Graph Adversarial Attacks and Defense: An Empirical Study on Citation Graph

Chau Pham
Computer Science Department
Texas Tech University
Lubbock, USA
chaupham@ttu.edu

Vung Pham
Computer Science Department
Texas Tech University
Lubbock, USA
vung.pham@ttu.edu

Tommy Dang
Computer Science Department
Texas Tech University
Lubbock, USA
tommy.dang@ttu.edu

2020 IEEE International Conference on Big Data (Big Data) | 978-1-7281-6251-5/20/$31.00 ©2020 IEEE | DOI: 10.1109/BigData50022.2020.9377988

***Abstract*—This paper details the methodologies and decisions making processes used while developing the attacking and defending models for the Graph Adversarial Attacks and Defense applied to a large citation graph. To handle the large graphs, our attack strategy is twofold: 1) randomly attack the structure first, 2) keep the structure unchanged, then continue the attack on the features using the gradient-based method. On the other hand, the defender is based on 1) filtering and normalizing the feature data, 2) applying the Graph Convolutional Network model, and 3) selecting the models with the highest accuracy and robustness based on our own attacking data. We applied these strategies in KDD Cup 2020 on Graph Adversarial Attacks and Defense dataset. The attacker can drop the accuracy of a surrogate 2-layer Graph Convolutional Network model from 60% to 30% on the test set. Our defending model has 68% accuracy on the validated data and has 89% of the target labels remained the same while adding fake nodes, generated by our attacking method, to the graph.**

***Index Terms*—graph neural network, graph convolutional network, graph adversarial attacks, graph defense**

I. Introduction

Graph representations are pervasive thanks to their expressive power to support analyzing relationships among entities [1], [2], comparing larger graphs [3], [4], or enabling subgraphs extractions [5] for knowledge discovery purposes. Thus, ubiquitous domains use graph representations and require graph analysis, in which the citation graph [6] is a typical example. Citation graph analysis is common because its solutions can be extended to different application domains such as white-collar crime [7], drug repurposing from literature knowledge graph construction [8], and financial forecasting [9], to name but a few.

Also, recent advancements in deep learning have given rise to a large number of works in applying learned models to solving tasks in different application areas [10] and have exhibited remarkable success [11], [12]. Graph analysis is no exception. Specifically, there is an enormous number of deep learning works in recent years regarding graph encodings [13] for different purposes, such as node classifications and link predictions. Among these works, Graph Convolutional Network (GCN) [14] gained initial success in the field and is the base for many other related techniques such as GraphSAGE [15], FastGCN [16], Graph Attention Networks (GAT) [17], Cluster-GCN [18], and Graph Isomorphism Network (GIN) [19].

However, many of the current solutions are still vulnerable to attacks if applied naïvely and or not analyzed thoroughly before deploying into the real-world. Specifically, there are works in the literature suggesting that many current deep learning models for graph analysis are vulnerable. For instance, Zügner et al. [20] show that a few, unnoticeable node feature and graph structure perturbations can drop classification accuracy significantly on node classification tasks for multiple datasets.

The adversarial attacks reduce the public trusts and hamper the development of techniques using graph analysis in many application domains. Therefore, this work investigates the vulnerability of academic graphs and devises strategies to defend them. Thus our contributions are:

- We devise strategies to attack the citation graphs with heuristics to avoid real-world data constraints such as large graph structure, discrete graph structure values, and conservative/unnoticeable perturbations.
- We also contrive and experiment techniques to build a robust Graph Convolutional Network model leveraging the knowledge gained through data exploration and characteristics of the poisoned data generated by our attacking strategies, with a prediction time constraint.
- Finally, we apply these strategies to the KDD Cup 2020 on Graph Adversarial Attacks and Defense citation graph dataset [21] to test and evaluate our attacking and defending approaches.

II. Datasets and attacking/defending tasks

This section defines our research problem and its requirements. Figure 1 summarizes the dataset and tasks for a typical graph adversarial attack and defense case. The left panel depicts the dataset with embedded features for the nodes and the adjacency matrix for the edges. The right panel explains the attacking and defending tasks competing over the classification accuracy based on the provided data. First, we need to devise strategies to attack the graph by adding nodes and edges. On the other hand, we also strive to improve methods to train models robust to the attacks. Specifically, this work uses the dataset provided by KDD Cup 2020 on Graph Adversarial Attacks and Defense [21] as a case study. This dataset contains 5,757,154 edges and 659,574 nodes. These nodes are divided into 60,9574 nodes for training and 50,000 nodes for testing, respectively. Also, each node was embedded by a 100-dimensional feature vector consisting of float numbers between $(-2, 2)$, so the data was ready for training and evaluating deep graph neural networks without having to go through the initial graph embedding processes.

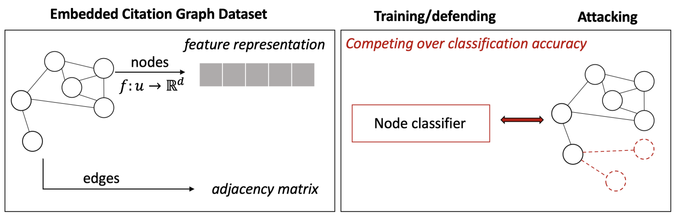
Embedded Citation Graph Dataset
Training/defending
Attacking
feature representation
nodes
$f: u \rightarrow \mathbb{R}^d$
edges
adjacency matrix
Competing over classification accuracy
Node classifier

Fig. 1. Overview of the dataset and tasks for graph adversarial attacks and defense cases. The left panel depicts the given graph dataset with embedded node features and adjacency matrix. The right panel shows the attacking task by adding artificial nodes/edges (dashed, red circles/lines) and the defending task by training robust models. These two tasks compete over the node classification accuracy.

For the attacking task, attackers need to perturb the graph by adding no more than 500 nodes and up to 100 edges per fake node. The adjacency matrix now becomes $A' = \begin{bmatrix} A & B^T \\ B & C \end{bmatrix}$, and the new feature matrix is $X' = \begin{bmatrix} X \\ X_{fake} \end{bmatrix}$, where A is the original adjacency matrix, and X is the original feature matrix. Figure 2 demonstrates more details on the data set and the constraints of the attack. The artificial node and edge limitations are to ensure that the attacks are conservative and unnoticeable.

The attacking purpose is to reduce the node classification accuracy for the testing nodes by the target models learned from the training nodes. We do not know the target model, so it is considered a black-box setting. Moreover, the defending model learned from the training data must be robust and can keep its accuracy on the testing data after being attacked by the fake nodes and edges. Besides, there are also memory and prediction time constraints. Specifically, the size of the defending model should not exceed a maximum of 1GB. The model should take less than 10 seconds on a K80 GPU and 60 GB memory server to perform node classification on the whole graph.

