# Improving Robustness to Attacks Against Vertex Classification

Benjamin A. Miller
miller.be@husky.neu.edu
Northeastern University
Boston, MA

Mustafa Çamurcu
camurcu.m@husky.neu.edu
Northeastern University
Boston, MA

Alexander J. Gomez
gomez.alexa@husky.neu.edu
Northeastern University
Boston, MA

Kevin Chan
kevin.s.chan.civ@mail.mil
Army Research Laboratory
Adelphi, MD

Tina Eliassi-Rad
t.eliassirad@northeastern.edu
Northeastern University
Boston, MA

## ABSTRACT

Vertex classification—the problem of identifying the class labels of nodes in a graph—has applicability in a wide variety of domains. Examples include classifying subject areas of papers in citation networks or roles of machines in a computer network. Recent work has demonstrated that vertex classification using graph convolutional networks is susceptible to targeted poisoning attacks, in which both graph structure and node attributes can be changed in an attempt to misclassify a target node. This vulnerability decreases users' confidence in the learning method and can prevent adoption in high-stakes contexts. This paper presents work in progress aiming to make vertex classification robust to these types of attacks.

We investigate two aspects of this problem: (1) the classification model and (2) the method for selecting training data. Our alternative classifier is a support vector machine (with a radial basis function kernel), which is applied to an augmented node feature-vector obtained by appending the node's attributes to a Euclidean vector representing the node based on the graph structure. Our alternative methods of selecting training data are (1) to select the highest-degree nodes in each class and (2) to iteratively select the node with the most neighbors minimally connected to the training set. In the datasets on which the original attack was demonstrated, we show that changing the training set can make the network much harder to attack. To maintain a given probability of attack success, the adversary must use far more perturbations; often a factor of 2–4 over the random training baseline. Even in cases where success is relatively easy for the attacker, we show that the classification and training alternatives allow classification performance to degrade much more gradually, with weaker incorrect predictions for the attacked nodes.

## KEYWORDS

Graph convolutional networks, vertex classification, poisoning attacks, robust learning


*MLG'19, Aug 05, 2019, Anchorage, AK*


**ACM Reference Format:**
Benjamin A. Miller, Mustafa Çamurcu, Alexander J. Gomez, Kevin Chan, and Tina Eliassi-Rad. 2019. Improving Robustness to Attacks Against Vertex Classification. In *MLG'19: The 15th International Workshop on Mining and Learning with Graphs, Aug 05, 2019, Anchorage, AK*. ACM, New York, NY, USA, 8 pages.

## 1 INTRODUCTION

Classification of vertices in graphs is an important problem in a variety of applications, varying from e-commerce (classifying users for targeted advertising) to security (classifying computer nodes as malicious or not) to bioinformatics (classifying roles in a protein interaction network). In the past several years, numerous methods have been developed for this task (see, e.g., [9, 12]). Until recently, however, little attention has been paid to the robustness of these methods to adversarial actions such as data poisoning. If an adversary was able to insert data into the training set (e.g., generate benign traffic during a data collection period that could cover its behavior during testing/inference time), he/she would be more likely to succeed in successfully evading detection (which is undesirable to the data analyst).

To classify vertices in the presence of adversarial activity, we must implement learning systems that are robust to such potential manipulation. If such malicious behavior has low cost to the attacker and high cost to the data analyst, machine learning systems will not be trusted and adopted for use in practice, especially in high-stakes scenarios such as network security and traffic safety. Understanding how to achieve robustness is key to realizing the full potential of machine learning.

Adversaries, of course, will attempt to conceal their manipulation. In a recent paper, Zügner et al. propose an adversarial technique called Nettack [21], which can create perturbations that are subtle[1] while still being extremely effective in decreasing performance on the target vertices. The authors use their poisoning attack against a graph convolutional network (GCN). Nettack's effectiveness is notable, but the authors do not consider how the end user of the classifier might defend against it.

