ABSTRACT

Graph neural networks (GNNs) have shown broad applicability in a variety of domains. Some of these domains, such as social networks and product recommendations, are fertile ground for malicious users and behavior. In this paper, we show that GNNs are vulnerable to the extremely limited scenario of a single-node adversarial example, where the node cannot be picked by the attacker. That is, an attacker can force the GNN to classify any target node to a chosen label by only slightly perturbing another single arbitrary node in the graph, even when not being able to pick that specific attacker node. When the adversary is allowed to pick a specific attacker node, the attack is even more effective. We show that this attack is effective across various GNN types (e.g., GraphSAGE, GCN, GAT, and GIN), across a variety of real-world datasets, and as a targeted and non-targeted attack. Our code is available anonymously at https://github.com/gnnattack/SINGLE.

1 INTRODUCTION

Graph neural networks (GNNs) (Scarselli et al., 2008; Micheli, 2009) have recently shown sharply increasing popularity due to their generality and computation-efficiency (Duvenaud et al., 2015; Li et al., 2016; Kipf & Welling, 2017; Hamilton et al., 2017; Veličković et al., 2018; Xu et al., 2019b). Graph-structured data underlie a plethora of domains such as citation networks (Sen et al., 2008), social networks (Leskovec & McAuley, 2012; Ribeiro et al., 2017; 2018), knowledge graphs (Wang et al., 2018; Trivedi et al., 2017; Schlichtkrull et al., 2018), and product recommendations (Shchur et al., 2018). Therefore, GNNs are applicable for a variety of real-world structured data.

While most work in this field has focused on improving the accuracy of GNNs and applying them to a growing number of domains, only a few past works have explored the vulnerability of GNNs to adversarial examples. Consider the following scenario: a malicious user joins a social network such as Twitter or Facebook. The malicious user mocks the behavior of a benign user, establishes connections with other users, and submits benign posts. After some time, the user submits a new adversarially crafted post, which might seem irregular but overall benign. Since the GNN represents every user according to all the user’s posts, this new post perturbs the representation of the user as seen by a GNN. As a result, another, specific benign user gets blocked from the network; alternatively, another malicious user submits a hateful post – but does not get blocked. This scenario is illustrated in Figure 1. In this paper, we show the feasibility of such a troublesome scenario: a single attacker node can perturb its own representation, such that another node will be misclassified as a label of the attacker’s choice.

Most previous work on adversarial examples in GNNs required the perturbation to span multiple nodes, which in reality requires the cooperation of multiple attackers. For example, the pioneering work of Zügner et al. (2018) perturbed a set of attacker nodes; Bojchevski & Günnemann (2019a) perturb edges that are covered by a set of nodes. Further and in contrast with existing work, we show that perturbing a single node is more harmful than perturbing a single edge.

In this paper, we present a first a single-node adversarial attack on graph neural networks. If the adversary is allowed to choose the attacker node, for example, by hacking into an existing account, the efficiency of the attack significantly increases. We present two approaches for choosing the attacker: a white-box gradient-based approach, and a black-box, model-free approach that relies on graph topology. Finally, we perform a comprehensive experimental evaluation of our approach on multiple datasets and GNN architectures.
Our vision.
not racist

We associate each node \( v \) where the dataset contains multiple graphs, the split into training and test sets is between graphs, and where the dataset contains a single graph \( G \).

The total number of layers \( L \) is usually determined empirically as a hyperparameter. In the node classification scenario, we use the final representation of the node \( v \) to classify \( v \).

For brevity, we focus our definitions on the semi-supervised transductive node classification goal, where the dataset contains a single graph \( G \), and the split into training and test sets is across nodes in the same graph. Nonetheless, these definitions can be trivially generalized to the inductive setting, where the dataset contains multiple graphs, the split into training and test sets is between graphs, and the test nodes are unseen during training.

We associate each node \( v \in V \) with a class \( y_v \in \mathcal{Y} = \{1, \ldots, Y\} \). The labels of the training nodes are given during training; the test nodes are seen during training – without their labels. The training subset is represented as \( \mathcal{D} = \left( G, \{(v_i, y_i)\}_{i=0}^{N_D} \right) \). Given the training set, the goal is to learn a model \( f_\theta : (G, V) \to \mathcal{Y} \) that will classify the rest of the nodes correctly. During training, the model \( f_\theta \) thus minimizes the loss over the given labels, using \( J(\cdot, \cdot) \), which typically is the cross-entropy loss:

\[
\theta^* = \arg\min_\theta \mathcal{L}(f_\theta, \mathcal{D}) = \arg\min_\theta \frac{1}{N_D} \sum_{i=0}^{N_D} J(f_\theta(G, v_i), y_i)
\]
3 Single-Node GNN Attack

In this section, we describe our Single-node IIndirect Gradient adversarial Evasion (SINGLE) attack. While our attack is simple, it is the first attack that focuses on perturbing nodes (in contrast to edges (Dai et al., 2018), which works with an arbitrary single attacker node (in contrast to multiple nodes (Zügner et al., 2018)) that is not the node under attack (in contrast to “direct” attacks where the attacker perturbs the node under attack directly (Zügner et al., 2018; Li et al., 2020)).

3.1 Problem Definition

Given a graph $G$, a trained model $f_\theta$, a “victim” node $v$ from the test set along with its classification by the model $\hat{y}_v = f_\theta(G, v)$, we assume that an adversary controls another node $a$ in the graph. The goal of the adversary is to modify its own feature vector $x_a$ by adding a perturbation vector $\eta \in \mathbb{R}^D$ of its choice, such that the model’s classification of $v$ will change.

We denote by $G_{x_a+\eta}$ the graph $G$ where the row of $X$ that corresponds to the node $a$ was added with the vector $\eta$. In a non-targeted attack, the goal of the attacker is to find a perturbation vector $\eta$ that will change the classification to any other class, i.e., $f_\theta(G_{x_a+\eta}, v) \neq f_\theta(G, v)$. In a targeted attack, the adversary chooses a specific label $y_{adv} \in \mathcal{Y}$ and the adversary’s goal is to force $f_\theta(G_{x_a+\eta}, v) = y_{adv}$.

