Graph Attacking - Node Injection via Fake Duplicate

Yulin Zhao
University of Illinois
Urbana-Champaign
Urbana, Illinois, United States
yulin6@illinois.edu

Zhuoer Wang
University of Illinois
Urbana-Champaign
Urbana, Illinois, United States
zhuoerw3@illinois.edu

Pengyu Lu
the University of Illinois
Urbana-Champaign
Urbana, Illinois, United States
pengyul4@illinois.edu

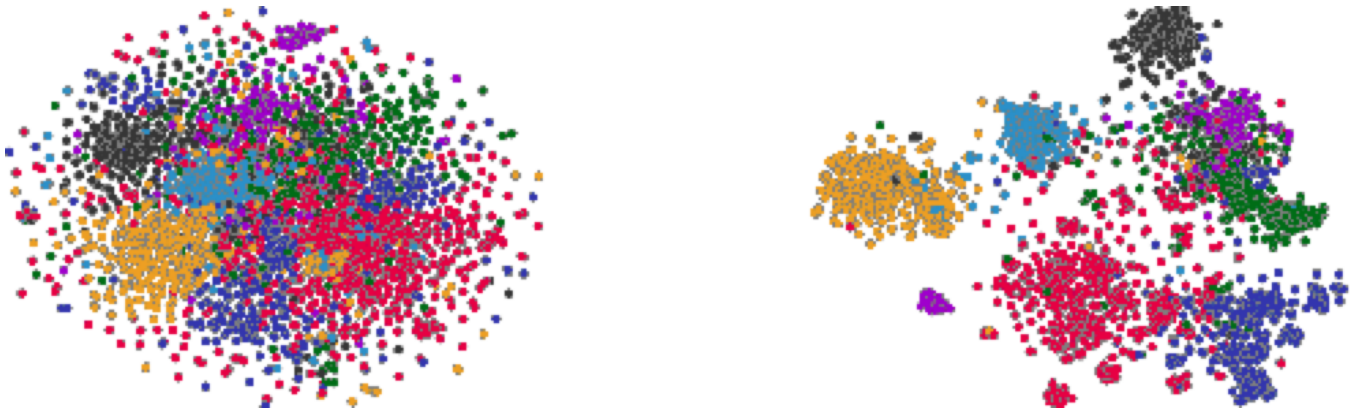

Figure 1: Node classification

ABSTRACT

As many node classification models are proposed for graph data mining tasks, there is more attention on the robustness of these classification models. Many studies suggest a small perturbation in the graph could result in a large decrease in the performance of graph models. In this project, we proposed a state-of-the-art attacking algorithm NIFD by injecting fake duplicated nodes with flipped labels and reconnecting neighbors to the fake nodes. In the experiments on two widely used datasets, our algorithm is proved to have small perturbation and result in a decrease in the performance of several popular node classification models.

CCS CONCEPTS

• **Information systems** → **Data mining**.

KEYWORDS

data mining, node classification, graph attack, poisoning attack, graph convolutional network, reinforcement learning

ACM Reference Format:
Yulin Zhao, Zhuoer Wang, and Pengyu Lu. 2022. Graph Attacking - Node Injection via Fake Duplicate. In *Proceedings of* . ACM, New York, NY, USA, 9 pages. https://doi.org/XXXXXXX.XXXXXXX

, *0.00*

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Attacking is an important part of developing defense strategies and improving the robustness of models. However, there are few works on attacking the network graphs. Nowadays, graph mining algorithms, including node classification and link prediction, are widely used in various application cases. Due to the graph network structure, it is hard to measure the change amount or design a very intuitive attacking algorithm.

In this project, our major tasks include deciding the attacking algorithm logic, setting a constraint by defining an effective metric for small perturbation, and implementing a machine learning model to select the attacked nodes or edges. We propose a new attacking algorithm by adding fake nodes and edges to poison the graph in the training stage. We make a duplication node of the attacked node with the same node feature and a different label. Then the duplication node will grab part of the neighbors from the original attacked node. The nodes to be attacked and the edges to be changed are decided by the importance of the neighbors of each node. We grouped the neighbors of each node, designed a machine-learning model to predict the label of the nodes using their neighbor groups, and take the prediction loss of each neighbor group as the indicator of the importance of the neighbors. We proposed different attacked node selection methods based on the calculated importance of the neighbors of each node. We also defined an effective metric for small perturbation based on the number of changed vertices and edges in the graph. We evaluate our attacking algorithms on two popular datasets, including CORA[6] and Pubmed[1], and different node classification models, including GCN[3], GraphSAGE[2], GAT[10], and SGC[11]. Our experiment results show that the attacking algorithm

[1]https://linqs.soe.ucsc.edu/data

decreases the performances of the classification models, especially GCN and GraphSAGE. The significance test shows that there is a significant difference in the node classification accuracy before and after the attack.

