07. Reddits dataset has continuous feature space; thus, it would be more vulnerable for fake nodes attacks than the discrete feature space datasets (Cora and Citeseer). Adding fake nodes is easily to introduce new features to the target-attacked node in continuous features space; thus, it will lead to accuracy dropping.

## 6. Conclusion

In this paper, we develop two algorithms, Greedy and Greedy-GAN, to attack GCNs by adding fake nodes and without changing any existing edge or feature. Our experimental results show that both algorithms can successfully attack GCNs. To make the attack unnoticeable, we added a discriminator using the Greedy-GAN algorithm to generate features of fake nodes. Furthermore, we explored parameter sensitivities on the degree of nodes; different normalization methods to evaluate the effectiveness of our attacks; different number of fake nodes, and the tradeoff between the realness of features and attacking efficiency. Moreover, we scaled up our attacks to large-scale graph data that use GraphSAGE (Hamilton et al., 2017).

In the future, we would like to explore the defense against adversarial attacks on GCNs as well as on large-scale networks such as GraphSAGE (Hamilton et al., 2017).

## Conflicts of Interest

The authors declare that they have no conflicts of interest to this work.

## References


Arnab, A., Miksik, O., & Torr, P. H. S. (2017). On the robustness of semantic segmentation models to adversarial attacks. *CoRR*, abs/1711.09856.

Carlini, N., & Wagner, D. A. (2017, May 22–26). Towards evaluating the robustness of neural networks. In *2017 IEEE symposium on security and privacy, SP 2017*, San Jose, CA, USA (pp. 39–57).

Chen, H., Zhang, H., Chen, P.-Y., Yi, J., & Hsieh, C.-J. (2017). Show-and-fool: Crafting adversarial examples for neural image captioning. *CoRR*, abs/1712.02051.

Chen, J., Ma, T., & Xiao, C. (2018). Fastgcn: Fast learning with graph convolutional networks via importance sampling. *CoRR*, abs/1801.10247.

Chen, P.-Y., Zhang, H., Sharma, Y., Yi, J., & Hsieh, C.-J. (2017, November 3). ZOO: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models. In *Proceedings of the 10th ACM workshop on artificial intelligence and security, AISec@ CCS 2017*, Dallas, TX, USA (pp. 15–26).

Cheng, M., Le, T., Chen, P.-Y., Yi, J., Zhang, H., & Hsieh, C.-J (2018). Query-efficient hard-label black-box attack: An optimization-based approach. *arXiv preprint arXiv: 1807.04457*.

Cheng, M., Yi, J., Zhang, H., Chen, P.-Y., & Hsieh, C.-J. (2018). Seq2sick: Evaluating the robustness of sequence-to-sequence models with adversarial examples. *CoRR*, abs/1803.01128.

Dai, H., Li, H., Tian, T., Huang, X., Wang, L., Zhu, J., & Song, L. (2018, July 10–15). Adversarial attack on graph structured data. In J. Dy, & A. Krause (Eds.), *Proceedings of the 35th international conference on machine learning*, volume 80. Proceedings of Machine Learning Research, Stockholmsmässan, Stockholm, Sweden (pp. 1123–1132). PMLR.

Defferrard, M., Bresson, X., & Vandergheynst, P. (2016, December 5–10). Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in neural information processing systems 29: Annual conference on neural information processing systems 2016*, Barcelona, Spain (pp. 3837–3845).

Ebrahimi, J., Rao, A., Lowd, D., & Dou, D. (2017). Hotflip: White-box adversarial examples for NLP. *CoRR*, abs/1712.06751.

Eykholt, K., Evtimov, I., Fernandes, E., Li, B., Song, D., Kohno, T., . . . Tramèr, F. (2017). Note on attacking object detectors with adversarial stickers. *CoRR*, abs/1712.08062.

Gao, J., Lanchantin, J., Soffa, M. L., & Qi, Y. (2018, May 24). Black-box generation of adversarial text sequences to evade deep learning classifiers. In *2018 IEEE Security and Privacy Workshops, SP Workshops 2018*, San Francisco, CA, USA (pp. 50–56).

Goodfellow, I. J., Shlens, J., & Szegedy, C. (2014). Explaining and harnessing adversarial examples. *CoRR*, abs/1412.6572.

Grover, A., & Leskovec, J. (2016, August 13–17). node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, San Francisco, CA, USA (pp. 855–864).