We first test our own defending and attacking models against each other. However, to be objective and accurately evaluate our attacking and defending strategies, we also submit our results to be assessed by the KDD Cup 2020 organizer. The organizer uses the following formula to calculate the final score of the attacker and defender of a team i versus other n competitors:

$$score_i = \frac{\sum_{j=1, j \neq i}^{n} (1 - f(d_j, a_i)) + \sum_{j=1, j \neq i}^{n} f(d_i, a_j)}{2n}$$

where, $f(d_i, a_j)$ means the accuracy of the defender d_i on the data attacked by attacker a_j. In other words, the attacker's performance is measured by the average of how much it can reduce the accuracy of the models developed by other competitors. On the other hand, the defender's performance is measured by its average accuracy on graph data attacked by other teams.

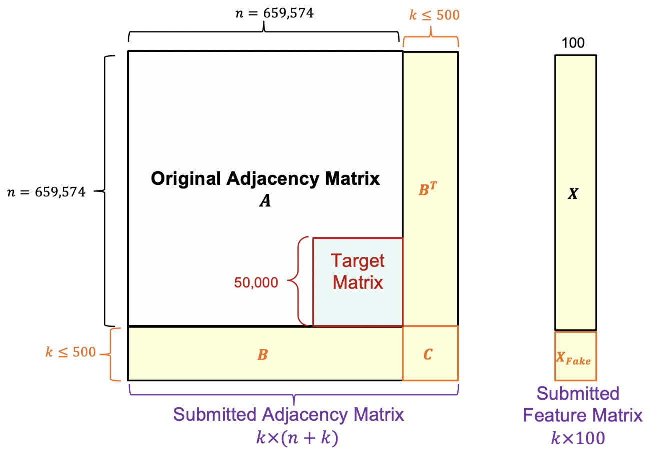

Fig. 2. A closer look at constraints of the attack. In this black-box scenario (i.e., the target model is unknown), no modification on the original data is allowed. We aim to attack 50,000 nodes in *Target Matrix* by adding no more than 500 nodes, each can only have up to 100 edges (Matrix B & C). The attack is carried out by appending an additional $k \times (k+n)$ adjacency matrix to the original adjacency matrix A and also a feature matrix X_{Fake} of the fake nodes containing k vectors in $\mathbb{R}^{100}$ to the original feature matrix X. Note that all the squared adjacency matrices are symmetric.

III. Related Work

A. Graph Neural Networks

It is critical to understand how Graph Neural Networks (GNNs) [22] work in graph adversarial attacks and defense research. Specifically, Convolutional Graph Neural Network (ConvGNN), first introduced in [23], is a type of GNNs that can work directly on graphs and takes advantage of the structural information. It can solve the problem of classifying nodes in a graph, where labels are only available for a small subset of nodes (semi-supervised learning). Figure 3 depicts the typical architecture of a ConvGNN with multiple layers for node classification, where *Gconv* denotes a convolutional graph layer, X is the feature matrix of the graph. The node representation is generated by aggregating its neighbors' features and the feature of itself. After that, these aggregates go through a non-linear function, such as ReLU. By stacking multiple graph convolutional layers, the message from a node can be passed further in the network to extract high-level node representations.

In [24], Wu and Zhang further categorize ConvGNNs into two categories, namely Spectral-based and Spatial-based. Spectral-based approaches define graph convolutions by introducing filters from the perspective of graph signal processing and using the convolutional graph operation to remove noises from graph signals. In contrast, Spatial-based approaches make

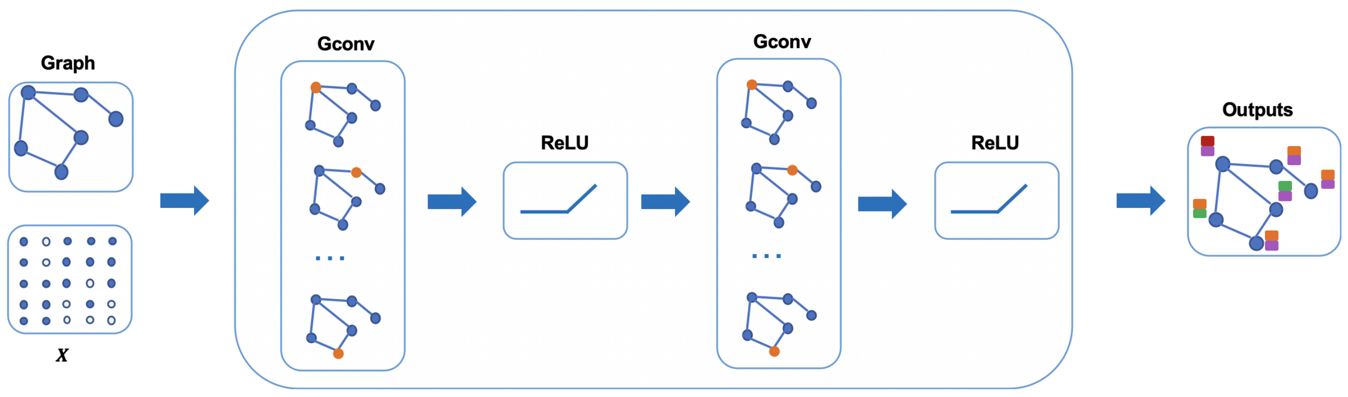

Fig. 3. The general architecture of a ConvGNN with multiple graph convolutional layers for node classification, where X is the feature matrix of the input graph and *Gconv* means a convolutional graph layer. The visualization is adapted from [24].

use of information propagation. Besides, as the name "Convolutional" suggests, the idea is to gather information from a node's neighborhood in graphs. This idea is generalized from the computer vision field of research. However, while images have a fixed structure, graphs are much more complicated.

Recently, Kipf and Welling introduced Graph Convolutional Network (GCN) [14], the bridge that filled the gap between spectral-based approaches and spatial-based approaches [24]. Since then, many new methods have been developed based on spatial-based due to its attractive efficiency and generality. In GCN, the information is aggregated by taking the weighted average of all neighbors' node features (including itself), preferring lower-degree nodes (i.e., lower nodes get larger weights). The resulting feature vectors are then fed into a fully connected layer for training and then passed through a non-linear function such as ReLU. Given a graph $G = (A, X)$, whereas X is the feature matrix and A is the adjacency matrix, the forward model then takes the simple form stated as the following formula:

$$Z = f(X, A) = \operatorname{softmax}\left(\hat{A}\operatorname{ReLU}\left(\hat{A}XW^{(0)}\right)W^{(1)}\right)$$

where, $\tilde{A} = A + I_N$ is the adjacency matrix of the graph G after adding self-loops via the identity matrix I_N. $W^{(l)}$ is the trainable weight matrix of layer l, $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$, and $\hat{A} = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}$.