In this paper, we present work in progress suggesting that there are ways to make classifiers more robust to Nettack. We focus on two potential methods: changing the classifier and changing the training data selection technique. We consider a classification scheme in which topology and vertex attributes are decoupled

[1] We are not convinced by the subtlety of the perturbations in [21]. As Figure 6 shows Nettack's perturbations can be detected by examining triangle distributions.

and considered as separate features. We see that—while there is a tradeoff between overall performance and robustness—we can substantially increase the level of perturbation required to achieve the same decrease in the classification margin. To select training data, we aim to train with a subset of nodes that are well connected to the held-out set. Here we also see a benefit, often raising the number of perturbations required for a given level of attack success by a factor of 2 to 4. In a context where there is flexibility to choose the training set, this can provide a significant advantage. Some combination of these methods will likely be useful to develop a more robust vertex classification system.

The contributions of this work are as follows:

- We tested transferability of Nettack to classifiers when we decouple structure from attributes and observe varying levels of robustness.
- We developed a new method for selecting training data that puts a greater burden on attackers.
- We expand the GCN model used in [21] to use two hidden layers and test this with the training data selection method, also noting an increase in the required budget in most cases.
- We identified potential tradeoffs between overall classifier performance and robustness to attack.

These contributions all point toward interesting future research in this area, such as developing a broad set of attacks against vertex classification and subsequent research into robustness.

The remainder of this paper is organized as follows. In Section 2 we describe the vertex classification problem and the Nettack method. Section 3 details the methods we investigate to provide greater robustness. Section 4 documents several experimental results, identifying promising areas for continued investigation. In Section 5, we briefly contextualize our work within the current literature. In Section 6 we conclude with a summary and outline future work.

## 2 PROBLEM MODEL

We consider the vertex classification problem as described in [21], where we are given a graph $G = (V, E)$ of size $N = |V|$ and an $N \times d$ matrix of vertex attributes $X$. Each node has an arbitrary numeric index from 1 to $N$. For this work, as in [21], we consider only binary attributes. In addition to its $d$ attributes, each node has a label denoting its class. We enumerate classes as integers from 1 to $C$. Given a subset of labeled instances, the goal is to correctly classify the unlabeled nodes.

The focus of [21] is on GCNs, which make use of the adjacency matrix for the graph $A = \{a_{ij}\}$, where $a_{ij}$ is 1 if there is an edge between node $i$ and node $j$ and is 0 otherwise. The GCN applies a symmetrized graph convolution to the input layer. That is, if we let $D$ be the diagonal matrix of vertex degrees—i.e., the $i$th diagonal entry is the number of edges connected to vertex $i$, $d_{ii} = \sum_{j=1}^{N} a_{ij}$—then the first layer of the network is expressed as

$$H = \sigma\left(D^{-1/2}AD^{-1/2}XW_1\right),$$

where $W_1$ is a weight matrix, $X$ is a feature matrix whose $i$th row is $x_i^T$, and $\sigma$ is the rectifier function. From the hidden layer to the output layer, a similar graph convolution is performed, followed by a softmax output:

$$Y = \text{softmax}\left(D^{-1/2}AD^{-1/2}HW_2\right).$$

The focus in [21] is on GCNs with a single hidden layer. Each vertex is then classified according to the largest entry in the corresponding row of $Y$.

The vertex attack proposed in [21] operates on a surrogate model where the ReLU activation function is replaced by a linear function, thus approximating the overall network as

$$\begin{aligned} Y &\approx \text{softmax}\left(\left(D^{-1/2}AD^{-1/2}\right)^2 XW_1W_2\right) \\ &= \text{softmax}\left(\left(D^{-1/2}AD^{-1/2}\right)^2 XW\right). \end{aligned} \tag{1}$$

Nettack uses a greedy algorithm to determine how to subtly perturb both $A$ and $X$ to make the GCN misclassify a target node. The changes are intended to be "unnoticeable," i.e., the degree distribution of $G$ and the co-occurrence of features are changed negligibly. Using the approximation in (1), Nettack perturbs by either adding or removing edges or turning off binary features so that the classification margin is reduced the most at each step. Note that while it can change the topology and the features, Nettack does *not* change the labels of any vertices. An additional variation on Nettack allows either "direct" attacks, in which the target node itself has its edges and features changed, or indirect "influencer" attacks, where the neighbors of the target have their data altered.