Hamilton, W. L., Ying, R., & Leskovec, J. (2017, June). Inductive representation learning on large graphs. *ArXiv e-prints*.

Ilyas, A., Engstrom, L., Athalye, A., & Lin, J. (2018, April). Black-box adversarial attacks with limited queries and information. *ArXiv e-prints*.

Jia, R., & Liang, P. (2017, September 9–11). Adversarial examples for evaluating reading comprehension systems. In *Proceedings of the 2017 conference on empirical methods in natural language processing, EMNLP 2017*, Copenhagen, Denmark (pp. 2021–2031).

Kipf, T. N., & Welling, M. (2016). Semi-supervised classification with graph convolutional networks. *CoRR*, abs/1609.02907.

Kurakin, A., Goodfellow, I., & Bengio, S. (2016, November). Adversarial machine learning at scale. *ArXiv e-prints*.

Li, J., Monroe, W., & Jurafsky, D. (2016). Understanding neural networks through representation erasure. *CoRR*, abs/1612. 08220.

Liang, B., Li, H., Su, M., Bian, P., Li, X., & Shi, W. (2017). Deep text classification can be fooled. *CoRR*, abs/1704.08006.

Lu, J., Sibai, H., & Fabry, E. (2017). Adversarial examples that fool detectors. *CoRR*, abs/1712.02494.

Madry, A., Makelov, A., Schmidt, L., Tsipras, D., & Vladu, A. (2017). Towards deep learning models resistant to adversarial attacks. *CoRR*, abs/1706.06083.

Metzen, J. H., Kumar, M. C., Brox, T., & Fischer, V. (2017, October 22–29). Universal adversarial perturbations against semantic image segmentation. In *IEEE international conference on computer vision, ICCV 2017*, Venice, Italy (pp. 2774–2783).

Papernot, N., McDaniel, P. D., & Goodfellow, I. J. (2016). Transferability in machine learning: from phenomena to black-box attacks using adversarial samples. *CoRR*, abs/ 1605.07277.

Papernot, N., McDaniel, P. D., Swami, A., & Harang, R. E. (2016, November 1–3). Crafting adversarial input sequences for recurrent neural networks. In *2016 IEEE military communications conference, MILCOM 2016*, Baltimore, MD, USA (pp. 49–54).

Pham, T., Tran, T., Phung, D., & Venkatesh, S. (2016, September). Column networks for collective classification. *ArXiv e-prints*.

Rhee, S., Seo, S., & Kim, S. (2017, November). Hybrid approach of relation network and localized graph convolutional filtering for breast cancer subtype classification. *ArXiv e-prints*.

Samanta, S., & Mehta, S. (2017). Towards crafting text adversarial samples. *CoRR*, abs/1707.02812.

Suya, F., Tian, Y., Evans, D., & Papotti, P. (2017). Query-limited black-box attacks to classifiers. *CoRR*, abs/1712.08713.

Tramèr, F., Kurakin, A., Papernot, N., Boneh, D., & McDaniel, P. D. (2017). Ensemble adversarial training: Attacks and defenses. *CoRR*, abs/1705.07204.

Xie, C., Wang, J., Zhang, Z., Zhou, Y., Xie, L., & Yuille, A. L. (2017, October 22–29). Adversarial examples for semantic segmentation and object detection. In *IEEE international conference on computer vision, ICCV 2017*, Venice, Italy (pp. 1378–1387).

Xu, X., Chen, X., Liu, C., Rohrbach, A., Darell, T., & Song, D. (2017). Can you fool AI with adversarial examples on a visual turing test? *CoRR*, abs/1709.08693.

Ying, R., He, R., Chen, K., Eksombatchai, P., Hamilton, W. L., & Leskovec, J. (2018, August 19–23). Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining, KDD 2018*, London, UK (pp. 974–983).

Zhao, Z., Dua, D., & Singh, S. (2017). Generating natural adversarial examples. *CoRR*, abs/1710.11342.

Zügner, D., Akbarnejad, A., & Günnemann, S. (2018, May). Adversarial attacks on neural networks for graph data. *ArXiv e-prints*.


**How to Cite:** Wang, X., Cheng, M., Eaton, J., Hsieh, C.-J., & Wu, S. F. (2022). Fake Node Attacks on Graph Convolutional Networks. *Journal of Computational and Cognitive Engineering* 1(4), 165–173, https://doi.org/10.47852/bonviewJCCE2202321