2 RELATED WORK

Our project targeting adversarial attacks for general types of node classification models, including Graph Convolutional Network (GCN)[3], GraphSAGE[2], Graph Attention Networks (GAT)[10], and Simple Graph Convolution (SGC)[11]. In recent years, graph adversarial attack has been a heated topic and different algorithms and frameworks have been proposed to perform the attacks on node classification algorithms. In 2017, a reinforcement learning-based method, NIPA[9], was designed to sequentially modify the adversarial information of the injected node. NIPA targeting on attacking node classification tasks. Later, Ma et al.[5] proposed another attacking framework via rewiring, which tries to modify the graph in a way that minimizes the change in the graph structure. The framework is named Rewatt and is designed for graph classification tasks attacking.

2.1 Node Classification Algorithms

2.1.1 Graph Convolution Network. The Graph Convolutional Network (GCN)[3] is proposed in 2017 as a semi-supervised learning approach on graph-structured data. The GCN model learns node representations via propagation in multiple convolutional layers. The layer-wise propagation rule is defined as

$$H^{(l)} = \sigma(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l-1)}W^{(l-1)}) \tag{1}$$

where $\tilde{A} = A + I_N$ is the adjacency matrix of the graph with self-connection, $\tilde{D}_{ij} = \sum_j \tilde{A}_{ij}$, $W^{(i)}$ is the layer-specific weight matrix to be trained and $H^{(l)}$ is the activation matrix of the current layer with $H^{(0)} = X$ where X is the matrix of node feature. $\sigma(\cdot)$ is the activation function of the current layer.

In this project, we adopted a two-layer GCN and the model can be reorganized by calculating $\hat{A} = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}$ in the pre-processing step, which simplifies the forward model in the following form:

$$f(X, A) = \sigma_2(\hat{A}\sigma_1(\hat{A}XW^{(0)})W^{(1)}) \tag{2}$$

where the weights $W^{(0)}$ and $W^{(1)}$ are trained with a neural network optimizer.

2.1.2 GraphSAGE. GCN is only suitable for transductive as it requires all nodes as the inputs. To address the new-node problem, the GraphSAGE[2] is proposed in 2018 to generate embeddings even for unknown nodes GraphSAGE samples and aggregates features from the local neighborhood on a graph-structured dataset by iteratively updating

$$h_{N(v)} = AGGREGATE_k(h_u^{k-1}, \forall v \in N(v)) \tag{3}$$

$$h_v^k = \sigma(W^k \cdot CONCAT(h_v^{k-1}, h_{N(v)}^k)) \tag{4}$$

with a L2 nromalization. It suggests three types of aggregators: mean, LSTM, and pooling aggregators. By adding a top (or output) layer, GraphSAGE can be directly applied to node classification tasks.

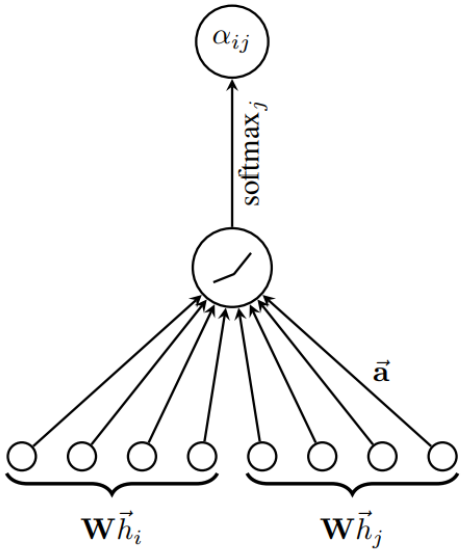

Figure 2: attention mechanism

2.1.3 Graph Attention Networks. The Graph Attention Networks (GAT)[10] is proposed in 2018 to introduce the masked self-attention layer to aggregate the features from neighborhoods and assign weights. GAT introduces attention coefficients $e_{ij} = a(W\overrightarrow{h_i}, \overrightarrow{h_j})$ to represent the importance of the node j to the node i, and further normalize them with the softmax function as

$$\alpha_{ij} = softmax(e_{ij}) = \frac{exp(e_{ij})}{\sum_{k \in N(i)} exp(e_{ik})} \tag{5}$$

where $N(i)$ represents the neighborhood of the node i. Parametrized with a weight vector and applied in the LeakyReLu function, we got the attention mechanism as shown in Fig. 2. The GAT outperforms GCN in several popular classification datasets as shown in Table 1.