The classifier is evaluated in a context where only some of the labels are known, and the labeled data are split into training and validation sets. To train the GCN, 10% of the data are selected at random (or by one of the alternative methods outlined in Section 3.2), and another 10% is selected for validation. The remaining 80% is the test data. After training, nodes are selected for attack among those that are correctly classified: the 10 where the margin is largest, the 10 where the margin is smallest, and 20 more at random. Each of these is taken as a target node for attack in an experiment. The attack is evaluated based on how much the classification margin decreases for the targeted nodes.

Taking the perspective of a defender against the attack, we want to ensure that this decrease in margin is minimized. The goal is to determine how to make the GCN (or an alternative vertex classifier) as robust as possible to this attack, minimizing the decrease in margin for a given level of perturbation by the attacker.

## 3 PROPOSED TECHNIQUES

### 3.1 Alternative Classification

Our first investigation of robustness to Nettack involves altering the classifier used by the defender. We still use the same attack as in [21] and apply to a different classifier. This is also done in [21], but we consider different alternative classifiers in a new context. In this case, we still classify based on both structural features and attributes, but decouple the two in a way that cannot be done with GCN.

In each case, we generate an $N \times d_e$ feature matrix $X_s$ based on the structure of the graph. We then concatenate these features to the attributes to obtain a new feature matrix $X_f = [X_s\ X]$, which is passed to a classifier.

We consider two graph embedding procedures, which map vertices to points in $\mathbb{R}^{d_e}$. The first is node2vec [8], which embeds each node in Euclidean space based on distance between nodes in a random walk. That is, the more often nodes $v_i, v_j \in V$ occur near each other in random walks, the smaller $\|f(v_i) - f(v_j)\|_2$ should be. The function $f(v)$ is the feature representation of vertex $v$ in the lower-dimensional space. Node2vec adapts the framework of word embedding to graphs. First, it finds "context" around each vertex $v \in V$ by conducting second-order biased random walks around the neighborhood of $v$. Let's denote this context by $N(v)$. Then, it maximizes the log-likelihood objective to find $f$: $\max_f \sum_{v \in V} \log \Pr(N(v)|f(v))$. In this way, $f(v)$ is predictive of the nodes in $v$'s neighborhood $N(v)$.

The other embedding technique we use is recursive feature extraction (ReFeX) [10]. This method starts with a base set of local features and iteratively aggregates the features of its neighbors until convergence. The base node features are degree, number of edges in the egonet, and number of edges from the egonet to the rest of the graph. At each iteration, ReFeX considers adding new features consisting of the sum and average of its neighbors features, only keeping them if they provide sufficient new information. We use the implementation provided in [6], which uses an alternative method to determine which features to keep (different from the original paper).

In the experiments in Section 4.1, we use a support vector machine (SVM) with a radial basis function kernel as our classifier. We use scikit-learn's implementation and enable probability outputs to evaluate the classification margins as we do with the GCN, for which we use the implementation provided by the authors of [21].

### 3.2 Alternative Training

The Nettack paper only considered one way of selecting the training set: a random sample (stratified across classes) of 20% of the data was taken, and split in half for training and validation, with the remaining 80% being used for testing. As we investigated classification performance, we noted that nodes in the test set with many neighbors in the training set were more likely to be correctly classified. This dependence on labeled neighbors is consistent with previous observations [14]. This suggested that a training set that provides something like a vertex cover—a kind of "scaffolding" for the data—could make the classification more robust.

