Anonymized GCN: A Novel Robust Graph Embedding Method via Hiding Node Position in Noise

Ao Liu

College of cybersecurity, Sichuan University, 610065, Chengdu
liuao@stu.scu.edu.cn

Abstract. Graph convolution network (GCN) have achieved state-of-the-art performance in the task of node prediction in the graph structure. However, with the gradual various of graph attack methods, there are lack of research on the robustness of GCN. In this paper, we prove the reason why GCN is vulnerable to attack: only training another GCN model can find the vulnerability of the target GCN model. To solve that, we propose a GCN model which is robust to attacks. By hiding the node's position in the Gaussian noise, the attacker will not be able to modify the connection information of the graph node, thus immune to the attack. Considering attackers usually modify the connection to interfere the prediction results of the target node, so, by hiding the connection of the graph in the noise through adversarial training, the accurate node prediction can be completed only by the node number rather than its specific position in the graph, thus let the nodes in the graph are no longer related to the graph itself, that is to say, make the node anonymous. Specifically, we first demonstrated the key to determine the embedding of a specific node: the row corresponding to the node of the eigenmatrix of the Laplace matrix, by target it as the output of the generator, we take the corresponding noise as input. The generator will try to find the correct position of the node in the graph. Then the encoder and decoder are spliced both in discriminator, so that after adversarial training, the generator and discriminator can cooperate to complete the node prediction. Finally, All node positions can generated by noise at the same time, that is to say, the generator will hides all the connection information of the graph structure. The evaluation shows that we only need to obtain the initial features and node numbers of the nodes to complete the node prediction, and the accuracy did not decrease, but increased by 0.0293.

Keywords: Graph Convolutional Network · Adversarial Attacks · Generative Adversarial Networks · Robust Deep Learning.

1 Introduction

Graphs are ubiquitous in the real world, it is the core for many high impact applications ranging from the analysis of social networks, over gene interaction networks, to interlinked document collections. In many tasks related to graph

structure, node classification has always been a hot issue, which can be described as: predicting the labels of unknown nodes based on a small number of labeled known nodes. The usual task flow is to first use graph encoding (or embedding) methods such as GCN to obtain the embedding of each node, and then decode the node embedding to obtain its label. Through the research of the researchers, various graph encoding and decoding methods were proposed, which improved the task of node prediction from all angles. In order to solve this problem, we design a graph neural network immune to all existing attacks, so that all nodes in the graph data are no longer affected by the perturbation designed for GCN.

However, with the deepening of research, the vulnerability of node prediction has been gradually explored, even only slight deliberate perturbations of the nodes' features or the graph structure can lead to completely wrong predictions, an obvious reason is that the embedding of nodes is significantly affected by their position (represents the connection structure of the entire graph and the position of the nodes in it) in the graph. Specifically, after selecting the target node, the attacker modifies the information directly or indirectly related to the node in the graph structure, such as adding / deleting edges. Due to the discrete nature of the graph, it is always possible to find a minimal action that significantly disrupts the final prediction, to complete the attack on node prediction. Since GCN model itself is rarely open source, most users will retrain the model based on the existing graph data. Therefore, the attacker can adjust the perturbation items repeatedly according to the trained model feedback, and add unobtrusive perturbation to the graph data to accurately manipulate the classification results of the target node. It can be seen that the causes of Vulnerability of GCN are:

The GCN model is very sensitive to the change of the position of nodes.

Therefore, the purpose of this paper is to reduce the sensitivity of GCN to location information. Specificly, if we hide the position of the node, the attacker will not be able to obtain information about the target node, that is, it will not be able to make corresponding perturbation to the node. Therefore, we propose an adversarial generation method for graph-connected structures, which only needs to give the target node number and initial features to obtain the accurate embedded features of the node. In this paper we will provide the following:

1. nonymized node. Through analysis and experiments, we have demonstrated the key to determining the location of nodes, and designed a generator that generates node locations from noise. The generator can receive the node number without prior information of the graph structure, and accurately confirm the position of the node, instead of directly obtaining the location from the graph structure. It realizes that only the node number and the node initial features are required to accurately encode. Since the position of a node determines the role of the node in the graph, we provide a encoding method that does not require obtaining its position, that is, a method to make the node anonymized.

2. Accurate node classification. We design the method of adversarial training so that the node can still be accurately embedded under anonymized. Specifically, we spliced the encoder and decoder in the discriminator, so that the generator can undertake the functions of both, and then perform accurate node classification.

3. Completely anonymous graph. After adversarial training, the positions of all nodes can be anonymous at the same time, that is to say, the graph can only contain node features, and all connection relationships are hidden in the generator.

Finally, given the number of the target nodes, we can complete the anonymized node classification, which significantly improves the robustness of the node classification.

2 Related Work

Attack. In 2018, Dai et al [1] and Zügner [2] first proposed adversarial attacks on graph structures, after which a large number of graph attack methods were proposed. Specific to the task of node prediction, Chang [3] attacked various kinds of graph embedding model with black-box driven, Aleksandar [4] provide the first adversarial vulnerability analysis on the widely used family of methods based on random walks, derive efficient adversarial perturbations that poison the network structure. Wang propose a threat model to characterize the attack surface of a collective classification method, target on adversarial collective classification. Basically, all attack types are based on the modified graph structure targeted by this article.

Defense without GAN. Tang [6] investigate a novel problem of improving the robustness of GNNs against poisoning attacks by exploring clean graphs, create supervised knowledge to train the ability to detect adversarial edges so that the robustness of GNNs is elevated. Jin [7] use the new operator in replacement of the classical Laplacian to construct an architecture with improved spectral robustness, expressivity and interpretability. Zügner [8] propose the first method for certifiable (non-)robustness of graph convolutional networks with respect to perturbations of the node attributes.

Defense with GAN. As in this paper, some defense methods also use adversarial training to enhance the robustness of the model. Deng [9] present batch virtual adversarial training (BVAT), a novel regularization method for graph convolutional networks (GCNs). By feeding the model with perturbing embeddings, the robustness of the model is enhanced by them, but this method trains a full-stack robust model for the encoder and decoder at the same time, without discussing the nature of the graph structure's vulnerability and solving it. Wang [10] first investigate the latent vulnerabilities in every layer of GNNs and propose corresponding strategies including dual-stage aggregation and bottleneck perceptron. Then, to cope with the scarcity of training data, they propose an adversarial contrastive learning method to train the GNN in a conditional

GAN manner by leveraging the high-level graph representation. But from a certain point of view, they still use the method based on node perturbation for adversarial training. This method is essentially a kind of "perturbation" learning, and uses adversarial training to adapt the model to various custom perturbations. This is a kind of node-based adversarial training, which requires a large number of specific perturbation to be customized, and the potential structure of the entire graph cannot be explored.

Graph GAN without considering attack and defense. Wang [11] Combining two methods of graph representation learning as generators and discriminators, respectively, to improve the accuracy of both in adversarial training. However, this method does not discuss the potential vulnerability of the graph structure, nor does it attempt to accurately perturb the final classification, and cannot be directly applied to the graph defense method. Ding [12]'s perspective is extended to the regional structure of the entire graph, but the task goal is still to obtain an accurate graph representation, and the generated fake samples cannot match the various perturbation that are carefully designed for the model vulnerabilities.