Table 1: Accuracy Comparison of GCN and GAT[10]

	Cora	Pubmed
GCN	81.5%	79.0%
GAT	83.0±0.7%	79.0±0.3%

2.1.4 Simple Graph Convolution. The Simple Graph Convolution (SGC)[11] is proposed in 2019, aimed to reduce the complexity of GCN structure and calculation. SGC removes the nonlinearities and collapses weight matrices of GCN layers, enabling the model to scale to larger datasets and improving the speed up to two orders of magnitude. There are two main optimizations. First, different from Eq. (1), the feature propagation of each layer can be replaced with matrix operations as

$$H_{SGC}^{(l)} = SH^{(l-1)}, S = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}} \tag{6}$$

Second, GCN uses the ReLU function as the activation function on each convolutional layer and uses the softmax function on the

output layer, which can be represented as

$$\hat{Y} = softmax(ReLU(ReLU(...ReLU(H^{(0)}\theta^{(0)})...)\theta^{(l-1)})\theta^{(l)}) \quad (7)$$

SGC suggests ignoring the nonlinear function ReLU and then we can simply multiply matrices as

$$\hat{Y}_{SGC} = softmax(S^l X\theta) \quad (8)$$

As shown in Table 2 and Table 3, SGC keeps similar accuracy with GCN but reduces a lot of training time.

Table 2: Time Comparison of GCN and SGC[11]

	Cora	Pubmed
GCN	0.49	8.31
SGC	0.13	0.29

Table 3: Accuracy Comparison of GCN and SGC[11]

	Cora	Pubmed
GCN	80.53± 1.40	77.09± 2.95
SGC	80.62± 1.21	77.02± 1.62

2.2 Graph Attacking Algorithms

In 2017, a reinforcement learning method named NIPA[9] is proposed for node injection attacks. In this new method, the link structure of the original graph is not modified since no nodes or edges from the original graph are deleted. NIPA determines the label of the injected nodes and the fake edges in the graph using a Finite Horizon Markov Decision Process(MDP) and hierarchical Q-learning network. Each state change in MDP is regarded as one poisoning attack action, in which only one label of an injected node is changed and only one new fake edge between this injected node and one other node is established.

Another graph adversarial attack method that inspires us is Graph Adversarial Attack via Rewiring[5]. This paper aims to modify the original graph's structure to perform the attack. To ensure that the modification changes the structure of the graph in an "unnoticeable" way, the authors proposed a rewiring operation. The rewiring operation involves three nodes, noticed as N1, N2, and N3. The three nodes should satisfy the following condition: N2 is a first-hop neighbor of N1, and N3 is a second-hop neighbor of N1 but not a first-hop neighbor of N1. The rewiring operation deletes the existing edge between N1 and N2 and adds a fake edge that connects N1 and N3. During the attack, K rewiring operations are allowed where K is related to the size of the graph.

3 PROPOSED ATTACKING ALGORITHM

Inspired by the ReWatt[5] and NIPA[9], we proposed our own attacking algorithm Node Injection via Fake Duplicate (NIFD) as described in Algorithm 1 on node classification tasks. The intuition is to create a fake node v_f to introduce noise by changing class labels and diluting the node's influence by grabbing neighbors. The perturbation to the graph is small according to our definition but we believe it could effectively attack the node classification model.

The graph dataset for node classification contains several parts: the graph information G, the node features X, and the class label y. The graph information G includes the nodes $V = \{v_1, v_2, \ldots, v_n\}$ and the directed edges $E = \{e1, e2, \ldots, e_m\}$. Each directed edge e_i starting from vertex v_a and pointing to vertex v_b can also be represented as $e[v_a, v_b]$. Our algorithm NIFD will poison the (V, E, X, Y) to (V', E', X', Y'), where $(V, E, X, Y) \approx (V, E', X', Y')$.