Generally, the classification of a node $v$ depends only on nodes whose distance to $v$ in the graph is lower than or equal $L$ – the number of GNN layers. Thus, a modification of the features of $a$ will affect the classification of $v$ only if the distance between $a$ and $v$ is lower than or equal $L$. Otherwise, $a$ will not be contained in the receptive field of $v$, and the attack will result in “under-reaching” (Alon & Yahav, 2020) – any perturbation of $a$ will not affect the prediction of $v$ (Barceló et al., 2020). Therefore, we require that $\text{distance}_G(a, v) \leq L$.

In this work, we focus on gradient-based attacks. These kinds of attacks assume that the attacker can access a similar model to the model under attack and compute gradients. As recently shown by Wallace et al. (2020), this is reasonable assumption: an attacker can query the original model; using these queries, imitate the model under attack by training an imitation model; find adversarial examples using the imitation model; and transfer these adversarial examples back to the original model. Under this assumption, these attacks are general and are applicable to any GNN and dataset.

3.2 Challenges

Unnoticeable Perturbations. Our first challenge is to find an adversarial example that will allow an imperceptible perturbation of the input. This objective is attainable in continuous domains such as images (Szegedy et al., 2013; Goodfellow et al., 2014) and audio (Carlini & Wagner, 2018) if we constrain $\ell_\infty$-norm of the perturbation vector $\eta$. It is, however, unclear what imperceptibility means in graphs. In most GNN datasets, a node’s features are a bag-of-words representation of the words that are associated with the node. For example, in Cora (McCallum et al., 2000; Sen et al., 2008), every node is annotated by a many-hot feature vector of words that appear in the paper; in PubMed (Namata et al., 2012), node vectors are TF-IDF word frequencies; in Twitter (Ribeiro et al., 2017), node features are averages of GloVe embeddings, which can be viewed as word frequency vectors multiplied by a (frozen) embedding matrix. We argue that an attack would be unnoticeable in an academic paper or in a set of Tweets if the frequency of some words is slightly modified. For example, a particular word may be repeated a few times throughout the text or remain unused.

To constrain the $\eta$ vector, we require that $\|\eta\|_\infty \leq \epsilon_\infty$ – the maximal absolute value of the elements in the perturbation vector – is bounded by $\epsilon_\infty \in \mathbb{R}^+$.

Perturbing nodes instead of edges. Previous work mostly focused on perturbing graph edges. Zügner et al. (2018) perturb both edges and node features, but conclude that “perturbations in the structure lead to a stronger change in the surrogate loss compared to feature attacks”; Wu et al. (2019b) also conclude that “perturbing edges is more effective than modifying the features”. In this paper, we counter these conclusions and show that small node feature perturbations are stronger: (i) first, removing all the edges of a particular node is a special case of node feature perturbation. There exists a perturbation $\eta$ such that $W^{-1}(x_a + \eta) = 0$, i.e., the modified feature vector $x_a + \eta$ is in the null
We evaluate and analyze the effectiveness of our SINGLE encoded attack. In previous work (Dai et al., 2018; Bojchevski & Günnemann, 2019b; Zhang & Zitnik, 2020). (iii) Finally, when a successful attack is caused by removing edges, it is unclear whether the misclassification is caused by sensitivity to non-robust features in the data (Ilyas et al., 2019), or simply due to smaller amount of information. Similarly, when a successful attack is caused by inserting edges, it is unclear whether this is simply due to incorrect or unrealistic added information.

3.3 Finding the Perturbation Vector

To find the perturbation, we iteratively differentiate the desired loss of \( v \) with respect to the perturbation vector \( \eta \), update \( \eta \) according to the gradient, and add it to the feature vector. In non-targeted attacks, we take the positive gradient of the loss of the undesired label to increase the loss; in targeted attacks, we take the negative gradient of the loss of the adversarial label \( y_{adv} \):

\[
\eta^{t+1} = \begin{cases} 
\eta^t + \gamma \nabla \eta J(f_\theta(G_{x_a+\eta^t}, v), y_v) & \text{non-targeted attack} \\
\eta^t - \gamma \nabla \eta J(f_\theta(G_{x_a+\eta^t}, v), y_{adv}) & \text{targeted attack}
\end{cases}
\]

where \( \gamma \in \mathbb{R}^+ \) is a learning rate. We repeat this process for a predefined number of \( K \) iterations, or until the model predicts the desired label.

Enforcing the constraints. We treat the node features as continuous throughout the attack iterations, whether they are discrete or continuous. Once the attack succeeds, we try to reset to zero as many perturbation vector elements as possible. We sort the perturbation vector elements in a decreasing order, according to their absolute value; \( i_1, ..., i_D \). We start with the index of \( \eta \) whose absolute value is the largest, \( \eta_{i_1} \), and reset the rest of the \( \{i_2, ..., i_D\} \) elements to zero. We then check whether perturbing only the \( i_1 \) index is sufficient. If the attack succeeds, we stop. If the attack fails (because of the large number of perturbation vector elements set to zero), we continue perturbing the rest of the elements of \( \eta \). In the worst case, we perturb all \( D \) vector elements of \( \eta \). In most cases, we stop much earlier, practically perturbing only a small fraction of the vector elements. If the original node features are discrete, we discretized features after the optimization.

Differentiate by frequencies, not by embeddings. When taking the gradient with respect to the perturbation vector \( \nabla \eta \), there is a subtle, but crucial, difference between the way that node representations are given in the dataset: (a) indicative datasets provide initial node representations \( X = [x_1, x_2, ...] \) that are word indicator vectors (many-hot) or frequencies such as (weighted) bag-of-words (Sen et al., 2008; Shchur et al., 2018); (b) in encoded datasets, initial node representations are given encoded, e.g., as an average of word2vec vectors (Hamilton et al., 2017; Hu et al., 2020). Indicative datasets can be converted to encoded by multiplying every vector by an embedding matrix; encoded datasets cannot be converted to indicative, without the authors releasing the textual data that was used to create the encoded dataset.