For semi-supervised classification, cross-entropy error is evaluated over all labeled examples:

$$\mathcal{L} = -\sum_{k \in \mathcal{Y}_k}\sum_{f=1}^{F} Y_{kf}\ln Z_{kf}$$

where, Y_k is the set of node indices that have labels, F is the dimension of feature maps in the output layer.

In practice, a deeper GNN can be generated by stacking more layers on top of each other. The output of a layer is the input for the next layer. In GNNs' concept, the number of layers is considered the maximum number of hops that messages can pass through. So, depending on how far we think a node should get information from the networks, we can configure a proper number of layers. In practice, we do not want to go too far. Typically, with a few hops (e.g., 6 in the case of social network [25]), we almost cover the entire graph for real-world graphs, making the aggregation less meaningful.

GNNs may become inefficient to work with large graphs that cannot fit into memory at once as the number of a node's neighbors can vary from a few to even thousands. To tackle this issue, GraphSAGE [15] samples a fixed size of nodes from each node's neighbors to aggregate the information. Furthermore, it also provides more sophisticated aggregators such as LSTM cell and max-pooling rather than the weighted average as in GCN.

Another popular ConvGNN model is Graph Attention Network (GAT) [17], which is inspired by well-known Attention mechanisms in Natural Language Processing tasks [26]. Attention mechanisms allow the machine learning model to focus on the most relevant parts of the input which in turn boosts the performance. More specifically, GAT uses an attention-based architecture to perform node classification of graph-structured data by assigning larger weights to the more "important" nodes. These weights are learned with a neural network within an end-to-end framework instead of using fixed weights as in GCN. The attention mechanism shows some improvement in performance over other GNNs techniques such as GraphSAGE on node classification tasks [24].

In [19], Xu et al. analyze the discriminative power of popular GNN variants, such as GCN [14] and GraphSAGE [15]. Their work shows that these GNNs are not able to distinguish certain simple graph structures. The authors then introduce Graph Isomorphism Network (GIN), a new technique that is closely related to Weisfeiler-Lehman graph isomorphism test (WL test) [27]. They state that the new techniques are as powerful as the WL test in distinguishing graph structures and achieves state-of-the-art performance on many graph classification benchmarks.

Finally, GNNs often require the graph to be represented in numeric forms (i.e., vectors for nodes and adjacency matrix for links), using graph embedding techniques. Regarding the citation graph in which nodes represent research papers while edges are the citations. In the first step, the papers are converted into vectors using, e.g., Natural Language Processing (NLP), embedding techniques such as Doc2Vec [28] and Sentence-BERT [29]. One of the ConvGNN methods can then be employed to solve specific tasks such as node classifications or link predictions.

B. Adversarial Attacks and Defense on Graphs

Graph neural networks have been around recently and are gaining traction. However, there are not many works on adversarial attacks for graphs. Attacking on a graph G means to perform small perturbations on it to make the classification performance drop. Given $G = (A, X)$, modifications of the adjacency matrix, A, are called structure attacks, while changes to nodes' features, X, are called feature attacks. Figure 4 demonstrates some possible types of attack on a graph in which target nodes are the nodes whose classification labels we want to change, whereas attacker nodes are the nodes that the attacker can modify [20]. Specifically, related nodes involve attacker nodes and target ones. Also, there are direct and indirect attacks depending on which nodes/links the attackers are altering.

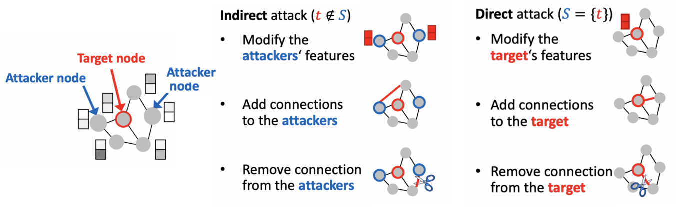
Target node
Attacker node
Attacker node
Indirect attack ($t \notin S$)
Modify the attackers' features
Add connections to the attackers
Remove connection from the attackers
Direct attack ($S = \{t\}$)
Modify the target's features
Add connections to the target
Remove connection from the target

Fig. 4. Two types of attacks on graphs (adapted from [30]). The indirect attack may modify the attackers' features, add connections to the attackers, or remove connections from the attackers. On the other hand, direct attacks may modify the targets' features, add connections to the targets, or remove connections from the targets.

One common method is to adopt adversarial attack techniques in Deep Neural Networks to the GNNs, such as gradient-based techniques. Using these techniques on graphs is not a trivial task since the adjacency matrix contains discrete values causing inaccurate gradient update. Consequently, the gradient-based approaches may achieve sub-optimal attack performance due to the inaccurate gradients on discrete data. In recent work, Zügner et al. [20] propose an efficient algorithm named Nettack for performing transferable attacks. First, the authors generate a linear surrogate model by replacing the non-linear function (e.g., ReLU) with a simple linear activation function and then performing both feature and structure attacks. There are also constraints on the changes to assure that the attack is unnoticeable. After that, they exploit the adversarial attacks' transferability to attack other models with similar architecture and are trained on the same dataset. The strategy can attack the node classification problem without knowledge about the graph.

In another work, Wu et al. [31] introduce a technique employing integrated gradients to determine the effect of changing selected features or edges. Specifically, attackers use integrated gradients for all nodes and edges in a target's neighborhood to attack it. These values are sorted and considered as "importance scores" for the corresponding nodes and edges. Then, the feature in the node or link with the highest score will get flipped first (i.e., 0 to 1, or 1 to 0), and the process keeps repeating for the rest of the nodes and edges. The authors also propose a defense technique. They observe that adding edges tends to have more effect than removing edges, and perturbing edges provides a better attack than modifying features. Moreover, high-degree nodes are more difficult to attack than low-degree nodes. Thus, in their defense method called GNN-Jaccard, they suggest removing edges that connect very dissimilar nodes and then train the model on the modified data to make the model more robust.

Wang et. al [32] employ a new attack strategy in a quite similar setup to ours. In their setting, the attacker can only add malicious fake nodes to the original graph without changing any existing edge or feature. Moreover, the proposed algorithm can also find a small perturbation to attack a group of nodes at the same time. It adopts a greedy approach starting all 0 in the additional adjacency and the feature matrices, then one feature or one edge is added at each step by changing its value to 1. Moreover, the authors propose a new technique named greedy-GAN to generate fake nodes with features similar to the real ones. They use a loss function to measure the trade-off between attacking power and the realness of the fake nodes. That being said, they make some assumptions in terms of models and data set, such as the target model is GCN [14], and both the additional feature matrix and adjacency matrix are discrete (i.e., 0/1 matrices).