3.1 Definition of Perturbation

The target of our graph attacking task is to affect the accuracy of the attacked model with small perturbation so that $(V, E, X, Y) \approx (V, E', X', Y')$. We define the perturbation of our attacking algorithm NIFD by the following formula

$$P(G, G') = \frac{\sum_{v_i \in V_G} \frac{\sum_{v_j \in (V_{G'} - V_G)} |||e[v_i, v_j]|||}{\sum_{v_j \in V_G} |||e[v_i.v_j]|||} + ||V_{G'} - V_G||}{||V_G||}$$

Given a graph G, the graph after the attack is denoted G'. The perturbation $P(G, G')$ is calculated using the proportion of nodes v_i that are changed among all the nodes in the original graph G. Since our algorithm does not change the label and the feature of existing nodes in the graph G, a changed node is either a node in the graph G that has lost some edges or a new duplication node that only exists in G'. For a node in G that has lost some of the edges, we define the change of such nodes using the ratio of lost edges to all edges. For new duplication nodes, we directly use the count of the nodes since all such nodes as well as their edges do not exist in the original graph G. We set the limit of small perturbation to $P(G, G') < 0.05$.

3.2 Node Injection via Fake Duplicates(NIFD)

The proposed Node Injection via Fake Duplication(NIFD) is attacking the graph by creating a fake node that is a duplication of some chosen node in the original graph. The duplication node will have a different label than the original node. We choose some neighbors of the original node, delete the edge between them, and connect these neighbors to the duplication nodes. In NIFD, a node v_j is the neighbor of the node v_i if $e[v_i, v_j]$ exists in the graph.

NIFD decides the nodes to grab and the neighbors of the attacked nodes to grab by the importance of the neighbors of the nodes. For each node, we group its neighbors with the same labels. Inspired by the idea of messaging passing, we built a model to calculate the importance of the groups of neighbors. The GraphSAGE calculates an embedding $CONCAT(h_v^{k-1}, h_{N(v)}^k)$, where $h_{N(v)}^k$ is an aggregation of the information of neighbors. We set the input of our model as the feature $F(v)$ of the node v and the mean of features of the neighbor $N(v)$ of the node v with the class label i, i.e. $X = CONCAT(F(v), mean(F(w), w \in N(v), y_w = i))$. Based on such information, we train a feed-forward model to predict the label of the node v. The model structure is shown in Fig. 3. After the training step, we calculate the loss between the prediction and the true class label and use it as the indicator of the importance of the neighbor groups. A neighbor group is more important to the node if it has a smaller loss. Because a smaller loss means the prediction

based on these neighbors is more accurate and these neighbors are providing more information in the correct prediction.

Assume a node has n neighbor groups with different labels, and the loss of each group is denoted as $l_i, \ldots, l_n$. We proposed three different ways to use the prediction loss of neighbor groups to choose the nodes to be attacked:

(1) Loss 1: take the average loss of all neighbor groups except the lowest one as the loss of each node. We pick the nodes with the highest node loss as the target of the attack.

$$Loss1 = \frac{(\sum_{i=1}^{n} l_i) - min(l_1, l_2, \ldots, l_n)}{n-1}$$

(2) Loss 2: take the smallest neighbor group loss as the loss of the node. Then we pick the nodes with the lowest node loss as the target of attack.

$$Loss2 = min(l1, l2, \ldots, ln)$$

(3) Loss 3: calculate the average loss of all neighbor groups and the average loss of all the neighbor groups except the lowest loss (i.e. Loss 1). The difference between the two average numbers is used as the node loss. Then we pick the nodes with the lowest node loss as the target of attack.

$$Loss3 = avg(l1, l2, \ldots, ln) - Loss1$$

When we attack a node, we will grab all the neighbors in the neighbor group with the lowest loss. Only nodes whose neighbors have at least two different labels are potential candidates for the attack because we will grab all the neighbors from a node whose neighbors have the same labels and that will make this node disconnected from all other nodes.

We proposed two different methods to determine the label of the duplication node. One method is randomly picking a label that is different from the attacked node. The other method is choosing the label of the attacked node's neighbor group that has the highest neighbor group loss.