We considered two methods to test this hypothesis. The first simply chooses the highest-degree nodes to be in the training set. The top 10% of each class is chosen for the training data, and the remaining 90% is randomly split (stratified by class) into test (80%) and validation (10%), maintaining the proportions of the original experiment in [21]. The other method uses a greedy approach in an attempt to ensure every node has at least a minimal number of neighbors in the training set. Starting with an empty training set and a threshold $k = 0$, we iteratively add the node with the largest number of neighbors connected to at most $k$ nodes in the training set. When there are no such neighbors, we increment $k$. This procedure continues until we have the desired proportion of the overall dataset for training (again, 10% in our experiments). The remaining data are randomly partitioned into test and validation sets. Algorithm 1 provides the pseudo-code. In this case, there is no stratification by class. Incorporating this aspect is part of our future work.

**Algorithm 1** GreedyCover

**Input:** Graph $G = (V, E)$, training proportion $t \in (0, 1)$
**Output:** Training set $T \subset V$
$k \leftarrow 0$, $T \leftarrow \emptyset$
**for all** $u \in V$ **do**
  $m_u \leftarrow 0$  ⟨⟨mark all nodes 0⟩⟩
**end for**
**while** $|T| < |V|t$ **do**
  $v \leftarrow \arg\max_{u \in V \setminus T} \sum_{u' \in \mathcal{N}(u)} \mathbb{I}[m_{u'} = k]$
  **if** $\sum_{u' \in \mathcal{N}(v)} \mathbb{I}[m_{u'} = k] = 0$ **then**
    $k \leftarrow k + 1$  ⟨⟨increment min. num. trained neighbors⟩⟩
  **else**
    $T \leftarrow T \cup \{v\}$
    $m_v \leftarrow -1$
    **for all** $u' \in \mathcal{N}(v) \setminus T$ **do**
      $m_{u'} \leftarrow m_{u'} + 1$
    **end for**
  **end if**
**end while**
**return** $T$

There are issues with both of these approaches. Since it adds edges to the graph, Nettack changes the degree of the nodes. The targets, however, are selected from among the test set (the nodes with "unknown" labels). These two aspects of the problem setup are in contention: selecting the target requires knowledge of the training set, but the training set selection requires knowledge of degree, which is not fully known until after the attack. This is an issue with the experimental setup that will need to be resolved to fully formalize this analysis. Given the results in Section 4.2, however, this strategy shows promise and is worth pursuing in greater depth.

## 4 EXPERIMENTAL RESULTS

We tested Nettack against the proposed defenses. In each case, we randomly select a test/validatiton split (after either deterministically or randomly selecting the training set), select 40 targets as discussed in Section 2, and perturb the graph either directly or indirectly to change the target's classification. We evaluate the perturbation by inspecting the classification margin for the target node. Specifically, the margins reported here are the log probability ratios for the correct class versus the highest probability incorrect class, i.e., if the true class is $c$,

$$\log \frac{p_c}{\max_{c' \neq c} p_{c'}}.$$

Thus, if the vertex is correctly classified, the margin will be positive; otherwise it will be negative.

For each target, we perturb some number of times. We do the following to determine the budget required for an attacker to achieve a given probability of success. First, compute the associated quantile of the distribution of margins across targets (e.g., 10th percentile for 10% attacker success probability) as a function of number of perturbations. We then determine where this function first becomes

nonpositive (i.e., the smallest number of perturbations for the attacker to successfully cause the target to be misclassified). This value is the required budget. We linearly interpolate between tested values to get better resolution. If a given level of attack success is never achieved within the perturbations attempted, we set the required budget to number of perturbations plus 1. We average this value over 5 trials and report standard errors in the budget plots in this section.

We use two of the datasets used in the Nettack paper in our experiments:

- **CiteSeer** The CiteSeer dataset has 3312 scientific publications put into 6 classes. The network has 4732 links representing citations between the publications. The features of the nodes contain 1s and 0s indicating the presence of the word in the paper. There are 3703 unique words considered for the dictionary.
- **Cora** The Cora dataset consists of 2708 machine learning papers classified into one of seven categories. The citation network consists of 5429 citations. For each paper (vertex) in the network there is a feature vector of 0s and 1s for whether it contains one of the 1433 unique words.

## 4.1 Classifier Variations

Our first test involves using an alternative classifier rather than a GCN. For both ReFeX and node2vec, we use 128 features for structure and select targets among nodes correctly classified *by the classifier being used*. We perturb each target up to 10 times. Only attacks against structure (not attributes) are used.