By summarizing the related work, it can be seen that there are no robust graph embedding methods for the purpose of hiding the "position" of the node, and the existing robust model design methods that use adversarial training as a means cannot solve the vulnerability of the graph structure from the root cause.

3 Discussion on graph convolution

We hope to re-examine the detailed process of graph convolution in this section, and find out the problems we want to solve.

3.1 Graph convolution encoding and decoding

let $\mathcal{G} = (X, \mathcal{E})$ be a graph, where $X = \begin{pmatrix} f(1) \\ \cdots \\ f(N) \end{pmatrix}$ is the set of the features of N nodes, while $f(i)$ denotes the feature vector of node i and $\mathcal{E}$ denotes the set of edges. The edges describe the relations between nodes and can also be represented by an adjacency matrix $\mathrm{A} \sim \mathbb{R}^{N \times N}$ and degree matrix $\mathrm{D} \sim \mathbb{R}^{N \times N}$. According to a, the Laplacian matrix of A graph can be obtained. The one-hot code of all node categories on $\mathcal{G}$ is $\mathcal{Y}$. The process of supervised training of graph convolution model on labeled graph dataset is as follows:

Encode. The graph convolution model maps x into the embedding space $X^E = \begin{pmatrix} f^e(1) \\ \cdots \\ f^e(N) \end{pmatrix}$ through the trainable parameter θ^{ENC}, while $f^e(i)$ denotes the embedding feature vector of node i. Let $\mathrm{ENC}_{\mathcal{G} \to X^E}$ denotes the encoder, the encode process expressed as:

$$\begin{aligned} X^E &= \underset{\mathcal{G}\to X^E}{\mathbf{ENC}}\left(X, \mathcal{E}; \theta^{ENC}\right) \\ or\ X^E &= \underset{\mathcal{G}\to X^E}{\mathbf{ENC}}\left(\mathcal{G}; \theta^{ENC}\right) \end{aligned} \tag{1}$$

Decode. Then the model decode X^E to label space $\mathcal{L} \sim \mathbb{R}^{\iota\times 1}$ through the decoder $\mathrm{DEC}_{X^E\to\mathcal{L}}$ (In general, it's GCN layers) through the trainable parameter θ^{DEX}, ι is the number of labeled categories of $\mathcal{G}$ and $\mathcal{L}$ is in the form of one hot encode.

Train. According to the label of known nodes, the parameters of encoder and decoder are trained end-to-end to complete the accurate node prediction task. Therefore, the supervised learning process of GCN is:

$$\underset{\theta^{ENC},\theta^{DEC}}{\arg\min} \frac{1}{N}\sum \mathrm{Loss}_{\mathrm{DEC}}\left(\mathcal{Y}, \underset{X^E\to\mathcal{L}}{\mathbf{DEC}}\left(\underset{\mathcal{G}\to X^E}{\mathbf{ENC}}\left(\mathcal{G}; \theta^{ENC}\right); \theta^{DEC}\right)\right) \tag{2}$$

Let $\underset{X^E\to\mathcal{L}}{\mathbf{DEC}}\left(\underset{\mathcal{G}\to X^E}{\mathbf{ENC}}\left(\mathcal{G}; \theta^{ENC}\right); \theta^{DEC}\right) = \mathbf{GCN}\left(\mathcal{G}; \theta^{GCN}\right)$ denote the process of encoding and decoding, the calculation method of graph convolution model parameters θ^{GCN} becomes:

$$\theta^{GCN} = \underset{\theta}{\arg\min} \frac{1}{N}\sum \mathrm{Loss}_{\mathrm{DEC}}\left(\mathcal{Y}, \mathbf{GCN}\left(\mathcal{G}; \theta\right)\right) \tag{3}$$

Formula 3 is the unified form of current graph convolution training methods. Based on it, we will discuss the different forms of spectral [13] and spatial domain[14] graph convolution methods.

3.2 Encoding and decoding of spectral/spatial graph convolution methods

In this section, the existing graph convolution is applied to the encoding-decoding process to support the argument in Section 3.1. Firstly, the spectral domain graph convolution process is introduced.

Set the Laplacian matrix of $\mathcal{G}$ is Δ, the matrix eigenvalues of Δ is expressed as $U = \begin{pmatrix} u_1(1) & \cdots & u_N(1) \\ \vdots & \ddots & \vdots \\ u_1(N) & \cdots & u_N(N) \end{pmatrix}$, $u_l = \begin{pmatrix} u_l(1) \\ \cdots \\ u_l(N) \end{pmatrix}$ represents the l-th eigenvector, $u(l) = \{u_1(l), \ldots, u_N(l)$ represents row vector consisting of the values of all eigenvectors at position l. For convenience, we use "node n" to mean "node with the number of n".

In the process of encoding the node feature X into X^E, first obtain the transpose matrix U^T of the matrix eigenvalues U, convert the node features X

to the spectral domain through the U^T, then complete the convolution through the trainable diagonal matrix $g_\theta(\Lambda) = \begin{pmatrix} \theta_1 & & \\ & \ddots & \\ & & \theta_N \end{pmatrix}$, and finally use U Convert from the spectral domain to the final node representation. The specific process is as formula 4.

$$\underset{X^E \to \mathcal{L}}{\widetilde{\mathbf{ENC}}} \left(X, \mathcal{E}, g_\theta(\Lambda)\right) = \sigma\left(U g_\theta(\Lambda) U^T X\right) \tag{4}$$

In this paper, the decoder of spectral domain graph convolution will be designed as a fully connected neural network, that is to say:

$$\underset{X^E \to \mathcal{L}}{\widetilde{\mathbf{DEC}}} \left(X^E; \theta^{DEC}\right) = X^E W^{DEC},\ W^{DEC} \sim \mathbb{R}^{N \times N} \tag{5}$$

In addition, the spatial graph convolution [14] is also implemented through the above process:

$$\begin{aligned} \underset{\mathcal{G} \to X^E}{\widehat{\mathbf{ENC}}} \left(X, \mathcal{E}; \theta^{ENC}\right) &= \mathrm{ReLU}\left(\hat{A} X W^{(0)}\right) \\ \underset{X^E \to \mathcal{L}}{\widehat{\mathbf{DEC}}} \left(X^E; \theta^{DEC}\right) &= \mathrm{softmax}\left(\hat{A} X^E W^{(1)}\right) \\ \underset{W^{(0)}, W^{(1)}}{\arg\min} \frac{1}{N} \sum \mathrm{Loss}_{\mathrm{DEC}} &\left(\mathcal{Y}, \mathcal{L}\right) \end{aligned} \tag{6}$$

$\hat{A}$ is the Symmetric normalized Laplacian matrix [14] calculated according to $\mathcal{E}$, $W^{(0)}$ and $W^{(1)}$ is trainable parameter matrix.