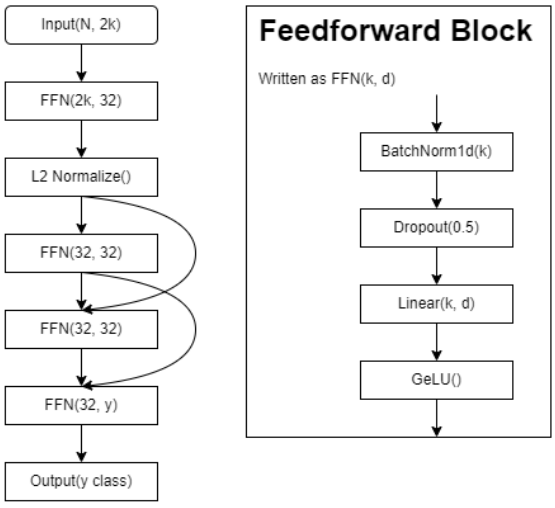

Figure 3: Model Structure

Algorithm 1 NIFD (Attacking Logic)

procedure NIFD(V, E, X, Y)
  for each node $v \in V$ **do**
    group neighbors of v by label $y_{neighbor}$
    calculate loss $l_1, \ldots, l_n$ of each neighbor group
    calculate node loss $Loss$
  end for
  sort the node loss and choose the nodes to attack V_{attack}
  for each node $v \in V_{attack}$ **do**
    Inject a fake node v_f with $X[v] = X[v_f]$
    Decide $Y[x_f]$ using label determine method
    Create $e[v, v_f]$ and $e[v_f, v]$
    Choose the neighbor group $V_{neighbor}$ with the lowest l_i
    for each node $v_i \in V_{neighbors}$ **do**
      Break $e[v, v_i]$
      Create $e[v_f, v_i]$
    end for
  end for
end procedure

4 DATASETS

We conduct our experiments on two widely used datasets for multiple graph-based data mining tasks, CORA[6] and Pubmed[2]. Both the CORA and Pubmed datasets we used are directed and multiclass. The edges of both datasets represent the "citation" relationship.

- CORA is a graph dataset built with scientific publications classified into seven categories. The features are a set of selected words. The edges represent the citation relationship.
- Pubmed consists of scientific publications from the Pubmed database related to diabetes. The features are TF/IDF weighted word vectors based on 500 unique keywords selected.

Table 4 is a summary of the datasets. For each dataset, we randomly select 80% of the nodes as the training set and the remaining 20% as the validation set.

Table 4: Dataset Summary

	Nodes	Edges	Features	Classes
CORA	2,708	5,429	1,433	7
Pubmed	1,9717	4,4338	500	3

5 EXPERIMENT RESULTS

5.1 Dataset Distribution

Data visualization is very useful to get a more direct picture of how data distribute and what information the model is trying to learn. In our project, all node classification models are neural networks and thus we can extract node embeddings to visualize the data distribution of two datasets. We extract the output of the hidden layer before the top layer (or prediction layer) and introduce Principle Component Analysis (PCA) to reduce the dimensionality to 2 so that we can plot them directly. The data distribution is shown in

[2]https://linqs.soe.ucsc.edu/data

Fig. 4, 5, 6, and 7. Each color represents a class label. We can see that each model learns the node embeddings differently but we can observe large clusters of the same class label. Thus, these four models seem to have good abilities to learn the graph information to predict node class labels.