The budget required to achieve a certain probability of attack success is shown in Figure 1. Results are shown for indirect attacks against the randomly selected targets as opposed to those with large or small margins. We should note that the overall classification performance is much lower for ReFeX—about 50% as opposed to the 86% achieved by the GCN—but those nodes that are correctly classified are much more robust to Nettack than using the original GCN formulation. At low attack success probability rates, the required budget is increased by about an order of magnitude. This raises several questions. Since the overall classification performance with ReFeX is lower, are these cases easier if we consider them with the GCN as well? Is the robustness gap maintained if we consider more perturbations (note that the top of the plot is at 10, which is the maximum number of perturbations we consider in this experiment)? We are currently investigating these results more deeply, but are highly encouraged by what we have seen so far.

The node2vec-based method, on the other hand actually seems to impose a *smaller* burden on the attacker. While using the node2vec embedding did not make these cases more robust to the attack, we do see a change in performance if we look more closely at the data. In many cases, the margin of classification decreases much more slowly using the node2vec-based method. Results for such cases are shown in Figure 2. This figure shows results for cases where improving performance is more difficult: direct (rather than influence) attacks against the target, cases where the margin is small (and thus any attack immediately causes misclassification), and cases where the margin is large (where it is often not possible to misclassify within 10 perturbations). The figure shows the median (50% attack success probability) across all targets in all 5 trials. In most of these cases, while the target becomes misclassified using either the node2vec-based method or the GCN at around the same number of perturbations, the decline in the margin is typically steeper for the GCN. This means that the classifier will be less confident in its incorrect predictions of the targeted nodes for a given budget. Note that this occurs because the initial classification margin for the node2vec-based method is smaller than with the GCN. As we continue this work, we will determine whether additional training can yield higher margins for the SVM to improve this method. Regardless, we see the reduction of the rate of decline as positive and worth pursuing further. Note that the results using ReFeX are virtually unchanged by the perturbations.

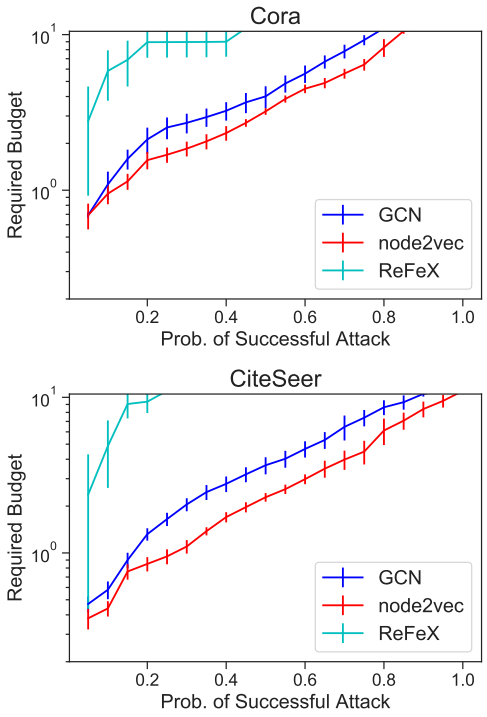


**Figure 1: Budget required to achieve a given probability of attack success, varying the classifier used. Results are shown for influencer attacks against structure only, for both the Cora and CiteSeer datasets. Targets are randomly chosen from test nodes correctly classified with the given classifier. Note that nodes correctly classified by applying an SVM to the ReFeX representation require substantially more perturbation—sometimes nearly an order of magnitude—to achieve a given level of attack success.**