There are some other works focusing more on the defense perspective. Entezari et al. [33] employ Low-Rank Solutions to resist attacks from Nettack introduced by Zügner et al. [20]. More specifically, they propose a technique computing the low-rank approximation of the adjacency and feature matrices through an SVD decomposition. The method acts as a noise filter and helps to retain only useful information of the graph. The authors claim that their defense strategy can achieve a performance close to the performance on the original graph. In another work, Wang et al. [34] introduce an adversarial training method called GraphDefense to improve the robustness of GCNs. The authors do not constrain the adjacency matrix element to be discrete during the training process allowing them to use gradient descent on the adjacency matrix effectively.

All in all, these above-mentioned attacking strategies have several assumptions regarding available information concerning the graphs, discrete in feature matrix, memory, and computation time that cannot be satisfied in this specific case. Therefore, in the next sections, we detail these constraints and our strategies to tackle the attacking and defending tasks.

TABLE I
SUMMARY OF COMMON CITATION NETWORKS [35] AND THE KDD CUP 2020 DATA SET.

Dataset	#Nodes	#Edges	#Features	#Classes
Citeseer	3,327	4,732	3,703	6
Cora	2,708	5,429	1,433	7
Pubmed	19,717	44,338	500	3
Cora-ML	2,995	8,416	2,879	7
KDD2020	659,574	5,757,154	100	18

IV. ATTACKER

There are some strategies to attack graphs, as mentioned in the previous session. However, these methods require a lot of memory to keep track of the attacking information related to the adjacency matrix (e.g., $5,757,154 \times 5,757,154$ matrix for the gradients when using gradient-based approaches, in this case). Besides, it is obvious that the KDD Cup 2020 data set is much bigger and has more classes than some well-used citation network data sets, as shown in Table I. Thus, the above-mentioned strategies are impractical and inapplicable in this situation from a memory perspective. On the other hand, we have many target nodes (50,000) to attack rather than focusing on only one target. Furthermore, there are more strict constraints upon attack strategies that can apply to the graph in this specific case. For instance, we cannot remove any edges from the input graph. Regarding adding new edges, the fake edges can only initiate from the artificial nodes, and not to mention that the labels of target nodes are unknown. Consequently, we can not apply the high complexity methods mentioned in the previous works directly to the challenge. The tight constraints make it difficult to influence target nodes. As a result, it is extremely challenging to launch a severe attack on the input graph. Therefore, it is necessary to develop a new strategy that can work well on large-scale graphs under strict conditions.

Our method is to choose a good graph structure attack first, instead of striking on both the structure and feature at once. By dealing with one type of attack at a time, it is easier to carry out the attack. After finding an effective attack strategy on the graph's structure, we keep it unchanged and adopt gradient-based techniques for feature attacks. By doing this, we do not have to keep track of the gradients of the adjacency matrix, and we can attack many target nodes at once.

Several heuristics were explored for the structure attacks. The following sections list out our main ideas (Figure 5):

Structure attack on all (50,000) target nodes: The most straightforward way for a structure attack is to randomly choose 100 target nodes for each fake node. The more advanced one would be clustering the 50,000 target nodes (e.g., K-means on feature vectors and/or labels to cluster the target nodes) so that each fake node will connect to 100 similar target nodes. The aims of doing these are that the attack can be more efficient since clustered nodes are similar and can be attacked in batch.

Structure attack on a part of target nodes: With this approach, we aim to concentrate on some promising target nodes rather than scatter attack power on the whole set of target nodes. After filtering some target nodes to attack, we can further cluster them to connect similar target nodes to the same fake node. The strategies to isolate nodes with higher priority to attack are:

- Attack on low degree nodes first. It's more challenging to attack higher degree nodes (though more attractive) because more neighbors connect to these nodes. Zügner et al. [20] stated the same in their work.
- If we know some nodes that the model predicts incorrectly, we do not need to attack them. Thus, we can focus on "promising" nodes that we think the model can predict correctly. To get a list of promising target nodes, we build an extra model to learn which nodes the model can predict. We use a gradient boosting algorithm, LightGBM [36], to get highly confident nodes that the model can predict and attack on these nodes only. The task is a binary classification, in which class 1 indicates a node that our GCN model can predict, whereas class 0 represents a node that cannot be predicted. When we get half of the target nodes to attack, 83% of these nodes can be predicted by the GCN model.

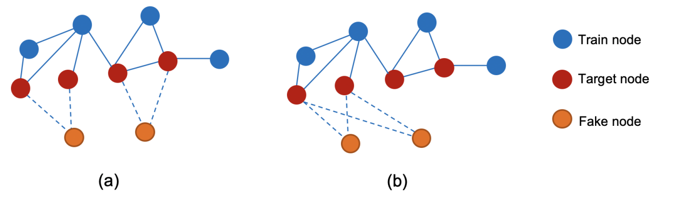

Fig. 5. Structure attack strategies: (a) All target nodes have at least an edge to the fake nodes; (b) Only some target nodes have edges to the fake nodes

As stated, due to the graph size, discrete optimization complexity, time, and memory constraints, we cannot perform both structure and feature attacks simultaneously. Therefore, after finishing the structure attack, we keep the structure unchanged and attack the features. To attack the target nodes, we need the labels of these nodes. However, these labels are missing from the input. Thus, we devise a strategy to obtain this by training a surrogate 2-layer GCN model on the data and then use the learned model to predict the target nodes' labels. In this specific case, to keep the feature attack unnoticeable, it is required to keep feature values in a range of [-2, 2]. However, we further constrain the value range to [-1.73, 1.62] after exploring this specific graph citation dataset's value range.

For each batch, the loss function is cross-entropy calculated on target nodes only. Besides this, we also have an alternative for the loss function called "targeted attack" loss. The only difference between these two functions is that instead of just maximizing the loss of the correct class, the latter maximizes the correct class's loss while also minimizing the loss of the target class. From the distribution of labels (Figure 8), it is obvious that some classes such as 12 and 14 have just a few observations. We assume the distribution is similar in the test set. In other words, it is also true that nodes that belong to these classes in the 50,000 target nodes are rare. Thus, in “targeted attack” loss, these minor classes are chosen as the targets. We expect to see an increase in the predictions of minor labels, and that it can help to reduce the performance of the model further. It’s worth noting that these labels’ actual meanings are unknown to the analysts in this case.