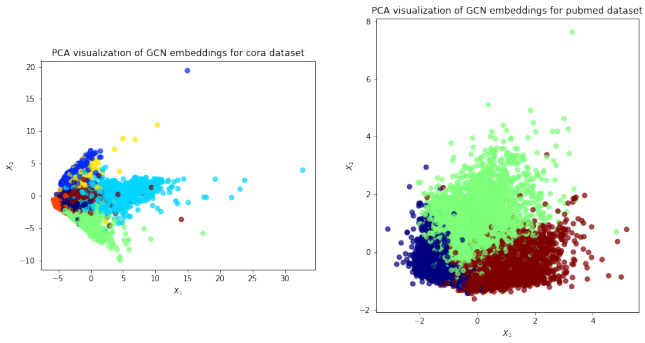

Figure 4: GCN embedding distribution of two datasets

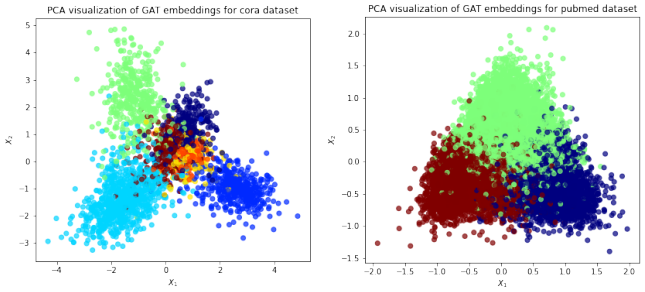

Figure 5: GAT embedding distribution of two datasets

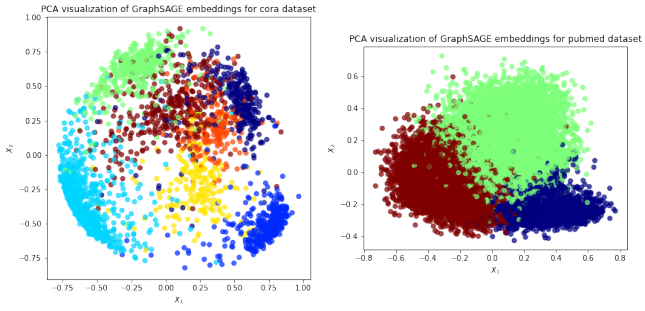

Figure 6: GraphSAGE embedding distribution of two datasets

5.2 Perturbation

We set the upper bound for the number of nodes we attack based on our definition of small perturbation. The following graph (Fig. 8) shows how perturbation becomes larger as the number of attacked nodes increases. The graph is generated with CORA dataset.

We defined a small perturbation as < 0.5. Therefore, we decided to attack 90 nodes for the CORA dataset and 360 nodes for the Pubmed dataset.

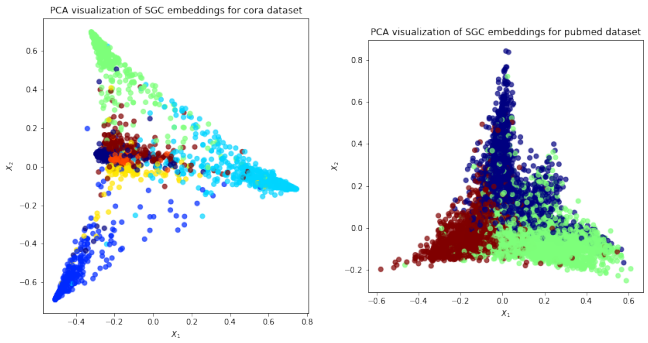

Figure 7: SGC embedding distribution of two datasets

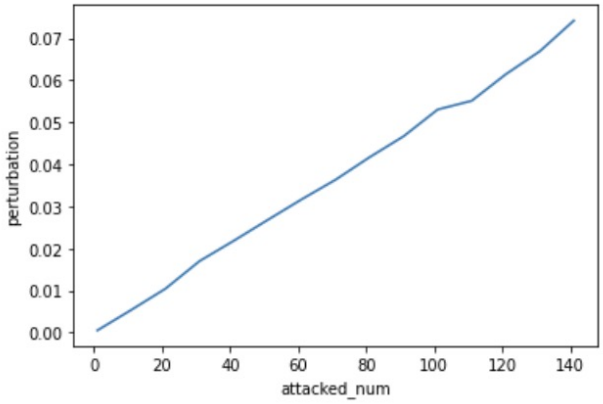

Figure 8: perturbation

5.3 Lossed and labels

We proposed several ways to pick the nodes we will attack based on the loss of neighbor groups (Loss1, Loss2, and Loss3). We also have two approaches, random and maximum loss, to generate labels for new fake nodes. First, we want to decide which loss we should use for choosing the attacked nodes and which label generation approach is better. So we experimented on three different losses and two label generation methods with the CORA dataset and all four classifiers. For each combination, we run the experiment ten times and get the average test accuracy. The results are as shown in Table 5.

From the results of the experiments, we found that the combination of the difference between average loss and average loss without the lowest (Loss 3) and randomly generated labels for the fake nodes can provide desirable results.

5.4 Experiments on different classifiers and datasets

Then with the selected Loss 3 for attacked nodes selection and random label for fake nodes, we had our final experiments on both the CORA and the Pubmed datasets with all GCN, GAT, SGC, and GraphSAGE classifiers. The results are in Table 6.