## 4.2 Training Variations

In our experiments varying the training selection method, we also considered adding another hidden layer. In this case, Nettack uses a different approximation of the output, modifying (1) to obtain

$$Y \approx \text{softmax}\left(\left(D^{-1/2}AD^{-1/2}\right)^3 XW'\right),$$

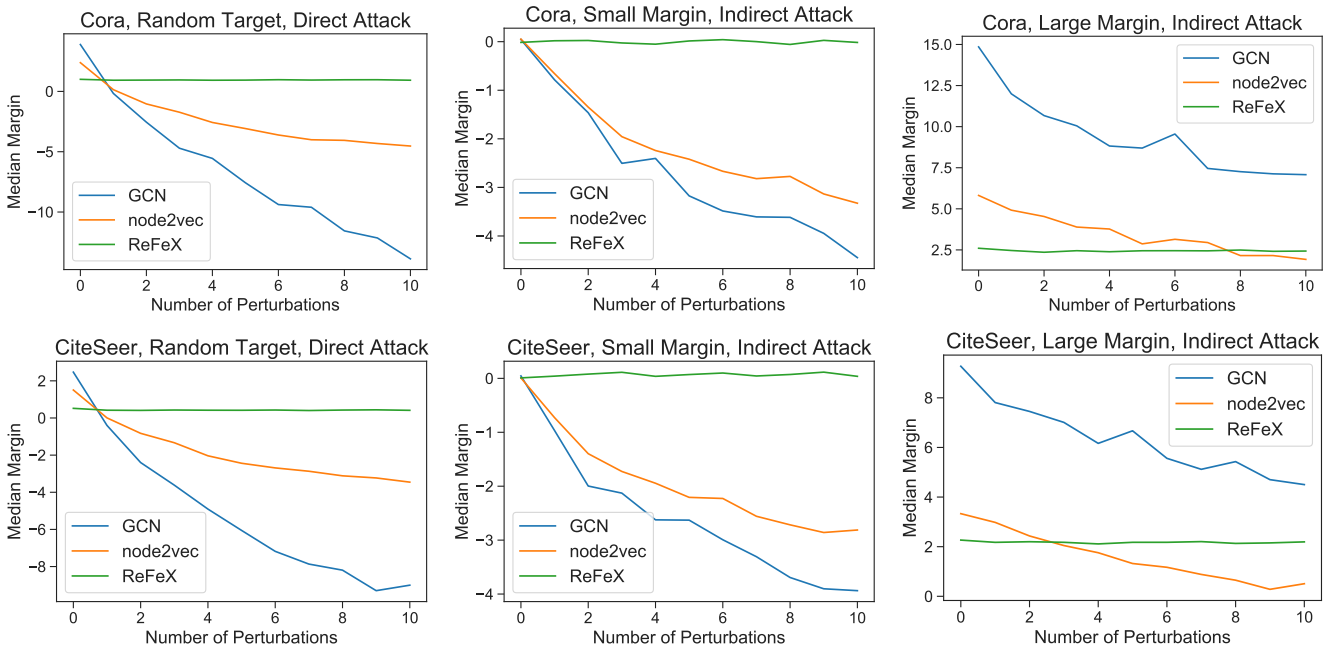


**Figure 2: Median margin after perturbation, varying the classifier, in cases that are more difficult to improve: direct attacks against random targets (left column), small margin cases (center column), and large margin cases (right column). The GCN becomes much more confident in its incorrect predictions than the SVM applied to node2vec features. Performance with an SVM applied to ReFeX features is virtually unchanged as more perturbations are applied.**

where in this case $W'$ is the product of 3 matrices representing the input layer and two hidden layers. This increases computational complexity as Nettack must consider additional paths (3 hops rather than 2) as it identifies the most promising perturbations. Both hidden layers have 16 units, as in the single hidden layer in the previous experiments. In these experiments, we apply up to 50 perturbations and attack either structure, attributes, or both.

We see that, across a wide swath of attack success probabilities, the alternative selection methods provide a significant increase in robustness, often requiring a factor of 2–4 to achieve a given probability of success. Results demonstrating the increase in required budget are shown in Figure 3. This plot includes influencer attacks against randomly selected targets. Consider the attacks against structure alone. Random selection sees little difference in using one or two hidden layers for either dataset. For GreedyCover, we see a significant increase in the required budget at high attack success probabilities with two layers, while selecting high-degree nodes is better with one layer when attack success probability is low. In almost all cases, performance using the alternatives is appreciably better than randomly selecting training data.