In indicative datasets, a perturbation of a node vector can be realized as a perturbation of the original text from which the indicative vector was derived. That is, adding or removing words in the text can result in the perturbed node vector. In contrast, a few-indices perturbation in encoded datasets might be an effective attack, but will not be realistic because there is no perturbation of the original text that will result in the perturbation of the vector. That is, when perturbing nodes, it is crucial to use indicative datasets, or convert encoded datasets to the indicative representation from which they were derived (as we do in Section 4) using their original text.

4 Evaluation

We evaluate and analyze the effectiveness of our SINGLE attack. In Section 4.1, we show that SINGLE is more effective than alternatives such as single-edge attacks. In Section 4.2, we show that if we are allowed to choose the attacker node, SINGLE is significantly more effective.

Setup. Our implementation is based on PyTorch Geometric (Fey & Lenssen, 2019) and its provided datasets. We trained each GNN type with two layers \( (L = 2) \), using the Adam optimizer, early

\footnote{This equation demonstrates GCN, but similar equations hold for other GNN types like GAT and GIN.}
Table 1: Test accuracy (lower is better) under different types of attacks, when the attacker node is chosen randomly. Performed using GCN, $\epsilon_{\infty} = 1$ for the discrete datasets (Cora and CiteSeer), and $\epsilon_{\infty} = 0.1$ for the continuous datasets (PubMed and Twitter).

<table>
<thead>
<tr>
<th>Attack Type</th>
<th>Cora</th>
<th>CiteSeer</th>
<th>PubMed</th>
<th>Twitter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clean (no attack)</td>
<td>80.5 ± 0.8</td>
<td>68.5 ± 0.7</td>
<td>78.5 ± 0.6</td>
<td>89.1 ± 0.2</td>
</tr>
<tr>
<td>EdgeGrad</td>
<td>65.1 ± 1.3</td>
<td>48.15 ± 0.9</td>
<td>59.7 ± 0.7</td>
<td>82.7 ± 0.0</td>
</tr>
<tr>
<td>SINGLE</td>
<td>60.1 ± 0.1</td>
<td>34.0 ± 3.6</td>
<td>45.5 ± 0.5</td>
<td>72.1 ± 7.2</td>
</tr>
<tr>
<td>SINGLE-hops</td>
<td>69.3 ± 0.9</td>
<td>45.1 ± 5.2</td>
<td>48.7 ± 0.9</td>
<td>74.5 ± 6.7</td>
</tr>
</tbody>
</table>

4.1 Main Results

Table 1 shows our main results for non-targeted attacks across various datasets. As shown, SINGLE is more effective than EdgeGrad across all datasets. SINGLE-hops, which is more unnoticeable than attacking with a neighbor node, performs almost as good as SINGLE which attacks using a non-neighboring node, and better than EdgeGrad. On Twitter, SINGLE reduces the test accuracy significantly better than EdgeGrad: 72.1% compared to 82.7%. Results for targeted attacks are shown in Appendix A.3.

Surprisingly, Table A.5 shows that Robust GCN (Zügner & Günnemann, 2019) is as vulnerable to the SINGLE attack as a standard GCN, showing that there is still much room for novel ideas and improvements to the robustness of current GNNs.

As we explain in Section 3.3, SINGLE tries to find a perturbation vector in which the number of perturbed elements is minimal. We measured the number of vector elements that the attack had improved to the robustness of current GNNs.

4.2 Baselines

In SINGLE (Section 3.3) the attacker node is selected randomly for each victim node, and the attack perturbs this node’s features according to $\epsilon_{\infty}$. SINGLE-hops is a modification of SINGLE where the attacker node is sampled only among nodes that are not neighbors, i.e., the attacker and the victim are not directly connected ($a, v \notin E$). We compare to additional approaches from the literature: EdgeGrad follows most previous work (Xu et al., 2019a; Li et al., 2020; Zügner & Günnemann, 2020): EdgeGrad randomly samples an attacker node as in SINGLE, and either inserts or removes a single edge from or to the attacker node, according to the gradient. If both use a randomly selected attacker node, EdgeGrad is strictly stronger than the GradArgmax attack of Dai et al. (2018), which only removes edges. We ran each approach 5 times with different random seeds for each dataset, and report the mean and standard deviation.

Baseline. In SINGLE (Section 3.3) the attacker node is selected randomly for each victim node, and the attack perturbs this node’s features according to $\epsilon_{\infty}$. SINGLE-hops is a modification of SINGLE where the attacker node is sampled only among nodes that are not neighbors, i.e., the attacker and the victim are not directly connected ($a, v \notin E$). We compare to additional approaches from the literature: EdgeGrad follows most previous work (Xu et al., 2019a; Li et al., 2020; Zügner & Günnemann, 2020): EdgeGrad randomly samples an attacker node as in SINGLE, and either inserts or removes a single edge from or to the attacker node, according to the gradient. If both use a randomly selected attacker node, EdgeGrad is strictly stronger than the GradArgmax attack of Dai et al. (2018), which only removes edges. We ran each approach 5 times with different random seeds for each dataset, and report the mean and standard deviation.

This can be implemented easily using edge weights: training the GNN with weights of 1 for existing edges, adding all possible edges with weights of 0, and taking the gradient with respect to the vector of weights.

Appendix A.2, we show consistent results across additional GNN types: GAT (Veličković et al., 2018), GIN (Xu et al., 2019a), GraphSAGE (Hamilton et al., 2017), SGC (Wu et al., 2019a), and RobustGCN (Zügner & Günnemann, 2019).

Data. We used Cora and CiteSeer (Sen et al., 2008) which are discrete datasets, i.e., the given node feature vectors are many-hot vectors. Thus, we set $\epsilon_{\infty} = 1$, the minimal possible perturbation. We also used PubMed (Sen et al., 2008) and the Twitter-Hateful-Users (Ribeiro et al., 2017) datasets, which are continuous, and node features represent frequencies of words. Continuous datasets allow a much more subtle perturbation, and we set $\epsilon_{\infty} = 0.1$. An analysis of these values is presented in Section 4.5.