The features of fake nodes are updated on their corresponding gradients. In updating the features of fake nodes, there are some strategies in this step as follows:

- Using Fast gradient sign method (FGSM) [37]
- Gradient attack [38]: instead of a fixed, large updates as in FGSM, in this method, we update the gradient by adding the product of its gradient and learning rate to the features. It’s similar to normal gradient descent, except we go with the direction of the gradient (not against it). Moreover, in this case, we need a large learning rate because we observed that the gradients are extremely small.
- Normalized Gradient attack [38]: Similar to the gradient attack mentioned above, but we normalize the gradient by its L2 norm vector.

For each structure attack, we apply a feature attack with a loss function to measure the result. To calculate the accuracy drop from the data provided, we create a test set in which the ground-truth can be accessed. It is worth noting that the attack will perform on robust defenders, which already have defense mechanisms. With that in mind, the choice of final attack strategy is not only based on the performance on the test set but also consider how likely the defender will detect it (i.e., using conservative attack strategies for unnoticeable perturbations).

Via experiments, we notice that other sophisticated structure attack approaches are not any better than the random method. Regarding the loss function, the predictions’ distribution fluctuated when using “targeted attack” loss. Furthermore, this loss function could not reduce the model’s performance much compared to the simple cross-entropy loss used in the final attack strategy. Thus, we first fix the structure attack by only getting 100 target nodes for each fake node.

Regarding feature attack, FGSM is likely to be detected by defenders because the features tend to lie at the extreme values (i.e., -1.73 or 1.62) as these seem to have higher attacking impacts. The defenders can easily ignore fake features by looking at the distribution and reshape these values (or using any other techniques to deal with outlying kind of values). We ended up using a Gradient attack, with a learning decay of 0.95, and a learning rate of 100,000. The fake nodes’ features are first initialized randomly by uniform distribution in the range of [-0.1, 0.1]. After 30 epochs, we got a good result, which can reduce the accuracy from 60% to 30% on the surrogate 2-layer GCN model generated for our attack strategies’ testing purposes. More importantly, the generated features also have a more similar distribution to the origin data.

V. Defender

A. Data exploration

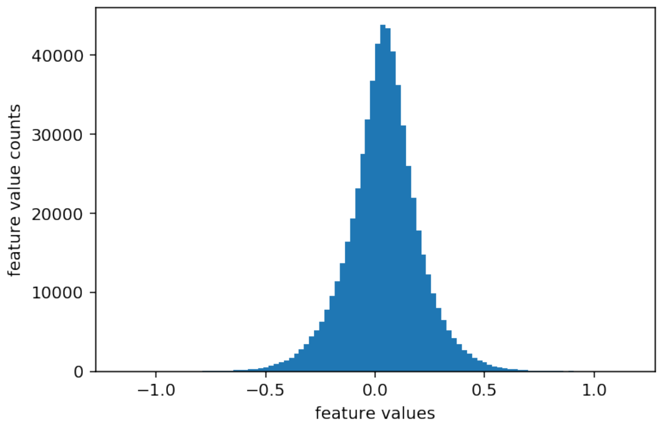

Fig. 6. Feature distribution of a sample feature (the first one of 100 features) from all the nodes. It’s observable that the feature values for the nodes are normalized to a standard normal distribution.

We first explored the data (graph node degrees, features, and class labels) before building prediction models and devising defending strategies. After exploring the features, it is observable that they are normalized using standard normalization (z-score normalization). For instance, Figure 6 shows the histogram describing the distributions of value counts over 100 bins for a sample feature (the first one of the 100 features). We further investigated the statistical descriptions of the values in all the features and found that they have *min*, *max*, *mean* values as approximately -1.735, 1.622, and -0.013, respectively. These feature descriptions help devise a strategy to increase model robustness by giving a constraint to the feature data as roughly in the range of -1.8 to 1.8. A conservative attacker should not have node feature values that exceed these boundaries. Therefore, we applied the following formulas to every input feature value (x):

$$x = ReLU(x + 1.8) - 1.8$$

$$x = 1.8 - ReLU(1.8 - x)$$

Furthermore, we also explored the characteristics of the features generated by our attack strategy, as described in Section IV. Even with the feature constraint in the specified range, it is likely that the attacking nodes have more values at the extreme borders (around -1.8 and 1.8), as shown in Figure 7. This skewed distribution might be explained by the fact that extreme values have higher attacking impacts. It’s worth noting from this figure that we know most of the attacking features are at the extremes because we know what our attacking nodes are. However, in the case of the data attacked by others, these nodes are unknown and are blended with the ground-truth values as long as they are in the boundary range (i.e., [-1.735, 1.622]). In other words, we cannot use the fact that most of the attacking nodes have values at the extreme and remove all the nodes with extreme values. Doing this also removes a

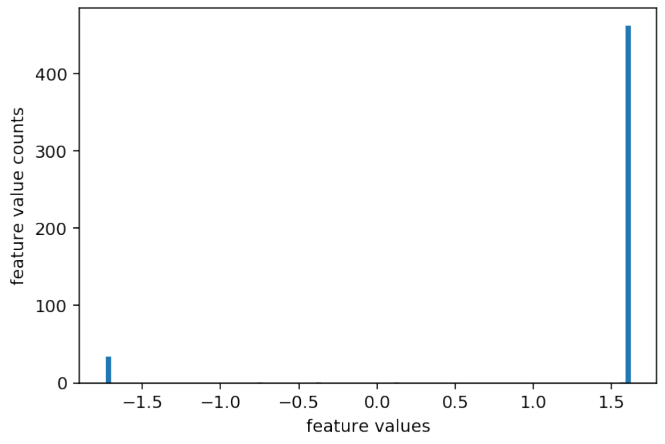

Fig. 7. Feature distribution of a sample feature (first feature, out of 100 embedded node features) from 500 attacking nodes. Most of the feature values are located at the two extremes (around -1.8 and +1.8).

large number of real nodes. Also, looking at these attacking data, one may devise strategies to detect these fake nodes as outliers. However, these strategies exceed the time constraint or only fit this specific attacking data (do not generalize well). Therefore, we decided to apply a normalization layer to the filtered data. This normalization layer helps to improve the robustness of the system, as explained in the later sections. Moreover, it helps the model to converge faster and achieve better learning time.

Regarding class labels, as shown in Figure 8, there are 19 labels in the dataset. It is observable that label 2 is missing, and there is an imbalance in the label counts. Heuristically, the more classes in the dataset, the more difficult the classification problem would be. Therefore, we decided to work with 18 available labels using their indexes (0 to 17).