3.3 Reasons for the vulnerability of graph convolution models

The current graph attack methods make the decoding of the target node change the most by traversing the perturbations ϱ on the graph $\mathcal{G}$: $\mathcal{G}_{poisoned} = \varrho\left(\mathcal{G}; \theta^\varrho\right)$, θ^ϱ denotes the specific perturbation, since this article only discusses the attack caused by modifying the connection, so $\varrho\left(\mathcal{G}; \theta^\varrho\right) = \varrho\left(\mathcal{E}; \theta^\varrho\right)$. After training the parameters according to formula 3 on the clean graph, the attacker finds the specific perturb method by calculating formula 7

$$\theta^\varrho = \arg\min_\theta \frac{1}{N} \sum \mathrm{Loss}_{\mathrm{DEC}} \left(\mathcal{Y}, \mathbf{GCN}\left(\varrho\left(\mathcal{E}; \theta\right); \theta^{GCN}\right)\right) \tag{7}$$

Then, if the attacker uses the poison attack [16,17,18], the victim will retrains the infected GCN with perturbation parameters $\hat{\theta}^{GCN}$ and classifies the target nodes to the wrong label space $\hat{\mathcal{L}}$:

$$\begin{aligned} \hat{\theta}^{GCN} = \arg\min_\theta \frac{1}{N} \sum \mathrm{Loss}_{\mathrm{DEC}} &\left(\mathcal{Y}, \mathbf{GCN}\left(\mathcal{G}_{poisoned}; \theta\right)\right) \\ \hat{\mathcal{L}} &= \mathbf{GCN}\left(\mathcal{G}; \hat{\theta}^{GCN}\right) \end{aligned} \tag{8}$$

if the attacker uses the evasion attack [19,20], the victim uses the model trained on the clean graph to misclassify the target nodes:

$$\hat{\mathcal{L}} = \mathbf{GCN}\left(\mathcal{G}_{poisoned}; \theta^{GCN}\right) \tag{9}$$

Let's look at how attackers take advantage of the vulnerability of GCN. From formula 8 and 9, we can see that $\theta^{GCN} = \left(\theta^{ENC}, \theta^{DEC}\right)$, the parameter of GCN, is the key to complete the attack, because $\mathcal{G} \xrightarrow{\theta^{\mathcal{P}}} \mathcal{G}_{poisoned}$, we simplify the behavior of exploiting GCN vulnerability in formula 7 to:

$$\theta^{\varrho} = \mathbf{ATTACK}\left(\theta^{ENC}, \theta^{DEC}, \mathcal{E}\right) \tag{10}$$

In formula [10], attackers can explore all the distribution possibilities of graph in X^E by traversing, so the encoder in GCN is very fragile, meanwhile the decoder is not the source of the problem, because the output of decoder only provides more accurate guidance for attackers to manipulate enc. For example, in reference [15], the attacker designed the best moving path in the embedding space by attacking decoder. Or, according to unsupervised learning, the moving scheme of the target node can designed in the embedding space by distance and similarity between samples, and realized by traversing graph perturbation. Therefore, further, the attacker's attack mode becomes more intuitive:

$$\theta^{\varrho} = \underset{intuitive}{\mathbf{ATTACK}}\left(\theta^{ENC}, \mathcal{E}\right) \tag{11}$$

As can be seen here, the attacker can successfully attack only by obtaining the encoder parameters θ^{ENC} and $\mathcal{E}$. Next, we will prove that attackers can manipulate output of encoder precisely by manipulating $\mathcal{E}$.

No matter what form of graph convolution is, for node κ, the transfer process between layers can be expressed as follows:

$$f^e(\kappa) = \mathbb{L}\left(\mathrm{D}, \mathrm{A}\right)_{\kappa} XW_1 \tag{12}$$

$\mathbb{L}_{\kappa}$ denotes the κ-th row of different forms of Laplace matrix. Δ_{κ} is the κ-th row of Laplacian matrix of $\mathcal{G}$, W_1 is parameters trained on a clean graph. The attacker's target is: Given any target direction $\mathcal{Q}(\kappa)$ in X^E, urge $f^e(\kappa)$ moves to the target along the direction $\mathcal{Q}(\kappa)$. We denote the fake position of all nodes after modification in X^E as $\mathcal{Q}'(\kappa)$(Generally, only the location of the target node κ will be modified in a single attack task), that is, $\mathcal{Q}'(\kappa) = \{f(1), \cdots, f^e(\kappa) + \mathcal{Q}(\kappa), \cdots, f(N)\}$. Next, imagine that node κ with label $\mathcal{Q}'(\kappa)$ is connected to a full-connect graph $\bar{\mathcal{G}}$ which each node has closed loops. The diagonal matrix and adjacency matrix of $\bar{\mathcal{G}}$ are respectively $\bar{\mathrm{D}}$ and $\bar{\mathrm{A}}$, and the node attribute sets of $\mathcal{G}$ and $\bar{\mathcal{G}}$ are same. In order for κ to be classified as $f^e(\kappa) + \mathcal{Q}(\kappa)$, the attacker only needs to get a modified scheme matrix $\mathcal{H}$, let $\mathcal{H}$ act on $\bar{\mathrm{A}}$, achieve:

$$\mathcal{Q}'(\kappa) = \mathbb{L}\left(\mathrm{D}, \mathcal{H} \circ \bar{\mathrm{A}}\right) XW_2 \tag{13}$$

$\circ$ represents Hadamard Product. At the same time, the attacker will not modify the connection directly connected with κ for concealment, so $\mathcal{H}$ no longer modifies line κ of A, Formula 13 becomes:

$$\mathcal{Q}'(\kappa) = \mathbb{L}\left(\mathrm{D}, \begin{pmatrix} \mathcal{H}_{1\cdot} \circ \bar{\mathrm{A}}_{1\cdot} \\ \cdots \\ \mathrm{A}_{\kappa\cdot} \\ \cdots \\ \mathcal{H}_{N\cdot} \circ \bar{\mathrm{A}}_{N\cdot} \end{pmatrix}\right) XW_2 = \mathbb{L}\left(\mathrm{D}, \underset{\kappa}{\mathcal{H}} \circ \bar{\mathrm{A}}\right) XW_2 \tag{14}$$

by set $\underset{attack}{\mathbf{ENC}}(x) = xW_2$, $\underset{attack}{\mathbf{DEC}}(x) = \mathbb{L}\left(\mathrm{D}, \underset{\kappa}{\mathcal{H}} \circ \bar{\mathrm{A}}\right) x$, its process can be expressed as follows:

$$W_2 = \arg\min_{W} \frac{1}{N} \sum \mathrm{Loss}\left(\mathcal{Q}'(\kappa), \underset{attack}{\mathbf{DEC}}\left(\underset{attack}{\mathbf{ENC}}\left(\bar{\mathcal{G}}; W_2\right); \mathcal{H}\right)\right) \tag{15}$$

Formula 15 is computationally universal (Graph neural networks with sufficient parameters are always computationally universal [21]). As can be seen, formula 15 and formula 2 are essentially the same, since formula 2 is the training method of GCN, so, **if the attacker wants to manipulate the location of the target node in the embedded space, it can be realized by another GCN training task - which is easy to implement.**

Therefore, by manipulating $\mathcal{E}$ to bypass the encoder, the essence of the attack becomes:

$$\theta^{\varrho} = \underset{essence}{\mathbf{ATTACK}}(\mathcal{E}) \tag{16}$$

Through the above analysis, we explain why the attack on the node prediction task can be realized by perturbing the connection structure of the graph.