The Twitter-Hateful-Users dataset is originally provided as an encoded dataset, where every node is an average of GloVe vectors (Pennington et al., 2014). We reconstructed this dataset using the original text from Ribeiro et al. (2017), to be able to compute gradients with respect to the weighted histogram of words, rather than the embeddings. We took the most frequent 10,000 words as node features, and used GloVe-Twitter embeddings to multiply by the node features. We thus converted this dataset to indicative rather than encoded. Statistics of all dataset are provided in the supplementary material.

Baselines. In SINGLE (Section 3.3) the attacker node is selected randomly for each victim node, and the attack perturbs this node’s features according to $\epsilon_{\infty}$. SINGLE-hops is a modification of SINGLE where the attacker node is sampled only among nodes that are not neighbors, i.e., the attacker and the victim are not directly connected ($a, v \notin E$). We compare to additional approaches from the literature: EdgeGrad follows most previous work (Xu et al., 2019a; Li et al., 2020; Zügner & Günnemann, 2020): EdgeGrad randomly samples an attacker node as in SINGLE, and either inserts or removes a single edge from or to the attacker node, according to the gradient. If both use a randomly selected attacker node, EdgeGrad is strictly stronger than the GradArgmax attack of Dai et al. (2018), which only removes edges. We ran each approach 5 times with different random seeds for each dataset, and report the mean and standard deviation.

4.1 Main Results

Table 1 shows our main results for non-targeted attacks across various datasets. As shown, SINGLE is more effective than EdgeGrad across all datasets. SINGLE-hops, which is more unnoticeable than attacking with a neighbor node, performs almost as good as SINGLE which attacks using a non-neighboring node, and better than EdgeGrad. On Twitter, SINGLE reduces the test accuracy significantly better than EdgeGrad: 72.1% compared to 82.7%. Results for targeted attacks are shown in Appendix A.3.

Surprisingly, Table A.5 shows that Robust GCN (Zügner & Günnemann, 2019) is as vulnerable to the SINGLE attack as a standard GCN, showing that there is still much room for novel ideas and improvements to the robustness of current GNNs.

As we explain in Section 3.3, SINGLE tries to find a perturbation vector in which the number of perturbed elements is minimal. We measured the number of vector elements that the attack had...
perturbed in practice. In PubMed, SINGLE used 76 vector elements on average, which are 15% of the elements in the feature vector. In Cora, SINGLE perturbed 717 elements on average, which are 50%. In CiteSeer, SINGLE used 1165 attributes on average, which are 31% of the features. In Twitter, SINGLE used 892 attributes on average, which are 9% of the features. In the experiments shown in Table 1, we used $\epsilon_{\infty} = 0.1$ in the continuous datasets (PubMed and Twitter). If we allow larger values of $\epsilon_{\infty}$, we can reduce the number of perturbed vector elements: using $\epsilon_{\infty} = 0.5$ requires perturbing only 3% of the attributes on average to achieve the same effectiveness; using $\epsilon_{\infty} = 1$ requires perturbing only 1.6% of the attributes on average to achieve the same effectiveness (in PubMed, where varying $\epsilon_{\infty}$ is meaningful).

4.2 Attacker Choice

If the attacker could choose its node, e.g., by hijacking an existing account in a social network, could they increase the effectiveness of the attack? We examine the effectiveness of two approaches for choosing the attacker node.

Gradient Attacker Choice (GradChoice) chooses the attacker node according to the largest gradient with respect to the node representations (for a non-targeted attack): $a^* = \arg\max_{a_i \in V} \|\nabla_{a_i} J(f_\theta(G, v), \hat{y}_v)\|_{\infty}$. The chosen attacker node is never the victim node itself.

Topological Attacker Choice (Topology) chooses the attacker node according to topological properties of the graph. As an example, we choose the neighbor of the victim node $v$ with the smallest number of neighbors: $a^* = \arg\min_{a \in N_v} |N_a|$. The advantage of this approach is that the attacker choice is model-free: if the attacker cannot compute gradients, they can at least choose the most harmful attacker node, and then perform the perturbation itself using other non-gradient approaches such as those proposed by Waniek et al. (2018) and Chang et al. (2020).

To perform a fair comparison, we compare these approaches with GlobalEdgeGrad, which is similar to EdgeGrad that can insert or remove an edge, with the difference that the chosen edge can be chosen from the entire graph.

**Results.** Results for these attacker choice approaches are shown in Table 2. The main results are that choosing the attacker node significantly increases the effectiveness of the SINGLE attack: for example, in Twitter, from 72.1% (Table 1) to 6.6% test accuracy (Table 2).

In datasets where the given initial node features are continuous (PubMed and Twitter), SINGLE+GradChoice and SINGLE+Topology show similar results: on Twitter accuracy difference is less than 0.5%; on PubMed SINGLE+Topology outperforms SINGLE+GradChoice by $\sim$3%, even though SINGLE+Topology is model-free. Both of those attacks are more efficient than GlobalEdgeGrad, showing the superiority of node perturbation over edge perturbation in the global view. In Appendix A.4, we show that allowing GlobalEdgeGrad to insert and remove multiple edges that belong to the same attacker node does not lead to a significant improvement.

Interestingly, GradChoice and Topology agree on the choice of attacker node for 50.3% of the nodes in Cora, 78.7% of the nodes in CiteSeer, 51.0% of the nodes in PubMed, and on 55.0% of the nodes in Twitter, showing that the node selection can sometimes be performed model-free.

In datasets where the initial node features are discrete (Cora and CiteSeer), i.e., many-hot vectors, GlobalEdgeGrad reduces the test accuracy more than GradChoice and Topology. We believe that the reason is the difficulty of two-step optimization in discrete datasets: for example, GradChoice needs to choose the node, and find the perturbation afterwards. Finding a perturbation for a discrete vector is more difficult than in continuous datasets, and the choice of the attacker node may not be optimal.