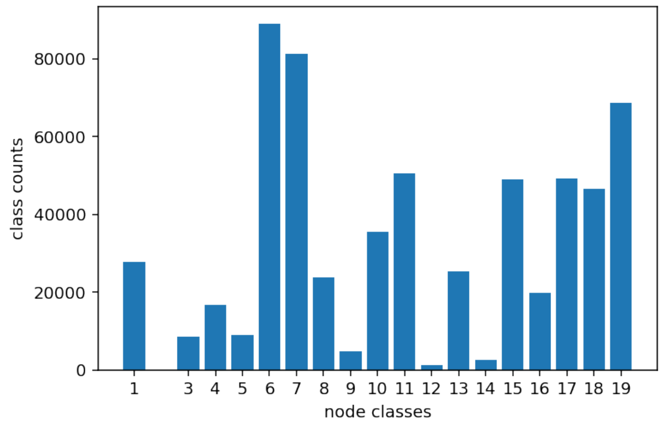

Fig. 8. Labels of the nodes in the dataset with 19 labels, imbalanced label counts, and label 2 is missing. It's worth noting that these labels' actual meanings are unknown to the analysts in this case.

Finally, we also explored the node degrees from the given adjacency matrix. Figure 9 (A) depicts the node degree counts in the log scale with a long tail. Specifically, many nodes have zero or few degrees while a few have very high degrees (the *min*, *max*, and *average* node degrees are 0, 4,517, and 8, correspondingly). Furthermore, the log-log scale of the node degrees and node degree probabilities, as shown in Figure 9 (B), indicates that this graph follows the power-law, which could be exploited to increase the robustness of the prediction model. However, we keep this direction as a future work due to time limitations.

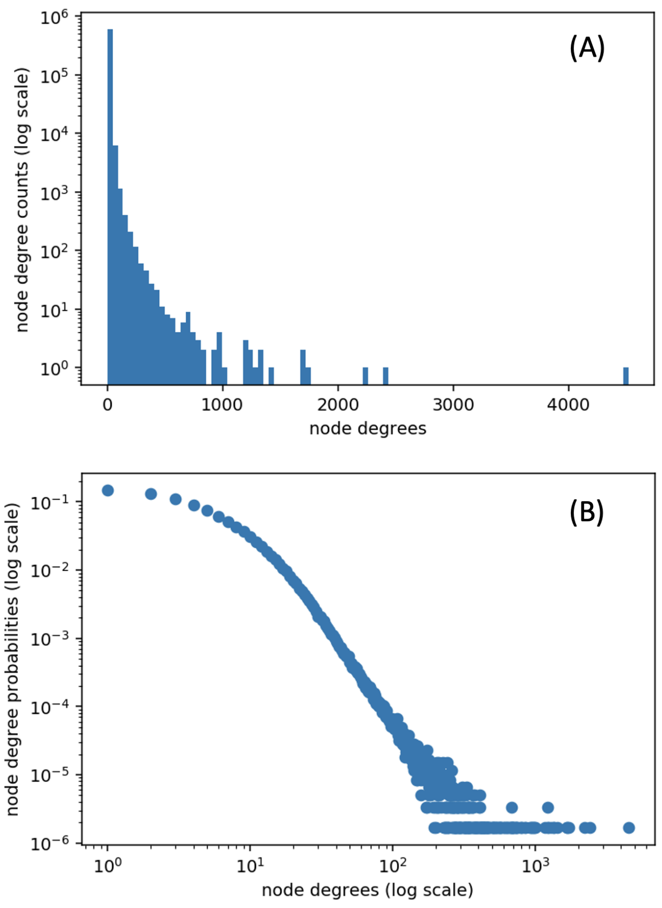

Fig. 9. Node degree distributions: Panel (A) shows the distributions of node degrees with a long tail; Panel (B), the log-log scale indicates that the graph follows the power-law.

B. Model selection

Methodologies: The main ideas for model selection are that we explore several standard model types for Graph Neural Networks using the given dataset. The data is split into 50% for training and another 50% for validation. In other words, we select the model that performs the best on the validation set with early stopping. Also, we utilize our attacking data generated as described in Section IV to check how vulnerable our models are to attacks. Since we do not have labels for the target nodes of the attacks (assumed to be the last 50,000 nodes in the given dataset), we measure the vulnerability by the percentage of nodes predicted differently with and without having the attacking nodes in the graph.

Model types: The simplest approach to this problem is training Multilayer Perceptrons (MLP), Random Forest, or more advanced models to make predictions. This approach has the advantage of being robust to structure attacks since it does not take the graph structure into account while predicting. However, the accuracy of this approach is low. Recent advancements in graph analysis offer more methods with higher accuracy for this task. Specifically, GCN [14] and GraphSAGE [15] are two popular methods in Graph Neural Networks [24]. GCN is a fast and efficient approach to learn and encode graphs into embeddings automatically. However, this approach is vulnerable to attacks if the data is not preprocessed carefully [20], [39]. On the other hand, GraphSAGE uses the sampling approach to saves memory and increases the robustness of the prediction models [32]. There are also many other different approaches surveyed in this work [24]. However, due to time limitations, we decided to explore GCN and GraphSAGE in this project.

Besides having high accuracy, another the primary purpose of the defending model is to be robust. Therefore, we started the model selection process with GraphSAGE models. After several experiments, we selected the architecture with the number of samples as 10, 5, and 3, for the first, second, and third levels of neighbors correspondingly. This GraphSAGE architecture archives an accuracy of approximately 64% on the validation set.

The advantages of GraphSAGE gained at the cost of sacrificing time efficiency and a fraction of the accuracy. Regarding accuracy, it has a slightly lower accuracy compared to the GCN model (discussed later in this section). Concerning the prediction time, it takes a long time (in terms of hours) to complete the sampling, calculating embeddings, and finally giving predictions for this reasonably large dataset. Therefore, though the robustness of the model is tempting, the time constraint (as 10s limitation for the prediction specified in this specific case) stopped us from further exploring the GraphSAGE approach.

This dataset's graph size (as opposed to the gradients of the adjacency matrix as in the attacking case) is suitable to fit in the memory (thanks to the sparse matrix format). Therefore, GCN is fast and efficient in this case. Figure 10 summaries the selected architecture after our experiments. It consists of a filter and a normalization layer. These two layers help improve the system's robustness by removing the impacts of the attacking features, which are too different from the original feature values. Three GCN layers follow these layers with 256, 128, and 64 hidden nodes correspondingly. Finally, the last GCN layer contains 18 hidden units corresponding to 18 labels of the nodes.