3.4 Problem statement and our targets

From the above discussion, we can know that graph convolution neural network can accurately classify nodes by simultaneously training encoder and decoder end-to-end. In this process, the encoder is responsible for finding the appropriate embedding position for the node. When the graph data is poisoned, in fact, the attacker has found a way to bypass the encoder: traversal. Because of the connectivity of the graph, traversal can always make the target node move to any custom direction in the embedded space (even if the moving direction is not accurate enough, at least the closest moving direction can be found in the current dataset at least), which is the root cause of node misclassification. At the same time, in connection prediction tasks, the attacker can manipulate the target node to "destroy" the target connection in the embedding space according to the gradient of the decoder [15]. Therefore, attackers can easily manipulate the structure of graph convolution neural network in embedding space, which is the root cause of the vulnerability of graph convolution neural network.The problem becomes:

Problem Statement. For node n, by modifying $\mathcal{E}$, the $f^e(n)$ can be significantly affected by the attacker. At the same time, $\mathcal{E}$ is an important reference among $f(n) \to f^e(n)$, **we need to ensure the correctness of** $f(n) \to f^e(n)$ **meanwhile protect** $\mathcal{E}$ **from attack**.

In order to solve this problem, we no longer embed the nodes in the explicit embedding space, but seal all the connection information of the nodes into a generator with Gaussian noise as the input, so as to ensure the security of the embedded space, thus blocking the attacker from directly traversing the path of the embedded space, and solving the vulnerability of the graph convolution neural network. Realize the encoding mode as shown in Figure 1

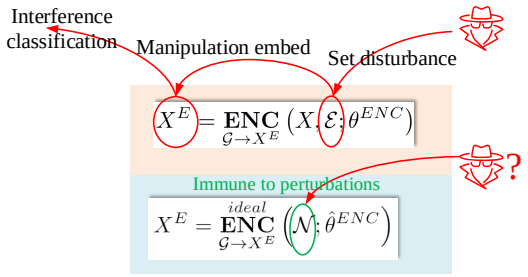

Fig. 1. Methods to solve the vulnerability of GCN

As shown in Figure 1, the node number set $\mathcal{N} = \{1, \ldots, N\}$ is used to replace the specific connection information $\mathcal{E}$, so as to prevent attackers from manipulating the embedded space at will. Specifically, we divided the task into three target:

Target 1. Find out how the connection structure $\mathcal{E}$ affects the embedding process when node n looks for its position in the embedded space through graph convolution.

Target 2. Design a generation method G of node location information $\mathcal{E}_{fake} = G(\mathcal{G})$, create a generated graph $\mathcal{G}_{fake} = (X, \mathcal{E}_{fake})$, Let $\mathbf{GCN}(\mathcal{G}_{fake}) \sim \mathbf{GCN}(\mathcal{G})$.

Target 3. Design an end-to-end training method for the proposed model, and accurately predict the label of the unknown node according to the label of the known node.

4 AN-GCN: Generators, discriminators and optimization methods

In formula 4, U contains the information of the edges in the graph, we plan to replace it with a matrix generated from Gaussian noise, so that the edges

in the graph are no longer restricted by the existing topology, which makes the attacks impossible delete / add edges (because the U recording the edge information has been replaced by the generated matrix), and thus making all nodes of the entire graph anonymous. Specifically, we will optimize the generative model through adversarial training. In this section, we will introduce the structure of the generator and discriminator, then give the method of model optimization.

4.1 Generator

Set node feature representation in the spectral domain $X^{spec} = g_\theta(\lambda)U^T X$, that is, $g_\theta(\lambda)U^T$ first integrate the features of all nodes X to obtain the spectral domain convolution features X^{spec}, which still contains the features of all nodes. After that, through

$$\begin{pmatrix} f(1)^e \\ \dots \\ f(1)^e \end{pmatrix} = \sigma \begin{pmatrix} u(1) \\ \cdots \\ u(N) \end{pmatrix} X^{spec} \tag{17}$$

the embedding of each node is obtained from X^{spec}. The key of this process is, each node through $u(n) = \left(u_1^{(n)}, \dots, u_N^{(n)}\right)$ to accurately obtain the final feature embedding belonging to the simple node from the X^{spec} that contains all the features. In other words, $u(n)$ is the key to locate specific nodes. This phenomenon is explained by Theorem 1

Theorem 1. *In the process of GCN, $u(n)$ affects the position of node n more than $u(\xi|\xi \neq n)$*

Proof. According formula 4, we get:

$$\begin{pmatrix} f^e(1) \\ \dots \\ f^e(N) \end{pmatrix} = \begin{pmatrix} u(1) \\ \dots \\ u(N) \end{pmatrix} \begin{pmatrix} \theta_1 \left(\sum_{i=1}^{v-1} u_1(i)f(i) + \sum_{j=v+1}^{N} u_1(j)f(j)\right) \\ \dots \\ \theta_N \left(\sum_{i=1}^{v-1} u_N(i)f(i) + \sum_{j=v+1}^{N} u_N(j)f(j)\right) \end{pmatrix} \tag{18}$$

so, to get the embedded attribute

$$f^e(v) = \sum_{l=1}^{N} \theta_l u_l^2(v) f(v) \left(\sum_{i=1}^{v-1} u_l(i)f(i) + \sum_{j=v+1}^{N} u_l(j)f(j)\right) \tag{19}$$

of each node, It can be seen initially that the power of $u(v)$ is greater than that of $u(\xi|\xi \neq n)$. Further, we explore the impact of $u(n)$ on node positioning on existing graphs, set $\left(\sum_{i=1}^{v-1} u_l(i)f(i) + \sum_{j=v+1}^{N} u_l(j)f(j)\right) = \rho(v, l, U_v)$, where U_v stands for U without $u(v)$, used to express the influence of $u(\xi|\xi \neq n)$ on the embedding of node v. Next, we reduce the value of $u(\xi|\xi \neq n)$ by a factor of $\mathcal{P}$, that is, get the matrix eigenvalues $U^{(\mathcal{P},v)} = \{u_i^{(\mathcal{P},v)} | i \in (1, N)\}, u_i^{(\mathcal{P},v)} =$

$\begin{cases} u(v), i = v \\ \mathcal{P}u(v), i \neq v \end{cases}$, obtain $U^{(\mathcal{P},v)}$ as the matrix eigenvalues, use the pre-trained $g_\theta(\Lambda)$ for GCN. In order to explore the position of the $\mathcal{P}$ effect (used to determine $u(v)$ and $u(\xi|\xi \neq v)$) influence on the embedding accuracy of node v, move the position of the $\mathcal{P}$ effect to the c_v neighbors $\{\gamma\}$ adjacent to v (Select nodes directly connected to v according to the weight of edges from large to small, the order from largest to smallest is $v^{close} = \{v_1^{close}, \ldots, v_{c_v}^{close}\}$. The final equation for the embedding of node v act by $\mathcal{P}$ is $f^{e,\mathcal{P}}(v) = \sum_{l=1}^{N} \theta_l u_l^2(v) f(v) \mathcal{P} \rho(v, l, U^{(\mathcal{P},v)})$. Using Chebyshev polynomial as the convolution kernel, and cora as the test data set, set $c_v = 14$, $\mathcal{P} = 1 - \frac{k}{100}, k \in \{1, \ldots, 50\}$. The result is as fig 2. It can be seen that only when $\mathcal{P}$ acts on the target point v, the embedding accuracy will suddenly drop(Measured by the Euclidean distance between $f^e(v)$ and $f^{e,\mathcal{P}}(v)$), expressed as $d_v^{\mathcal{P}}$, and the other conditions will remain stable, in other words, $u(v)$ has a much greater impact on the embedding of node v than $u(\xi|\xi \neq v)$.