<table>
<thead>
<tr>
<th></th>
<th>Cora</th>
<th>CiteSeer</th>
<th>PubMed</th>
<th>Twitter</th>
</tr>
</thead>
<tbody>
<tr>
<td>GlobalEdgeGrad</td>
<td>29.7 ± 2.4</td>
<td>11.9 ± 0.8</td>
<td>15.3 ± 0.4</td>
<td>82.7 ± 0.0</td>
</tr>
<tr>
<td>SINGLE+GradChoice</td>
<td>31.0 ± 1.9</td>
<td>19.0 ± 4.2</td>
<td>8.5 ± 1.2</td>
<td>7.0 ± 1.1</td>
</tr>
<tr>
<td>SINGLE+Topology</td>
<td>31.1 ± 1.2</td>
<td>18.1 ± 3.4</td>
<td>5.2 ± 0.1</td>
<td>6.6 ± 0.5</td>
</tr>
</tbody>
</table>

Table 2: Test accuracy when the adversary can choose the attacker node.
As shown in Table 4, adversarial training indeed improves the model’s robustness against the different types of attacks. However, the main result of this section is that adversarial training was the most efficient by Zügner et al. (2018). Table 3 shows the test accuracy of these ablations. In the previous sections, we studied the effectiveness of the SINGLE attack. In this paper, we focus on in this paper is a SINGLE approach that always perturbs a single node, which is not the victim node (a ̸= v). We now examine our SINGLE attack in other, easier but less realistic, scenarios: SINGLE-two attackers follows Zügner et al. (2018) and Zang et al. (2020), randomly samples two attacker nodes and perturbs their features using the same approach as SINGLE. SINGLE-direct perturbs the victim node directly (i.e., a = v), an approach that was found to be the most efficient by Zügner et al. (2018). Table 3 shows the test accuracy of these ablations. In Appendix A.5.3, we additionally experiment with more than two attacker nodes.

### 4.3 Scenario Ablation

The main scenario that we focus on in this paper is a SINGLE approach that always perturbs a single node, which is not the victim node (a ̸= v). We now examine our SINGLE attack in other, easier but less realistic, scenarios: SINGLE-two attackers follows Zügner et al. (2018) and Zang et al. (2020), randomly samples two attacker nodes and perturbs their features using the same approach as SINGLE. SINGLE-direct perturbs the victim node directly (i.e., a = v), an approach that was found to be the most efficient by Zügner et al. (2018). Table 3 shows the test accuracy of these ablations. In Appendix A.5.3, we additionally experiment with more than two attacker nodes.

### 4.4 Adversarial Training

In the previous sections, we studied the effectiveness of the SINGLE attack. In this section, we investigate to what extent can adversarial training (Madry et al., 2018) defend against SINGLE. For each training step and labeled training node, we perform $K_{\text{train}}$ adversarial steps to adversarially perturb another randomly sampled node, exactly as in SINGLE, but at training time. The model is then trained to minimize the original cross-entropy loss and the adversarial loss:

$$L(f_\theta, D) = \frac{1}{2N_D} \sum_{i=0}^{N_D} \left(J(f_\theta(G_i, v_i), y_i) + J(f_\theta(G_{x_i + \eta_i}, v_i), y_i)\right).$$

The main difference from Equation (3) is the adversarial term $J(f_\theta(G_{x_i + \eta_i}, v_i), y_i)$, where $a_i$ is the randomly sampled attacker for the node $v_i$. In every training step, we randomly sample a new attacker for each victim node and compute new $\eta_i$ vectors. After the model is trained, we attack the model with $K_{\text{test}}$ SINGLE adversarial steps. This is similar to Feng et al. (2019) and Deng et al. (2019), except that they used adversarial training as a regularizer, to improve the accuracy of a model while not under attack. In contrast, we use adversarial training to defend a model against an attack at test time. We used $K_{\text{train}} = 5$, as we found it to be the maximal value for which the model’s accuracy is not significantly hurt while not under attack (“clean”), and $K_{\text{test}} = 20$ as in the previous experiments. As shown in Table 4, adversarial training indeed improves the model’s robustness against the different SINGLE attacks. However, the main result of this section is that SINGLE, SINGLE+GradChoice and SINGLE+Topology are still very effective attacks, as they succeed in attacking the adversarially trained model, reducing its test accuracy to 58.5%, 30.6% and 21.1%, respectively.

### 4.5 Sensitivity to $\epsilon_\infty$

How does the intensity of the adversarial perturbation affect the performance of the attack? Intuitively, we say that the less we restrict the perturbation (i.e., larger values of $\epsilon_\infty$), the more powerful the attack. We examine whether this holds in practice.
In our experiments in Sections 4.1 to 4.4, we used $\epsilon_\infty = 0.1$ for the continuous datasets (PubMed and Twitter). In this section, we vary the value of $\epsilon_\infty$ across different GNN types and observe the effectiveness of the attack. Figure 2 shows the results on PubMed. We used this dataset because it is larger than Cora and CiteSeer (Appendix A.1), and most importantly, its features are continuous, thus real-valued perturbations are feasible. As shown in Figure 2, the most significant difference is between performing the perturbation ($\epsilon_\infty = 0.1$) and not attacking at all ($\epsilon_\infty = 0$). As we increase the value of $\epsilon_\infty$, GCN and GraphSage (Hamilton et al., 2017) show a natural descent in test accuracy. Contrarily, GAT (Veličković et al., 2018) and GIN (Xu et al., 2019b) are more robust to increased absolute values of perturbations, while GAT is also the most robust compared to the other GNN types.

4.6 Distance Between Attacker and Victim

In Section 4.1, we found that SINGLE performs similarly to SINGLE-hops, although SINGLE-hops samples an attacker node $a$ whose distance from the victim node $v$ is at least 2. We further question whether the effectiveness of the attack depend on the distance in the graph between the attacker and the victim. We trained a new model for each dataset using $L = 8$ layers. Then, for each test victim node, we sampled attackers according to their distance to the test node.

As shown in Figure 3, the effectiveness of the attack increases as the distance between the attacker and the victim decreases. At distance of 5, the curve seems to saturate. A possible explanation for this is that apparently more than few layers (e.g., $L = 2$ in Kipf & Welling (2017)) are not needed in most datasets. Thus, the rest of the layers can theoretically learn not to pass much of their input starting from the redundant layers, excluding adversarial signals as well.

5 Related Work

Works on adversarial attacks on GNN differ in several main aspects. In this section, we discuss the main criteria, to clarify the settings that we address.