The improvement in performance is not universal across scenarios. When attributes alone are attacked, for example, the classifier is just as robust using random training as the other methods. Understanding this phenomenon is a goal of our ongoing work.

Just as when we considered varying the classifier, we see improvements that do not manifest themselves in the overall classification performance. Results for these more difficult cases are provided in Figure 4. As when we varied the classifier, we see that, although the margin may cross zero at approximately the same point, the decrease is much more gradual in most cases using the alternative selection methods (direct attacks against CiteSeer being a notable exception). For example, in attacks against small-margin targets in Cora, creating the same reduction in margin as with 10 perturbations under random selection requires 20 perturbations using the alternative methods. We again see that there are larger margins using the baseline method, which makes those targets less vulnerable to attack, even if their margins decrease more quickly as they are perturbed.

One additional consideration is whether either of these methods degrades overall system performance, thus yielding a tradeoff between overall performance and robustness. The experiments we have run so far suggest that there is some variation, as illustrated in Figure 5. While it seems that using high-degree vertices does slightly degrade classification performance, it is less clear that the GreedyCover does; it slightly improves performance for CiteSeer, perhaps because ensuring more unlabeled nodes have labeled neighbors enhances the utility of the structure in the sparser graph.

## 5 RELATED WORK

Adversarial examples in deep neural networks have received considerable attention since they were documented a few years ago [18]. Since that time, numerous attack methods have been proposed, largely focused on the image classification domain (though there has been interest in natural language processing as well, e.g., [11]). In addition to documenting adversarial examples, Szegedy et al.