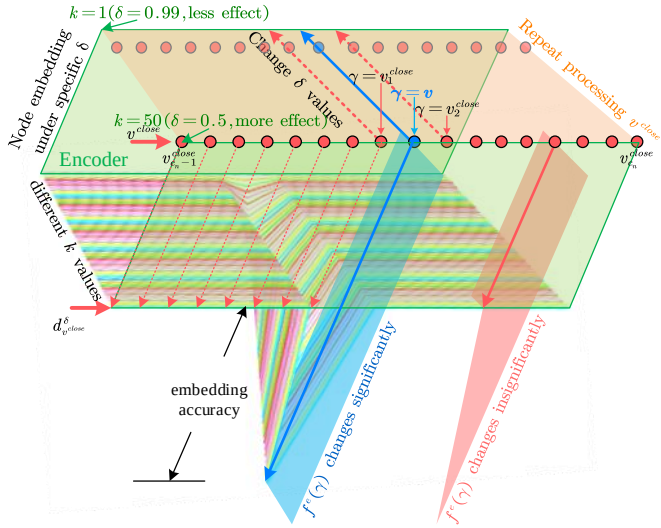

Fig. 2. The effect of $\mathcal{P}$ on the accuracy of node embedding when acting on different positions

Furthermore, we continue to explore the reverse effect of nodes on $u(n)$ to prove the effect of node position on $u(n)$. Delete a node τ in the graph $\mathcal{G}$ to obtain the deleted graph $\mathcal{G}_\tau^{(d)}$. Calculate the laplacian matrix $L_\tau^{(d)} \sim \mathbb{R}^{(N-1)\times(N-1)}$ and matrix eigenvalues $U_\tau^{(d)} = \{u^{(d)}(1), \ldots, u^{(d)}(N-1)\} \sim \mathbb{R}^{(N-1)\times(N-1)}$. To keep $\mathcal{G}_\tau^{(d)}$ will connected, All edges connected to τ will be re-connected by traversal. Specifically, we stipulate that ω_{ij} is the weight of connecting nodes i and j in $\mathcal{G}$, and $\omega_{ij}^{(d)}$ corresponds to graph $\mathcal{G}_\tau^{(d)}$. The calculation method of all edge weights

in $\mathcal{G}_\tau^{(d)}$ is:

$$\omega_{ij}^{(d)} = \begin{cases} \omega_{ij}, \omega_{\tau i} = 0 \text{ or } \omega_{\tau j} = 0 \\ \omega_{ij} + \frac{\omega_{\tau i} + \omega_{\tau j}}{2}, \omega_{\tau i} \neq 0, \omega_{\tau j} \neq 0 \end{cases} \tag{20}$$

After obtaining the fully connected $\mathcal{G}_\tau^{(d)}$, we recalculate corresponding $U_\tau^{(d)}$, Obtain all β-order neighbors of τ: $\text{Nabor}_\beta(\tau) = \{\tau_\beta(1), \tau_\beta(2), \ldots\}$, $|\text{Nabor}_\beta(\tau)| = b$. For all the nodes to be deleted in each order, find $u(\text{Nabor}_\beta(\tau))$ corresponding to the all position. Calculate the change of each $u(\tau_\beta(\cdot))$ with the corresponding node in $\mathcal{G}$, that is, $u^{(d)}(\tau_\beta(\cdot))$. The quantitative representation of the change between the two is as follows, u_i represents the i-th item of the vector:

$$\mathrm{C}\left(u\left(\tau_\beta(\cdot)\right), u^{(d)}\left(\tau_\beta(\cdot)\right)\right) = \sum_{i=1}^{N-1} \log\left|u\left(\tau_\beta(\cdot)\right)_i\right|^2 - \log\left|u^{(d)}\left(\tau_\beta(\cdot)\right)_i\right|^2 \tag{21}$$

Replace different betas and calculate C, the result is as shown in figure 3 Since the node number will change after the node is deleted, because we previously stated that the letter expression of the node is used instead of the node number, so in this article, the expression of the node will not change after deleting a node. We select the first 500 nodes according to the number of connections from large to small. It can be seen from the figure 3 that after deleting node τ, the overall difference in the change of $u\left(\text{Nabor}_\beta(\tau)\right)$ for each order neighbor is large, and the u of the first-order neighbor $u\left(\text{Nabor}_1(\tau)\right)$ has the largest change (the vertical axis is $-C$). In other words, after the node τ is deleted, the $u(\cdot)$ corresponding to its first-order neighbor the $u(\tau_1(\cdot))$ has changed significantly, while the $u(\tau_2(\cdot))$ and $u(\tau_3(\cdot))$ has changed less and showed a decreasing trend. Since deleting node τ significantly affects the position of its first-order neighbors, as the order increases, the degree of influence gradually decreases, so the change of its $u(\tau_\beta(\cdot))$ also gradually decreases. In other words, the position of node τ is inseparable from $u(\tau)$, but has a small relationship with $u(\xi|\xi \neq \tau)$.

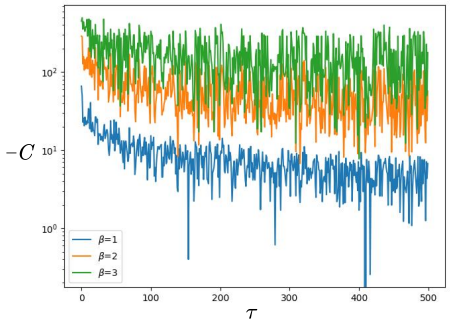

Fig. 3. After deleting τ, the change of neighbor $u\left(\text{Nabor}_\beta(\tau)\right)$ of different orders β of τ

When n is completely generated by noise, the specific points will be hidden before the task requirements are clarified, thereby making the graph attack lose its target. So we make $u(n)$ as the generation target, and the output of the generator named $u^G(n)$, which tries to approximate the underlying true $u(n)$ distribution.

In order to enable the generator to locate a specific point, the input noise of the generator will be constrained by the position of the target point, namely Staggered Gaussian distribution, the purpose is to make the noise not only satisfy the Gaussian distribution, but also do not coincide with each other, densely distributed on the number axis.