Table 5: Attacking performance with different losses and labels

Classifier	Loss	Label	Test Acc (before or) after Attack	Acc Change
GCN	Loss 1	Random	(92.73%) 91.75%	-0.98%
		Max Loss	(92.73%) 91.25%	-1.48%
	Loss 2	Random	(92.63%) 92.34%	-0.29%
		Max Loss	(92.62%) 92.45%	-0.17%
	Loss 3	Random	(92.73%) 91.72%	-1.01%
		Max Loss	(92.73%) 91.89%	-0.84%
GAT	Loss 1	Random	(91.94%) 91.77%	-0.17%
		Max Loss	(92.01%) 91.71%	-0.23%
	Loss 2	Random	(91.99%) 91.69%	-0.30%
		Max Loss	(91.99%) 92.05%	0.06%
	Loss 3	Random	(93.00%) 91.76%	-1.24%
		Max Loss	(92.98%) 91.82%	-1.16%
SGC	Loss 1	Random	(91.23%) 91.07%	-0.16%
		Max Loss	(91.23%) 91.70%	0.47%
	Loss 2	Random	(91.05%) 91.94%	0.89%
		Max Loss	(91.07%) 92.05%	0.98%
	Loss 3	Random	(91.23%) 91.13%	-0.1%
		Max Loss	(91.23%) 91.69%	0.46%
GraphSAGE	Loss 1	Random	(95.21%) 93.57%	-1.64%
		Max Loss	(95.21%) 93.96%	-1.25%
	Loss 2	Random	(95.20%) 93.81%	-1.39%
		Max Loss	(95.21%) 94.03%	-1.18%
	Loss 3	Random	(95.21%) 93.50%	-1.71%
		Max Loss	(95.21%) 93.98%	-1.23%

In general, our attacking model performs better on GCN and GraphSAGE classifiers and doesn't seem to affect much the performance of the SGC classifier.

Table 6: Attacking performance on CORA and Pubmed

Dataset	Classifier	Test Acc (before or) after Attack	Acc Change
CORA	GCN	(92.53%) 91.52%	-1.01%
	GAT	(93.00%) 91.76%	-1.24%
	SGC	(91.23%) 91.13%	-0.1%
	GraphSAGE	(95.21%) 93.50%	-1.71%
Pubmed	GCN	(91.23%) 90.50%	-0.73%
	GAT	(86.74%) 86.1%	-0.64%
	SGC	(77.92%) 77.10%	-0.82%
	GraphSAGE	(91.54%) 90.53%	-1.01%

The experiment is to compare the influence of attacking algorithms on the same four models and two datasets. Thus, we can use the paired t-test to evaluate the statistical significance. The null hypothesis is that there is no difference in the performance before and after the attack, which is $H0 : \mu_d = 0$. The t-stats are shown in Eq. 9 and the corresponding p-value is 0.00093. With the confidence interval $\alpha = 0.05$, we can reject the null hypothesis. In other words, we believe there is a significant difference before and after attacking, which shows our attacking algorithm does poison the graph resulting in the worse performance of graph models.

$$t = \frac{\bar{d} - 0}{s_d/\sqrt{n}} = -5.475 \tag{9}$$

6 NEXT STEPS

6.1 Reinforcement Learning for attacking

Our current NIFD algorithm selects the nodes to be attacked and edges to be grabbed from the attacked nodes based on the neighbor group loss calculated by a machine learning model. We are also planning to adopt reinforcement learning in NIFD.

Currently, we have built a new NIFD system using Q-learning to select nodes and edges to be attacked. The state is always the intermediate step of the attacked graph with the fake node added in the previous step. The reinforcement agent will first pick a node as the sample of the fake node, and then the agent selects the set of edges for the fake node. Finally, the agent assigns a new label for the fake node. We used Eq.10 as our reward function and terminate the learning processing when $P(G, G') \geq 0.05$. To reduce the complexity of calculating rewards for each state, we use the transferred model to estimate the accuracy. Due to the small perturbation on the graph for each action (or attack), most of the weights of our model can be re-used. Thus, for each state, we re-train the classifier model from the previous state on the new graph for 10 extra iterations, instead of training a new model.

$$r_t(s_t, a_1, a_2) = \begin{cases} acc(s_t) - acc(s_{t+1}) & \text{if } acc(s_t) > acc(s_{t+1}) \\ -1 & \text{otherwise} \end{cases} \tag{10}$$

However, due to computation resources and time limits, we are not able to train and test our new NIFD model based on Q-learning. For each training iteration, the RL model takes about 30 seconds and our setting has 20,000 steps, which is impossible to test and optimize. We plan to acquire more GPU resources and test our new NIFD system in the future. Another step we want to implement is to optimize our RL model to reduce the running time of taking action and calculating rewards.

6.2 Other Datasets

Our current experiments are conducted on the CORA dataset and Pubmed dataset. Both of them contain data about publications and citations. For publications, the relationship between them is usually positively related since the publications tend to cite papers from the same fields or topics. In the future, we plan to test our NIFD algorithm on data in different realms. One possible choice is GitHub Social Network[7], whose nodes are users who have starred more than ten repositories and edges represent the follower relationship among the selected users. Another dataset for our future work is the Deezer Europe Social Network[8], which is collected from the music streaming service Deezer.