Single vs. multiple attackers All previous works allowed perturbing multiple nodes, or edges that are covered by multiple nodes: Zügner et al. (2018) perturb features of a set of attacker nodes; Zang et al. (2020) assume “a few bad actors”; other works perturb edges that in realistic settings their perturbation would require controlling multiple nodes (Bojchevski & Günnemann, 2019a; Sun et al., 2020; Chen et al., 2018).

Node vs. edge perturbations Most adversarial attacks on GNNs perturb the input graph by modifying the graph structure (Zügner & Günnemann, 2019; Wang et al., 2020; Xu et al., 2019a). For example, Dai et al. (2018) iteratively remove edges, yet their attack manages to reduce the accuracy by about 10% at most when perturbing a single edge. Li et al. (2020) also allow the insertion of edges; Waniek
et al. (2018) and Chang et al. (2020) allow insertion and deletion of edges, using attacks that are based on correlations and eigenvalues, and not on gradients. Yefet et al. (2019) perturb one-hot node vectors, in the restricted domain of computer programs. Zügner et al. (2018) and Wu et al. (2019b) perturb both edges and nodes; but they concluded that perturbing edges is more effective than perturbing nodes. In this work, we counter these conclusions and show that perturbing node features is more effective than perturbing edges.

**Direct vs. influence attacks** Another difference between prior works lies in the difference between direct attacks and influence attacks. In direct attacks, the attacker perturbs the target node itself. For example, the attack of Zügner et al. (2018) is the most effective when the attacker and the target are the same node. In influence attacks, the perturbed nodes are at least one hop away from the victim node. In this paper, we show that the strong direct assumption is not required (SINGLE-direct in Section 4.2), and that our attack is effective when the attacker and the target are not even direct neighbors, i.e., they are at least two hops away (SINGLE-hops in Section 4.1).

**Poisoning vs. evasion attacks** In a related scenario, some work (Zügner & Günnemann, 2019; Bojchevski & Günnemann, 2019a; Li et al., 2020; Zhang & Zitnik, 2020) focuses on poisoning attacks that perturb examples before training. Contrarily, we focus on the standard evasion scenario of adversarial examples in neural networks (Szegedy et al., 2013; Goodfellow et al., 2014), where the attack operates at test time, after the model was trained, as Dai et al. (2018).

**Attacking vs. certifying** Zügner & Günnemann (2020) focus on certifying the robustness of GNNs against adversarial perturbations; and Bojchevski & Günnemann (2019b) certified PageRank-style models. In contrast, we study the effectiveness of the adversarial attack itself.

## 6 Conclusion

We demonstrate that GNNs are susceptible even to the extremely limited scenario of a single-node indirect adversarial example (SINGLE). The practical consequences of these findings are that a single attacker in a network can force a GNN to classify any other target node as the attacker’s chosen label, by slightly perturbing some of the attacker’s features. We further show that if the attacker can choose its attacker node – the effectiveness of the attack increases significantly. We study the effectiveness of these attacks across various GNN types and datasets.

We believe that this work will drive research in this field toward exploring novel defense approaches for GNNs. Such defenses can be crucial for real-world systems that are modeled using GNNs. Furthermore, we believe that the surprising results of this work motivate better theoretical understanding of the expressiveness and generalization of GNNs. To these ends, we make all our code and trained models publicly available.

## References


Fuli Feng, Xiangnan He, Jie Tang, and Tat-Seng Chua. Graph adversarial training: Dynamically regularizing based on graph structure. *IEEE Transactions on Knowledge and Data Engineering*, 2019.


A Supplementary Material

A.1 Dataset Statistics

Statistics of the datasets are shown in Table A.1.

<table>
<thead>
<tr>
<th></th>
<th>#Training</th>
<th>#Val</th>
<th>#Test</th>
<th>#Unlabeled Nodes</th>
<th>#Classes</th>
<th>Avg. Node Degree</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cora</td>
<td>140</td>
<td>500</td>
<td>1000</td>
<td>2708</td>
<td>7</td>
<td>3.9</td>
</tr>
<tr>
<td>CiteSeer</td>
<td>120</td>
<td>500</td>
<td>1000</td>
<td>3327</td>
<td>6</td>
<td>2.7</td>
</tr>
<tr>
<td>PubMed</td>
<td>60</td>
<td>500</td>
<td>1000</td>
<td>19717</td>
<td>3</td>
<td>4.5</td>
</tr>
<tr>
<td>Twitter</td>
<td>4474</td>
<td>248</td>
<td>249</td>
<td>95415</td>
<td>2</td>
<td>45.6</td>
</tr>
</tbody>
</table>

A.2 Additional GNN Types

Tables A.2 to A.4 present the test accuracy of different attacks applied on GAT (Veličković et al., 2018), GIN (Xu et al., 2019b), GraphSAGE (Hamilton et al., 2017), RobustGCN (Zügner & Günnemann, 2019), and SGC (Wu et al., 2019a), showing the effectiveness of SINGLE across different GNN types.

<table>
<thead>
<tr>
<th></th>
<th>Cora</th>
<th>CiteSeer</th>
<th>PubMed</th>
</tr>
</thead>
<tbody>
<tr>
<td>EdgeGrad</td>
<td>66.4 ± 1.2</td>
<td>49.4 ± 1.4</td>
<td>64.9 ± 1.0</td>
</tr>
<tr>
<td>SINGLE</td>
<td>40.0 ± 1.5</td>
<td>33.2 ± 6.7</td>
<td>35.7 ± 13.3</td>
</tr>
<tr>
<td>SINGLE-hops</td>
<td>42.0 ± 1.5</td>
<td>41.7 ± 5.8</td>
<td>35.5 ± 13.6</td>
</tr>
<tr>
<td>GlobalGradEdge</td>
<td>67.8 ± 4.9</td>
<td>48.3 ± 5.1</td>
<td>63.5 ± 4.6</td>
</tr>
<tr>
<td>SINGLE+GradChoice</td>
<td>43.1 ± 4.9</td>
<td>32.4 ± 4.7</td>
<td>36.4 ± 8.0</td>
</tr>
<tr>
<td>SINGLE+Topology</td>
<td>32.2 ± 6.4</td>
<td>25.5 ± 8.0</td>
<td>27.8 ± 5.7</td>
</tr>
<tr>
<td>SINGLE-two attackers</td>
<td>12.7 ± 6.3</td>
<td>11.0 ± 1.4</td>
<td>26.8 ± 11.8</td>
</tr>
<tr>
<td>SINGLE-direct</td>
<td>23.6 ± 6.5</td>
<td>14.8 ± 4.3</td>
<td>21.8 ± 2.5</td>
</tr>
</tbody>
</table>