Theorem 2. *(Staggered Gaussian distribution). Given a minimum probability ε, N Gaussian distributions centered on $x = 0$ satisfy* $\mathrm{P}(x, n) \sim \mathrm{Norm}(2\sigma(2n - N - 1)\sqrt{\log(\sqrt{2\pi}\sigma\varepsilon)}, \sigma^2)$*, so that the probability density function of each distribution is greater than ε. Where* Norm *represents the Gaussian distribution, n represents the node number, σ represents the standard deviation, and ε represents the set minimum probability.*

Proof. Given a probability density function $h(x_p)$ of the Gaussian distribution $\mathrm{Norm}(\mu_p, \sigma^2)$, when $h(x_p) = \varepsilon$,

$$x_p = \mu_p \pm 2\sigma\sqrt{\log\left(\sqrt{(2\pi)}\sigma\varepsilon\right)} \tag{22}$$

Let $2\sigma\sqrt{\log\left(\sqrt{(2\pi)}\sigma\varepsilon\right)} = r$ as the distance from the average value μ_p to maximum and minimum value of x_p. Specify that each x_p represents the noise distribution of each node. In order to make all the distribution staggered and densely arranged, stipulate $\max(x_p) = \min\left(x_{p+1}\right)$, and keep all distributions symmetrical about $x = 0$. So , when the total number of nodes is N, $\mu_1 = (1 - N)r$, $\mu_2 = (3 - N)r$, ..., $\mu_N = (N - 1)r$, that is, $\mu_n = (2n - N - 1)r = 2\sigma(2n - N - 1)\sqrt{\log(\sqrt{2\pi}\sigma\varepsilon)}$

The process of generating sample $u^G(v)$ from staggered Gaussian noise $Z_v \sim P(x, v)$ is expressed as $u^G(v) = G\left(Z_v; \theta^G\right)$, θ^G denotes the weight of G. The process of generator is as shown in Fig. 4.

4.2 Discriminator and GAN framework

After proposing the generation of $u^G(n)$, we need to set discriminator D to evaluate the quality of $u^G(n)$ generated by generator G, and set the optimization mechanism of the two In order to make $u^G(n)$ can complete the graph embedding well.

We use graph embedding quality as an evaluation indicator to drive the entire process of confrontation generation, rather than simply letting G fit U. (Doing so may not guarantee the rigorous mathematical nature of the generating matrix,

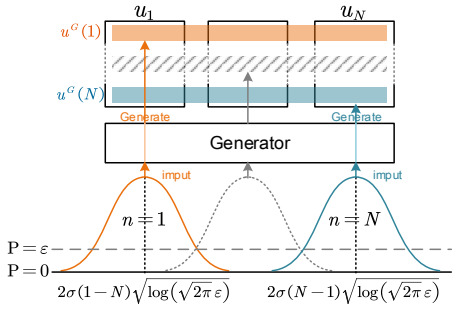

Fig. 4. Generator with Staggered Gaussian distribution as input

but for analog graph generation, we only need to obtain the best applicable sample, for example, it looks very similar to the target, rather than discussing the rigor of generating the sample). Specifically, D is divided into two parts: $D^{(enc)}$ used for encode (embed) the graph, and $D^{(dec)}$ used for decode, whose weights are $g_\theta^{D,dec}(\Lambda)$ and $\theta^{D,enc}$, respectively.

According to equation 4, Node embedding $f^e(v) = f(v)\sum_{l=1}^{N}\theta_l u_l^2(v)\rho(v,l,U_v)$, where two $u_l(v)$ are from U and U^T in Equation 1, respectively. In order to deal with the impact caused by different positions, we set different initializations for the two $u_l(v)$, $u_l(v)$ from U as the output of the generator, named $g^{(l,v)}$. The $u_l(v)$ from U^T as the initial weight of the discriminator, named $u_l^d(v)$, gradually progressive to $g^{(l,v)}$ during the adversarial training, the general term formula for its value in training epoch e is

$$u_{l,e}^d(v) = \begin{cases} u(v), e = 1 \\ u_{l,e-1}^d(v) + q\left(G_{l,e,v} - u_{l,e-1}^d(v)\right), e > 1 \end{cases} \tag{23}$$

In Equation 7, $u_{l,e}^d(v)$ represents the value of $u_l^d(v)$ in epoch e. $G_{l,e,v}$ represents for the generated value of v at position l in epoch e. q is the custom progressive coefficient. Next, we use $f^{e,G}(v)$ and $f^{e,D}(v)$ to represent the embedding of the node v when using $u^G(v)$ and $u(v)$ at epoch e, respectively. It can be seen that the purpose of adversarial training is to make $f^{e,G}(v)$ and $f^{e,D}(v)$ as similar as possible. The calculation equation for node embedding is:

$$\begin{aligned} f^{e,G}(v) &= f(v)\sum_{l=1}^{N}\theta_l g^{(l,v)} u_{l,e}^d(v)\rho(v,l,U_v) = u^G(v) g_\theta^{D,enc}(\Lambda) U^D f(v) \\ f^{e,D}(v) &= f(v)\sum_{l=1}^{N}\theta_l u(v) u_{l,e}^d(v)\rho(v,l,U_v) = u(v) g_\theta^{D,enc}(\Lambda) U^D f(v) \\ u^G(v) &= \left\{g^{(1,v)},\ldots,g^{(N,v)}\right\} \end{aligned} \tag{24}$$

The process of GAG is given in the algorithm 1, $U^{D,1} = U$ is the initialized feature matrix, $\left\{u_{l,e}^d(\gamma) \in U^{D,e} | \gamma \in (1,N)\right\}$ for epoch e, and gradually approaches

the generator matrix during the training process. In each training epoch e, first we obtain $u^G(v)$ and $U^{D,e}$ related to the target node v (According to theorem 1, $u^G(v)$ target the position of v. According to equation 2, U is the key to the conversion of graph convolution into spectral domain, and during the training process, $U^{D,e}$ assumes the role of U). Specifically, generate the $u^G(v)$ from the Staggered Gaussian noise corresponding to node v, and update the corresponding column of $U^{D,e-1}$ to $U^{D,e}$. Second, we use $D^{(enc)}$ and $D^{(dec)}$ to evaluate the quality of this epoch of generators, get the node embedding $f^{e,G}(v)$ of v through $D^{(enc)}$, decode $f^{e,G}(v)$ through $D^{(dec)}$ to get the label possibility y^{fake}. Third, calculate the loss functions of y^{real} and y^{fake} respectively, and update the weight of D according to the gradient.The loss function of D is:

$$\mathrm{Loss}_D(y) = \begin{cases} \mathrm{CrossEntropy}\left(\mathrm{Sigmoid}(y), label_v\right), y = y_{real} \\ \mathrm{CrossEntropy}\left(\mathrm{Sigmoid}(y), \mathrm{Sample}(\{label_\gamma | \gamma \neq v\})\right), y = y_{fake} \end{cases} \tag{25}$$

where $label_i$ represents the true label of node i (one-hot). Next we train G, through obtain its label probability y^{fool} of the output of G, and update the weight of G according to the output of D, in order to make the output of G is judged as real by D.