6.3 Other Node Classification Models

We have applied the NIFD attacking algorithm on GCN, GAT, SGC, and GraphSAGE. In the future, we will apply NIFD on PPNP[4], GraphWave[1], and other state-of-art graph node classification models.

7 CONCLUSION

In this paper, we proposed the NIFD algorithm for attacking the network graphs. We started with defining the small perturbation for graph modification. The small perturbation sets the upper bound for the number of nodes we can attack in each network. Then we designed the algorithm for picking the nodes and edges we want to attack based on the idea of message passing. We conducted experiments on two datasets, CORA and Pubmed, and four classifiers, GCN, GAT, SGC, and GraphSAGE. The experiment results show that our attacking algorithms significantly decrease the accuracy of the node classification task on GCN and GraphSAGE by more than 1%. However, there is still a lot of room for improvement. For example, we can use reinforcement learning to optimize the node and edge selection process.

REFERENCES

[1] Claire Donnat, Marinka Zitnik, David Hallac, and Jure Leskovec. 2017. Spectral Graph Wavelets for Structural Role Similarity in Networks. *CoRR* abs/1710.10321 (2017). arXiv:1710.10321 http://arxiv.org/abs/1710.10321

[2] William L. Hamilton, Rex Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. *CoRR* abs/1706.02216 (2017). arXiv:1706.02216 http://arxiv.org/abs/1706.02216

[3] Thomas N. Kipf and Max Welling. 2016. Semi-Supervised Classification with Graph Convolutional Networks. *CoRR* abs/1609.02907 (2016). arXiv:1609.02907 http://arxiv.org/abs/1609.02907

[4] Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. 2018. Personalized Embedding Propagation: Combining Neural Networks on Graphs with Personalized PageRank. *CoRR* abs/1810.05997 (2018). arXiv:1810.05997 http://arxiv.org/abs/1810.05997

[5] Yao Ma, Suhang Wang, Tyler Derr, Lingfei Wu, and Jiliang Tang. 2021. Graph Adversarial Attack via Rewiring. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery Data Mining* (Virtual Event, Singapore) *(KDD '21)*. Association for Computing Machinery, New York, NY, USA, 1161–1169. https://doi.org/10.1145/3447548.3467416

[6] Andrew McCallum, Kamal Nigam, Jason D. M. Rennie, and Kristie Seymore. 2004. Automating the Construction of Internet Portals with Machine Learning. *Information Retrieval* 3 (2004), 127–163.

[7] Benedek Rozemberczki, Carl Allen, and Rik Sarkar. 2019. Multi-scale Attributed Node Embedding. arXiv:1909.13021 [cs.LG]

[8] Benedek Rozemberczki and Rik Sarkar. 2020. Characteristic Functions on Graphs: Birds of a Feather, from Statistical Descriptors to Parametric Models. In *Proceedings of the 29th ACM International Conference on Information and Knowledge Management (CIKM '20)*. ACM, 1325–1334.

[9] Yiwei Sun, Suhang Wang, Xianfeng Tang, Tsung-Yu Hsieh, and Vasant Honavar. 2019. Node Injection Attacks on Graphs via Reinforcement Learning. https://doi.org/10.48550/ARXIV.1909.06543

[10] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net. https://openreview.net/forum?id=rJXMpikCZ

[11] Felix Wu, Tianyi Zhang, Amauri H. Souza Jr., Christopher Fifty, Tao Yu, and Kilian Q. Weinberger. 2019. Simplifying Graph Convolutional Networks. *CoRR* abs/1902.07153 (2019). arXiv:1902.07153 http://arxiv.org/abs/1902.07153

A TEAM MEMBERS

A.1 Zhuoer Wang

1/3 overall contribution:

- Prepare two datasets: 100%
- Design our RL environment: 33.3%
- Implement our RL environment and agents: 40%
- Implement four node classification model: 100%
- Design our attacking algorithm: 40%
- Produce results table: 100%

A.2 Yulin Zhao

1/3 overall contribution:

- Implement our GCN: 100%
- Design our RL environment: 33.3%
- Implement our RL environment and agents: 40%
- Design our attacking algorithm: 40%
- Implement our attacking algorithm: 33.3%
- Produce the visualization of datasets: 100%

A.3 Pengyu Lu

1/3 overall contribution:

- Design our attacking algorithm: 20%
- Design our RL environment: 33.3%
- Implement our RL environment and agents: 20%
- Implement our attacking algorithm: 66.6%