This selected GCN architecture has an accuracy of approximately 68% on the validation set (slightly higher than the GraphSAGE approach). More importantly, it takes a much shorter time (in terms of a few seconds) to complete the class predictions for all the nodes in the given dataset, making it suitable regarding prediction time constraints. Furthermore, as discussed earlier, to evaluate the vulnerability, we used our

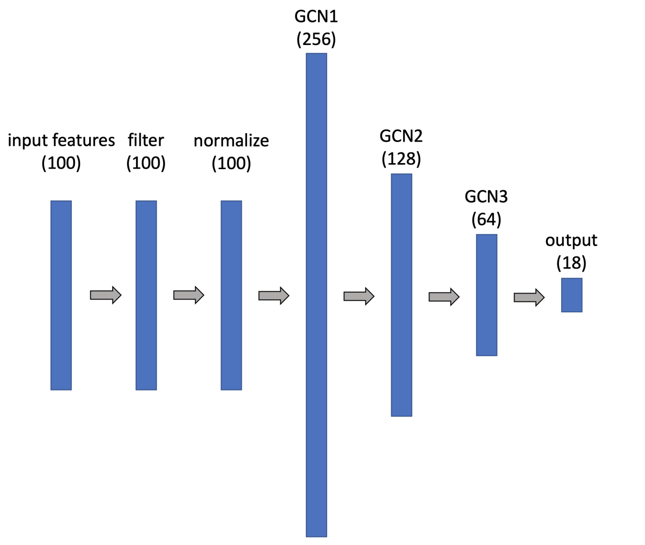

Fig. 10. Summaries of the selected GCN architecture with layer types and corresponding hidden nodes. It's worth noting that the filter layer is to reshape the input values into the range [-1.8, 1.8], and the normalization layer helps to stabilize the learning and increase the robustness of the model.

generated model to predict the labels for the target nodes (the last 50,000) with and without having the fake nodes in the graph. The resulted difference is 89%, which means that the attacking data could change 11% of our accuracy. Also, it's worth noting that without the normalization layer (as discussed earlier), the difference is 58%. This difference means the attacking data could reduce about half of our accuracy for the targeting nodes if we do not use the normalization layer.

As discussed earlier, the MLP model with predictions based on the original node features without taking the graph structure into account is robust to attacks. Therefore, we decided to explore the combination of MLP and GCN with the hope that the resulted model can make the predictions from the feature data to increase the robustness (via the MLP layers) and can still keep the accuracy whenever possible (via the GCN layers). As shown in Figure 11, after filtering and normalizing, the data is branched to two more MPL layers (with 128 and 64 hidden nodes, respectively). The output from GCN and MLP layers are then combined (at the orange arrow) and fed into another linear layer (with 64 hidden nodes) before the final output layer. The accuracy of this approach is 67% (slightly lower than the GCN only approach). Regarding vulnerability based on our attacking data, the prediction results between having and not having the fake nodes in the graph are 27% the same (for the 50,000 targeting nodes), which means that the attack could alter 73% of the targeting labels. Therefore, this GCN and MLP combination approach does not work in this case.

Finally, a recent work [17] assumes that contributions of neighboring nodes to the central node are neither identical (like GraphSAGE) nor predetermined (like GCN). Thus, they proposed a GAT (Graph Attention Network) method that adopts attention mechanisms to learn the relative weights between two connected nodes. Besides, more powerful GNNs

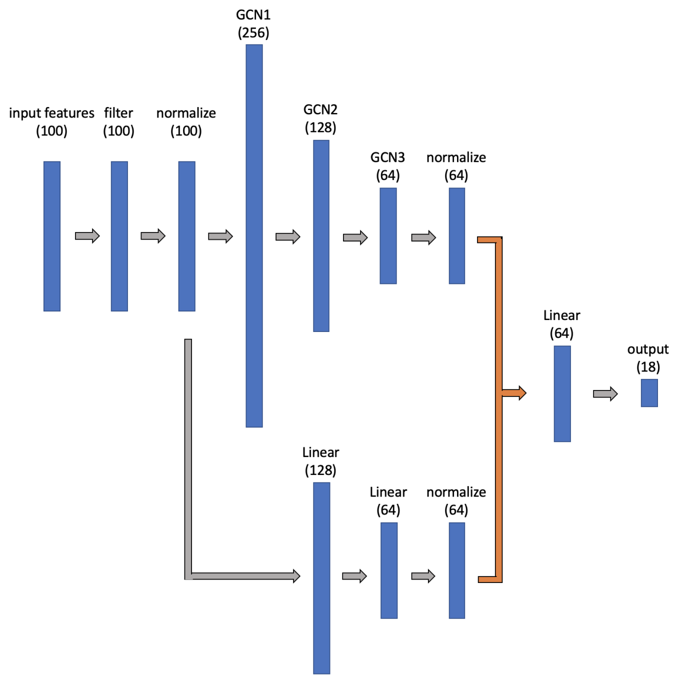

Fig. 11. The explored model with the combination of MLP and GCN. After filtering and normalizing, the input data branch out (to MLP and GCN stack of layers), then the outputs of these stacks are merged again (at the orange arrows) before the last two fully connected layers.

such as GIN [19] would be an auspicious candidate. These methods are promising and can be used in the defense model. However, they also take time to perform the classification on the whole large-scale graph. Thus, it does not satisfy the specified time constraint. Therefore, the final selected model for this specific case is the GCN approach, as described in Figure 10.

VI. Implementation and Results

The source code for our attacking and defending models are available at the Github page of the project at https://github.com/iDataVisualizationLab/GAAD. Also, our models achieve a score of 0.635 as calculated by (II) and ranked at the 7th place out of 16 winners list (among 411 single-player and 42 multi-player teams respectively) [21].

VII. Conclusion and Future Work

This work proposes strategies to attack and defend a large citation graph with memory and time constraints under a black-box scenario. Specifically, for attacking, we first build a surrogate GCN model, then try to attack the graph structure first by assigning each 100 target nodes to a fake node randomly. Then, with the structure fixed, we attack the node features by using the gradient-based method. We aim to reduce the model prediction performance significantly and constraint the fake features to have a similar node value distribution to the origin data. On the other hand, our defending model is based on GCN due to its high accuracy and fast prediction. To avoid vulnerability, we carefully pre-process the input data using a value filter and a normalization layer. Also, we validated the selected model based on our own attack strategies.

In future work, we aim to explore different strategies on structure attack, such as using multi-level fake nodes in which some "root" fake nodes connect indirectly to the target nodes through other fake nodes. In addition, it is worth to conduct experiments on combining some different loss functions in feature attack. For defense, GAT and GIN model types with a strategy of adversarial training and power-law of the node degrees should be the next directions for devising a more robust model. Besides, we would like to delve deeper into normalization techniques and their defense abilities to further improve the robustness of GNNs.

References

[1] D. Zhou, L. Zheng, J. Han, and J. He, "A data-driven graph generative model for temporal interaction networks," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 401–411.