Formally, take u as input, we use $D(u;\theta^D)$ to express the output under the weight $\theta^D = \{g_\theta^{D,dec}(\Lambda), \theta^{D,enc}\}$, G and D are playing the following two-player minimax game with value function $V(G,D)$:

$$\max_{\theta^G}\min_{\theta^D}\sum_{v=1}^{N}\left(\mathbb{E}_{u\in U}\mathrm{Loss}_D(D(u;\theta^D)) + \mathbb{E}_{u\sim G(Z_v;\theta^G)}\left(1-\mathrm{Loss}_D(D(u;\theta^D))\right)\right) \tag{26}$$

The AN-GCN process is represented by fig 5

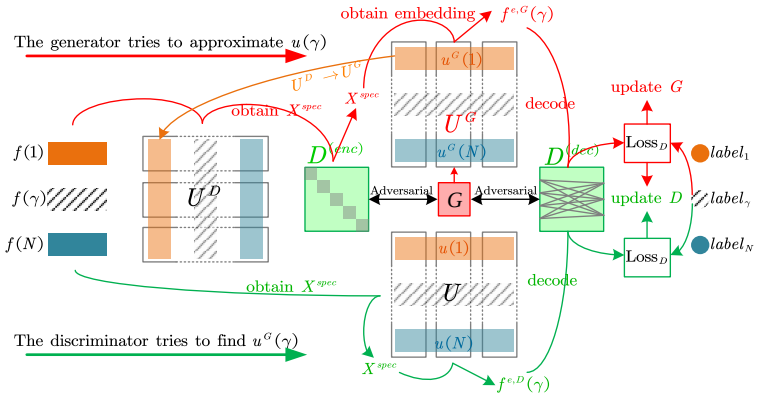

Fig. 5. The main process of AN-GCN

Algorithm 1 AN-GCN

Require: Weights of D: $g_\theta^{D,enc}(\Lambda)$ is used for GCN (encoding), $\theta^{D,dec}$ is used to decode to a specific label. Weights of G: θ^G. Training epoch of D and G.

Initialize $U^D = U$
for Train epochs for D **do**
  Random sample a node v
  Sample m noise samples $Z_v^D = \left\{z_1^{(v)}, \ldots, z_m^{(v)}\right\}$ from noise prior $P(z, v)$
  $u^G(v) \leftarrow G(Z_v)$ // generate
  • Column v of U^d moves closer to $u^G(n)$ with a coefficient of q
  $U_{\cdot v}^D = U_{\cdot v}^D + q\left(u^G(v) - U_{\cdot v}^D\right)$
  • Embedding and decoding for real and fake samples
  $f^{e,G}(v) \leftarrow u^G(v) g_\theta^{D,enc}(\Lambda) U^D f(v)$ // Get node embedding with $u^G(v)$
  $y^{fake} \leftarrow f^{e,G}\theta^{D,dec}$ // Decoding, expressed as $D_{U^D}(u^G(v); \theta^D)$
  $f^{e,D}(v) \leftarrow u(v) g_\theta^{D,enc}(\Lambda) U f(v)$ // Get node embedding with real $u(v)$
  $y^{real} \leftarrow f^{e,D}(v)\theta^{D,dec}$ // Decoding, expressed as $D_U(u(v); \theta^D)$
  • Calculate the gradient of D and update the weight of D
  $\nabla_{update}^D \leftarrow \nabla_{g_\theta^{D,dec}(\Lambda), \theta^{D,enc}}(\text{Loss}_D\left(y^{real}\right) + \left(1 - \text{Loss}_D\left(y^{fake}\right)\right)$
  for Train epochs for G **do**
    Sample m noise samples $Z_v = \left\{z_1^{(v)}, \ldots, z_m^{(v)}\right\}$ from $P(z, v)$
    $y^{fool} \leftarrow G(Z_v^G) g_\theta^{D,enc}(\Lambda) U^D f(v) \theta^{D,dec}$ // Get samples used to fool D
    • Calculate the gradient of D and update the weight of D
    $\nabla_{update}^G \leftarrow \nabla_{\theta^G}\left(1 - \text{Loss}_D\left(y^{fool}\right)\right)$
  end for
end for

Ensure: Trained generator weights θ^G.

5 Evaluation

We hope that AN-GCN can output accurate node embedding while keeping the node position completely generated by the well-trained generator, so we use the accuracy of node embedding to evaluate the effectiveness of AN-GCN. Moreover, because the current graph attack method based on the modified graph structure directly acts on the Laplace matrix, and AN-GCN ensures that the matrix eigenvalues of the Laplacian matrix is completely generated by the generator. Therefore, AN-GCN is immune to such attacks from the source, so we no longer evaluate it by reproducing such attacks.

Since the generator does not directly generate the node embedding, but completes the node prediction task by cooperating with the trained discriminator, we define the accuracy of the generator acc_G to be the accuracy of the node label after G and D cooperate, that is $acc_G = \frac{TP_G}{N}$, TP_G represents the number of True Positive samples when G determines the node embedding, the calculation method is

$$\mathrm{TP}_G = \sum_{\gamma=1}^{N} \zeta_\gamma, \zeta_\gamma = \begin{cases} 1, \operatorname{argmax}\left(D_{U^D}\left(G\left(Z_\gamma\right); \theta^D\right)\right) = \operatorname{argmax}\left(\mathrm{label}_\gamma\right) \\ 0, \operatorname{argmax}\left(D_{U^D}\left(G\left(Z_\gamma\right); \theta^D\right)\right) \neq \operatorname{argmax}\left(\mathrm{label}_\gamma\right) \end{cases} \tag{27}$$

At the same time, the accuracy of the discriminator is the accuracy of classifying nodes using U, $acc_D = \frac{TP_D}{N}$

$$\mathrm{TP}_D = \sum_{\gamma=1}^{N} \zeta_\gamma, \zeta_\gamma = \begin{cases} 1, \operatorname{argmax}\left(D_U\left(u(\gamma); \theta^D\right)\right) = \operatorname{argmax}\left(\mathrm{label}_\gamma\right) \\ 0, \operatorname{argmax}\left(D_U\left(u(\gamma); \theta^D\right)\right) \neq \operatorname{argmax}\left(\mathrm{label}_\gamma\right) \end{cases} \tag{28}$$

Since G performs more than 1 training epochs within each epoch of D training, the acc change of D is represented by points, and the acc change of G is represented by lines. They correspond to the same total epoch. We visualize U^G and X^E during training. U^G visualization uses the matshow function in numpy, X^E visualization uses tsne, and the nodes of different labels are marked with different colors, that is to say, if the visualized X^E shows good clustering and the colors in the cluster are uniform, It can be proved that the generator locates all nodes well. At the same time, we also use acc (accuracy) to quantify the node classification accuracy of the discriminator. The result is shown in the figure 6

As shown in Figure 4, acc_D and acc_G are rising at the same time. When epoch > 1400, acc_G remains at a high and stable state. We select G at epoch $= 1475$ as the final model selection, and the node embedding accuracy is 0.8227. We use acc_{GCN} to denote the accuracy of only GCN with the same kernal of D as a comparison. In the 1500 epochs of training, the highest value of acc_{GCN} is 0.7934. The experimental results show that under the same convolution kernel design, AN-GCN not only effectively maintains the anonymity of the node, but also has a higher detection accuracy than only GCN 0.0293 higher.