Table A.2: Test accuracy of GAT under different non-targeted attacks

<table>
<thead>
<tr>
<th></th>
<th>Cora</th>
<th>CiteSeer</th>
<th>PubMed</th>
</tr>
</thead>
<tbody>
<tr>
<td>EdgeGrad</td>
<td>32.9 ± 1.1</td>
<td>18.5 ± 3.0</td>
<td>33.3 ± 1.7</td>
</tr>
<tr>
<td>SINGLE</td>
<td>27.1 ± 1.3</td>
<td>12.3 ± 2.9</td>
<td>12.9 ± 1.0</td>
</tr>
<tr>
<td>SINGLE-hops</td>
<td>32.6 ± 0.7</td>
<td>18.5 ± 3.1</td>
<td>14.0 ± 0.6</td>
</tr>
<tr>
<td>GlobalGradEdge</td>
<td>10.7 ± 2.8</td>
<td>4.8 ± 2.1</td>
<td>10.3 ± 1.0</td>
</tr>
<tr>
<td>SINGLE+GradChoice</td>
<td>15.9 ± 2.0</td>
<td>8.1 ± 1.7</td>
<td>10.0 ± 1.6</td>
</tr>
<tr>
<td>SINGLE+Topology</td>
<td>16.1 ± 1.7</td>
<td>7.6 ± 1.6</td>
<td>6.3 ± 1.7</td>
</tr>
<tr>
<td>SINGLE-two attackers</td>
<td>2.7 ± 0.7</td>
<td>5.4 ± 2.0</td>
<td>6.2 ± 1.8</td>
</tr>
<tr>
<td>SINGLE-direct</td>
<td>5.7 ± 1.4</td>
<td>4.7 ± 1.6</td>
<td>3.1 ± 3.1</td>
</tr>
</tbody>
</table>

Table A.3: Test accuracy of GIN under different non-targeted attacks

Surprisingly, Table A.5 shows that Robust GCN (Zügner & Günnemann, 2019) is as vulnerable to the SINGLE attack as a standard GCN, showing that there is still much room for novel ideas and improvements to the robustness of current GNNs.

A.3 Targeted Attacks

Tables A.6 to A.9 show the results of targeted attacks across datasets and approaches. Differently from other tables which show test accuracy, Tables A.6 to A.9 present the targeted attack’s success rate, which is the fraction of test examples that the attack managed to force a specific label prediction.
Table A.4: Test accuracy of GraphSAGE under different non-targeted attacks

<table>
<thead>
<tr>
<th></th>
<th>Cora</th>
<th>CiteSeer</th>
<th>PubMed</th>
</tr>
</thead>
<tbody>
<tr>
<td>EdgeGrad</td>
<td>62.9 ± 1.9</td>
<td>45.9 ± 3.4</td>
<td>64.2 ± 1.6</td>
</tr>
<tr>
<td>SINGLE</td>
<td>62.7 ± 2.4</td>
<td>32.3 ± 4.3</td>
<td>57.1 ± 0.8</td>
</tr>
<tr>
<td>SINGLE-hops</td>
<td>70.0 ± 3.3</td>
<td>45.5 ± 4.3</td>
<td>60.9 ± 0.8</td>
</tr>
<tr>
<td>GlobalGradEdge</td>
<td>48.9 ± 2.7</td>
<td>40.4 ± 3.3</td>
<td>64.7 ± 1.1</td>
</tr>
<tr>
<td>SINGLE+GradChoice</td>
<td>37.3 ± 3.4</td>
<td>18.0 ± 3.2</td>
<td>8.2 ± 0.7</td>
</tr>
<tr>
<td>SINGLE+Topology</td>
<td>37.4 ± 3.6</td>
<td>19.2 ± 4.2</td>
<td>6.6 ± 0.3</td>
</tr>
<tr>
<td>SINGLE-two attackers</td>
<td>14.4 ± 0.9</td>
<td>11.1 ± 0.1</td>
<td>45.4 ± 0.8</td>
</tr>
<tr>
<td>SINGLE-direct</td>
<td>19.6 ± 2.1</td>
<td>13.5 ± 3.9</td>
<td>0.0 ± 0.1</td>
</tr>
</tbody>
</table>

Table A.5: Test accuracy of GCN, Robust GCN (Zügner & Günnemann, 2019), and SGC (Wu et al., 2019a) under different non-targeted attacks, on PubMed.

<table>
<thead>
<tr>
<th></th>
<th>Clean</th>
<th>RobustGCN</th>
<th>SGC</th>
</tr>
</thead>
<tbody>
<tr>
<td>EdgeGrad</td>
<td>78.5 ± 0.6</td>
<td>73.9 ± 1.6</td>
<td>78.9 ± 0.5</td>
</tr>
<tr>
<td>SINGLE</td>
<td>59.7 ± 0.7</td>
<td>–</td>
<td>65.1 ± 1.3</td>
</tr>
<tr>
<td>SINGLE+topology</td>
<td>45.5 ± 0.5</td>
<td>34.3 ± 1.4</td>
<td>47.3 ± 1.2</td>
</tr>
<tr>
<td>SINGLE+hops</td>
<td>48.7 ± 0.9</td>
<td>29.7 ± 1.1</td>
<td>49.6 ± 1.2</td>
</tr>
<tr>
<td>GlobalGradEdge</td>
<td>15.3 ± 0.4</td>
<td>–</td>
<td>15.3 ± 0.4</td>
</tr>
<tr>
<td>SINGLE+GradChoice</td>
<td>8.5 ± 1.2</td>
<td>19.6 ± 0.9</td>
<td>11.3 ± 1.2</td>
</tr>
<tr>
<td>SINGLE+Topology</td>
<td>5.2 ± 0.1</td>
<td>72.5 ± 1.9</td>
<td>5.6 ± 0.5</td>
</tr>
<tr>
<td>SINGLE-two attackers</td>
<td>27.7 ± 0.2</td>
<td>20.0 ± 1.1</td>
<td>30.0 ± 1.8</td>
</tr>
<tr>
<td>SINGLE-direct</td>
<td>0.3 ± 0.1</td>
<td>15.8 ± 1.1</td>
<td>0.5 ± 0.2</td>
</tr>
</tbody>
</table>

Table A.6: Success rate (higher is better) of different targeted attacks on a GCN network.