[2] V. Pham, V. T. N. Nguyen, and T. Dang, "DualNetView: Dual Views for Visualizing the Dynamics of Networks," in *EuroVis Workshop on Visual Analytics (EuroVA)*, C. Turkay and K. Vrotsou, Eds. The Eurographics Association, 2020.

[3] Y. Bai, H. Ding, S. Bian, T. Chen, Y. Sun, and W. Wang, "Simgnn: A neural network approach to fast graph similarity computation," in *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, 2019, pp. 384–392.

[4] H. G. Patsolic, Y. Park, V. Lyzinski, and C. E. Priebe, "Vertex nomination via seeded graph matching," *Statistical Analysis and Data Mining: The ASA Data Science Journal*, vol. 13, no. 3, pp. 229–244, 2020.

[5] D. Sussman, Y. Park, C. E. Priebe, and V. Lyzinski, "Matched filters for noisy induced subgraph detection," *IEEE transactions on pattern analysis and machine intelligence*, 2019.

[6] D. J. D. S. Price, "Networks of scientific papers," *Science*, pp. 510–515, 1965.

[7] M. C. Shekar and J. A. Cottam, "Graph generation with a focusing lexicon," in *2019 IEEE International Conference on Big Data (Big Data)*. IEEE, 2019, pp. 4928–4931.

[8] Q. Wang, M. Li, X. Wang, N. Parulian, G. Han, J. Ma, J. Tu, Y. Lin, H. Zhang, W. Liu *et al.*, "Covid-19 literature knowledge graph construction and drug repurposing report generation," *arXiv preprint arXiv:2007.00576*, 2020.

[9] D. Zhou, L. Zheng, Y. Zhu, J. Li, and J. He, "Domain adaptive multi-modality neural attention network for financial forecasting," in *Proceedings of The Web Conference 2020*, 2020, pp. 2230–2240.

[10] V. Pham, N. V. Nguyen, and T. Dang, "Scagcnn: Estimating visual characterizations of 2d scatterplots via convolution neural network," in *Proceedings of the 11th International Conference on Advances in Information Technology*, ser. IAIT2020. New York, NY, USA: Association for Computing Machinery, 2020. [Online]. Available: https://doi.org/10.1145/3406601.3406644

[11] S. Zhang, H. Tong, J. Xu, and R. Maciejewski, "Graph convolutional networks: a comprehensive review," *Computational Social Networks*, vol. 6, no. 1, p. 11, 2019.

[12] Y. Rong, T. Xu, J. Huang, W. Huang, H. Cheng, Y. Ma, Y. Wang, T. Derr, L. Wu, and T. Ma, "Deep graph learning: Foundations, advances and applications," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 3555–3556.

[13] W. L. Hamilton, R. Ying, and J. Leskovec, "Representation learning on graphs: Methods and applications," *arXiv preprint arXiv:1709.05584*, 2017.

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

[15] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in neural information processing systems*, 2017, pp. 1024–1034.

[16] J. Chen, T. Ma, and C. Xiao, "FastGCN: Fast learning with graph convolutional networks via importance sampling," in *International Conference on Learning Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=rytstxWAW

[17] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," in *International Conference on Learning Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=rJXMpikCZ

[18] W.-L. Chiang, X. Liu, S. Si, Y. Li, S. Bengio, and C.-J. Hsieh, "Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, ser. KDD '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 257–266. [Online]. Available: https://doi.org/10.1145/3292500.3330925

[19] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" *arXiv preprint arXiv:1810.00826*, 2018.

[20] D. Zügner, A. Akbarnejad, and S. Günnemann, "Adversarial attacks on neural networks for graph data," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018, pp. 2847–2856.

[21] KDD Cup 2020, "Kdd cup 2020 on graph adversarial attacks and defense," https://www.biendata.xyz/competition/kddcup_2020, accessed: 2020–10-07.

[22] M. Gori, G. Monfardini, and F. Scarselli, "A new model for learning in graph domains," in *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, vol. 2, July 2005, pp. 729–734 vol. 2.

[23] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and locally connected networks on graphs," *arXiv preprint arXiv:1312.6203*, 2013.

[24] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[25] L. Zhang and W. Tu, "Six degrees of separation in online society," 2009.

[26] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[27] B. Weisfeiler and A. A. Lehman, "A reduction of a graph to a canonical form and an algebra arising during this reduction," *Nauchno-Technicheskaya Informatsia*, vol. 2, no. 9, pp. 12–16, 1968.

[28] Q. Le and T. Mikolov, "Distributed representations of sentences and documents," in *International conference on machine learning*, 2014, pp. 1188–1196.

[29] N. Reimers and I. Gurevych, "Sentence-bert: Sentence embeddings using siamese bert-networks," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 11 2019. [Online]. Available: https://arxiv.org/abs/1908.10084

[30] D. Zügner, A. Akbarnejad, and S. Günnemann, "Adversarial attacks on neural networks for graph data," https://www.in.tum.de/fileadmin/w00bws/daml/nettack/kdd_talk.pdf, 2018, accessed: 2020–11-06.

[31] H. Wu, C. Wang, Y. Tyshetskiy, A. Docherty, K. Lu, and L. Zhu, "Adversarial examples on graph data: Deep insights into attack and defense," *arXiv preprint arXiv:1903.01610*, 2019.

[32] X. Wang, J. Eaton, C. Hsieh, and S. F. Wu, "Attack graph convolutional networks by adding fake nodes," *CoRR*, vol. abs/1810.10751, 2018. [Online]. Available: http://arxiv.org/abs/1810.10751

[33] N. Entezari, S. A. Al-Sayouri, A. Darvishzadeh, and E. E. Papalexakis, "All you need is low (rank) defending against adversarial attacks on graphs," in *Proceedings of the 13th International Conference on Web Search and Data Mining*, 2020, pp. 169–177.

[34] H. Zhang, G. Wu, and Q. Ling, "Distributed stochastic gradient descent for link prediction in signed social networks," *EURASIP Journal on Advances in Signal Processing*, vol. 2019, no. 1, p. 3, 2019.

[35] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad, "Collective classification in network data," *AI magazine*, vol. 29, no. 3, pp. 93–93, 2008.

[36] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," in *Advances in Neural Information Processing Systems*, 2017, pp. 3146–3154.

[37] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *arXiv preprint arXiv:1412.6572*, 2014.

[38] A. Kolter, Zico; Madry, "Adversarial robustness - theory and practice," https://adversarial-ml-tutorial.org/, accessed: 2020–10-07.

[39] A. Kurakin, I. Goodfellow, S. Bengio, Y. Dong, F. Liao, M. Liang, T. Pang, J. Zhu, X. Hu, C. Xie *et al.*, "Adversarial attacks and defences competition," in *The NIPS'17 Competition: Building Intelligent Systems*. Springer, 2018, pp. 195–231.