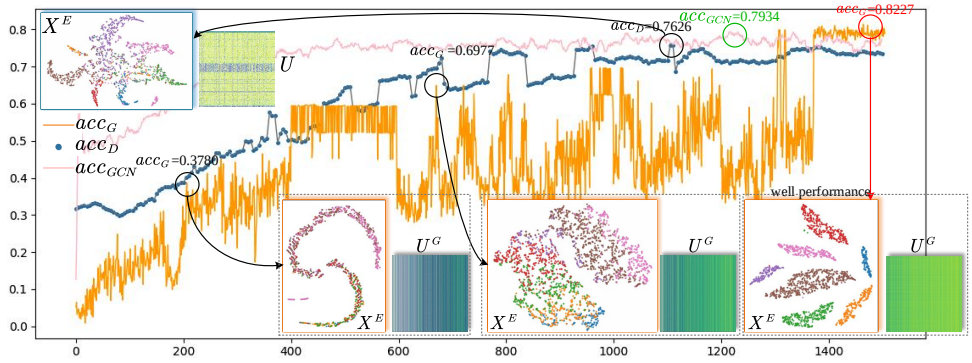

Fig. 6. Accuracy and visualization of node embedding during training, and visualization of node embedding

6 Conclusions and Future Work

We first proved that in GCN, the key to determining the position of node τ in the graph is $u(\tau)$, and then we designed a generator that can encode the node number, and generated the corresponding node number with $u(\tau)$ as the target. Then we designed the discriminator to complete the identification of the quality of $u^{(G)}(\tau)$ generated by generator, and designed an optimization method to combine the generator and the discriminator to complete the anonymous GCN of the node, and then complete the anonymous GCN of the node position of the whole graph, that is, AN- GCN.

At present, there is a problem with our work, that is, the generator contains the position prior information U^D of the graph. Although we try to make the U^D as close as possible to the generated samples during the training process, thereby gradually eliminating the a prior information, we have not yet given a specific judgment method to determine whether the a prior information reaches a safe level. However, this has little effect on the anonymization of nodes, because the prior information U^D implements the process of converting the overall graph structure to the spectral domain, and does not involve the positioning of individual nodes. That is to say, unless the attacker can modify the entire graph structure, we will discuss the security of the prior information introduced by U^D, which is basically unrealistic.

References

1. Dai Hanjun, Hui Li, Tian Tian, Xin Huang, Lin Wang, Jun Zhu, and Le Song. "Adversarial Attack on Graph Structured Data." In International Conference on Machine Learning (ICML), vol. 2018. 2018.
2. Zügner Daniel, Amir Akbarnejad, and Stephan Günnemann. "Adversarial attacks on neural networks for graph data." In Proceedings of the 24th ACM SIGKDD

International Conference on Knowledge Discovery and Data Mining, pp. 2847-2856. ACM, 2018.
3. Heng Chang, Yu Rong, Tingyang Xu, et al. A Restricted Black-box Adversarial Framework Towards Attacking Graph Embedding Models. AAAI Conference on Artificial Intelligence. 2020.
4. Aleksandar Bojchevski, Stephan Günnemann. Adversarial Attacks on Node Embeddings via Graph Poisoning. International Conference on Machine Learning. 97:695-704. 2019.
5. Binghui Wang, Neil Zhenqiang Gong, ACM Conference on Computer and Communications Security. 2019.
6. Xianfeng Tang, Yandong Li, Yiwei Sun, et al. Transferring Robustness for Graph Neural Network Against Poisoning Attacks. ACM International Conference on Web Search and Data Mining. 2020.
7. Ming Jin, Heng Chang, Wenwu Zhu, Somayeh Sojoudi. Power up! Robust Graph Convolutional Network against Evasion Attacks based on Graph Powering . International Conference on Learning Representations. 2020.
8. Daniel Zügner, Stephan Günnemann. Certifiable Robustness and Robust Training for Graph Convolutional Networks. ACM Knowledge Discovery and Data Mining. 2019.
9. Zhijie Deng, Yinpeng Dong, Jun Zhu. Batch Virtual Adversarial Training for Graph Convolutional Networks. ICML 2019 Workshop on Learning and Reasoning with Graph-Structured Data. 2019.
10. Shen Wang, Zhengzhang Chen, Jingchao Ni, et al. Adversarial Defense Framework for Graph Neural Network. arXiv. 1905.03679. 2019
11. Hongwei Wang, Jia Wang, Jialin Wang. GraphGAN: Graph Representation Learning with Generative Adversarial Nets .AAAI Conference on Artificial Intelligence. 2018
12. Ming Ding, Jie Tang, Jie Zhang. Semi-supervised Learning on Graphs with Generative Adversarial Nets. ACM International Conference on Information and Knowledge Management. 2018.
13. Joan Bruna, Wojciech Zaremba, Arthur Szlam, Yann LeCun. Spectral Networks and Deep Locally Connected Networks on Graphs. Annual Conference on Neural Information Processing Systems. 2014
14. Thomas N. Kipf, Max Welling. Semi-Supervised Classification with Graph Convolutional Networks. International Conference on Learning Representations. 2017
15. Hengtong Zhang, Tianhang Zheng, Jing Gao, Chenglin Miao, Lu Su, Yaliang Li, Kui Ren. Towards Data Poisoning Attack against Knowledge Graph Embedding. International Joint Conference on Artificial Intelligence. 2019
16. Aleksandar Bojchevski, Stephan Günnemann. Adversarial Attacks on Node Embeddings via Graph Poisoning. Proceedings of the 36th International Conference on Machine Learning, PMLR 97:695-704, 2019.
17. Minghong Fang, Guolei Yang, Neil Zhenqiang Gong , Jia Liu. Poisoning Attacks to Graph-Based Recommender Systems. Proceedings of the 34th Annual Computer Security Applications Conference: 381-392. 2018
18. Xuanqing Liu, Si Si, Xiaojin Zhu, Yang Li, Cho-Jui Hsieh. A Unified Framework for Data Poisoning Attack to Graph-based Semi-supervised Learning. Annual Conference on Neural Information Processing Systems. 2019
19. Jiaqi Ma, Shuangrui Ding, Qiaozhu Mei. Towards More Practical Adversarial Attacks on Graph Neural Networks. Annual Conference on Neural Information Processing Systems. 2020

20. Yiwei Sun, Suhang Wang, Xianfeng Tang, Tsung-Yu Hsieh, Vasant Honavar. Non-target-specific Node Injection Attacks on Graph Neural Networks: A Hierarchical Reinforcement Learning Approach. International World Wide Web Conferences. 2020
21. Andreas Loukas. What graph neural networks cannot learn: depth vs width.International Conference on Learning Representations (ICLR), 2020