(in these results, higher is better). These results suggest that in targeted attack settings, node-based attacks (such as SINGLE) have an even bigger advantage over edge-based attacks (such as EdgeGrad).

<table>
<thead>
<tr>
<th></th>
<th>Cora</th>
<th>CiteSeer</th>
<th>PubMed</th>
</tr>
</thead>
<tbody>
<tr>
<td>EdgeGrad</td>
<td>6.1 ± 0.4</td>
<td>12.5 ± 1.2</td>
<td>17.9 ± 1.5</td>
</tr>
<tr>
<td>SINGLE</td>
<td>33.7 ± 8.6</td>
<td>43.5 ± 11.1</td>
<td>50.7 ± 15.8</td>
</tr>
<tr>
<td>SINGLE-indirect</td>
<td>26.6 ± 7.3</td>
<td>29.88 ± 8.6</td>
<td>50.3 ± 15.7</td>
</tr>
<tr>
<td>GlobalGradEdge</td>
<td>6.0 ± 1.4</td>
<td>14.6 ± 2.8</td>
<td>22.3 ± 3.6</td>
</tr>
<tr>
<td>SINGLE+GradChoice</td>
<td>25.8 ± 5.3</td>
<td>38.6 ± 8.5</td>
<td>50.5 ± 13.5</td>
</tr>
<tr>
<td>SINGLE+Topology</td>
<td>41.3 ± 5.3</td>
<td>52.5 ± 11.3</td>
<td>63.0 ± 10.2</td>
</tr>
</tbody>
</table>

Table A.7: Success rate (higher is better) of different targeted attacks on a GAT network.
A.4 MultiEdge attacks

We strengthened the EdgeGrad attack by allowing it to add and remove multiple edges that are connected to the attacker node – MultiEdgeGrad. Accordingly, MultiGlobalEdgeGrad is equivalent to GlobalEdgeGrad, except that MultiGlobalEdgeGrad can choose the attacker node.

As shown in Table A.10, allowing the attacker node to add and remove multiple edges (MultiEdgeGrad and MultiGlobalEdgeGrad) results in a very minor improvement compared to EdgeGrad and GlobalEdgeGrad, while SINGLE, SINGLE+Topology and SINGLE+GradChoice are much more effective.

A.5 Additional Baselines

A.5.1 Zero-features approach

We experimented with a baseline where we set $\eta = -x_a$ as the feature perturbation. The objective of experimenting with such an attack is to illustrate that SINGLE can find better perturbations than simply canceling the node feature vector, making the new vector a vector of zeros (and thus effectively removes the edges of the attacker node in GCN).

As shown, Zero features is barely effective (compared to “Clean”), and SINGLE can find much better perturbations.
We also study an additional type of a realistic attack that is based on node injection. In this approach, we insert a new node to the graph with a single edge attached to our victim node. The attack is performed by perturbing the injected node’s attributes. Since there is no initial node feature vector to measure the $\epsilon_\infty$ distance to, the injected node is allowed to find any realistic representation (e.g., without choosing negative frequencies). This attack is very powerful, reducing the test accuracy down to 0.02% on PubMed.

### A.5.3 Larger number of attackers

<table>
<thead>
<tr>
<th>Number of attackers</th>
<th>PubMed</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>45.5</td>
</tr>
<tr>
<td>2</td>
<td>27.7</td>
</tr>
<tr>
<td>3</td>
<td>19.0</td>
</tr>
<tr>
<td>4</td>
<td>15.3</td>
</tr>
<tr>
<td>5</td>
<td>12.9</td>
</tr>
</tbody>
</table>

Table A.12: Test accuracy for different number of attackers on PubMed.

We performed additional experiments with up to five randomly sampled attacker nodes simultaneously (Table A.12). As expected, allowing a larger number of attackers reduces the test accuracy. However, the main observation in this paper is that even a single attacker node is surprisingly effective.

### A.6 Limiting the allowed $\epsilon_0$

In Section 4.5, we analyzed the effect of the value of $\epsilon_\infty$, that is, the maximal allowed perturbation in each vector attribute, on the performance of the attack. However, in datasets such as Cora and CiteSeer, the input features are binary (i.e., the input node vector is many-hot), so the possible perturbation to each vector element is only “flipping” its value from zero to one, or vice-versa. Thus, in these datasets, it is interesting to analyze the value of $\epsilon_0$, the maximal number of allowed perturbed vector elements, on the performance of the attack. In this case, measuring the $l_0$ norm is equivalent to measuring the $l_1$ norm: $||\eta||_0 = ||\eta||_1$, and is proportional to the $l_2$ norm.

We performed experiments where we measured the test accuracy of the model while limiting the number of allowed perturbed vector elements. The results are shown in Figure A.1.

As shown, when $\epsilon_0 = 0$, no attack is allowed, and the test accuracy is equal to the “Clean” value of Table 1. When $\epsilon_0 = 100\%$, the results are equal to the SINGLE values of Table 1 – resulting in flipping 50% of the features on average in Cora, and 31% of the features on average in CiteSeer.

It is important to note that in practice, the average number of perturbed features is much lower than the maximal number of allowed features. For example, in CiteSeer, allowing 100% of the features results in actually using only 31% on average.
Maximal fraction of perturbed features

Acc

Figure A.1: Test accuracy compared to the maximal allowed $\epsilon_0$, the number of perturbed features (divided by the total number of features in the dataset). In practice, the average number of perturbed features is much lower than the maximal number of allowed features.