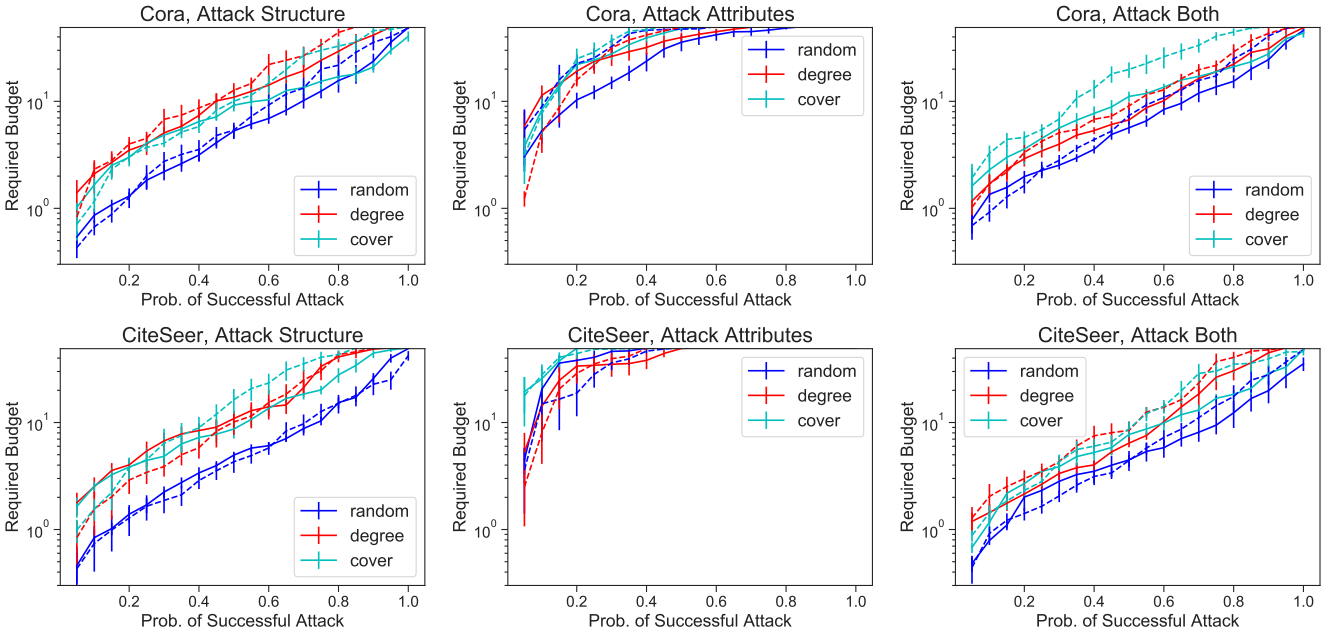


**Figure 3: Budget required to achieve a given probability of attack success, varying the training data selection method. Results are shown for GCNs with one (solid line) or two (dash line) hidden layers. We consider attacks on the structure of the graph (left column), the vertex attributes (center column), and attacks against both simultaneously (right column). All results use attacks against neighbors of a randomly selected target. Using GreedyCover consistently outperforms random selection, often by a factor of 2 and sometimes by a factor of 4. Training with high-degree nodes also typically shows a substantial benefit.**

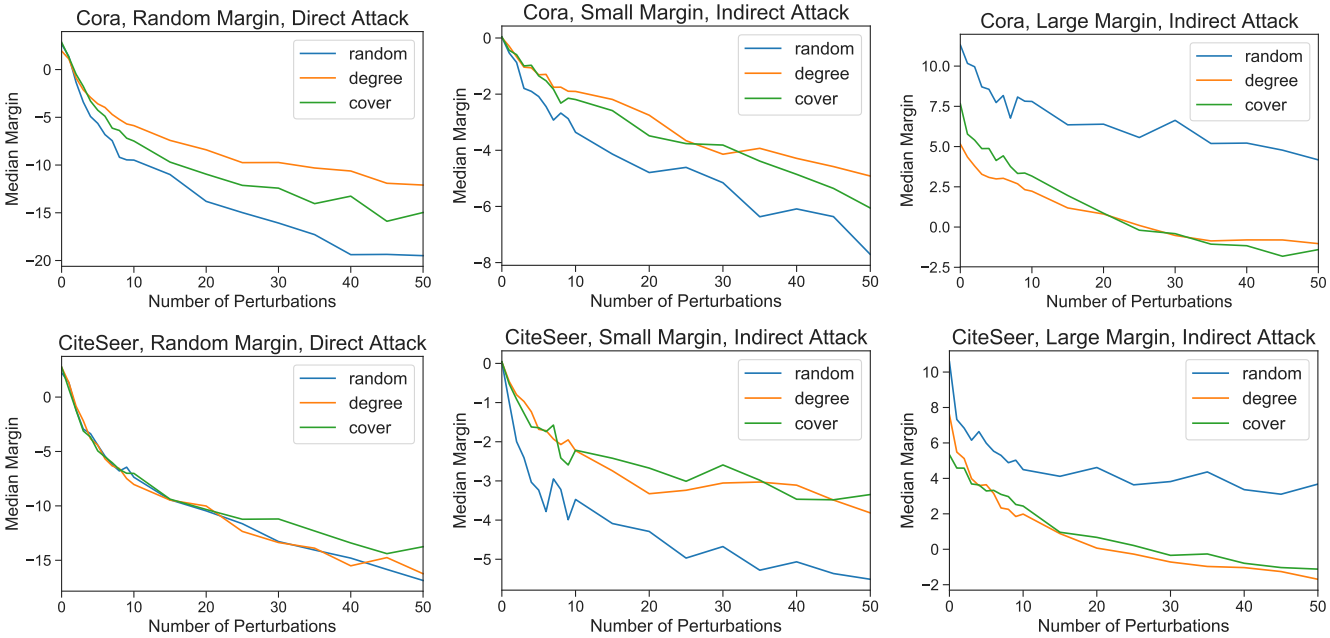


**Figure 4: Median margin after perturbation, varying the training data selection method, in cases that are difficult to improve: direct attacks (left column), small margin cases (center column), and large margin cases (right column). Attacks are against a GCN with one hidden layer and change both structure and attributes. In the direct and small margin attacks, even when misclassification occurs (0 crossing) at the same perturbation level, the margin often decreases faster with random selection.**

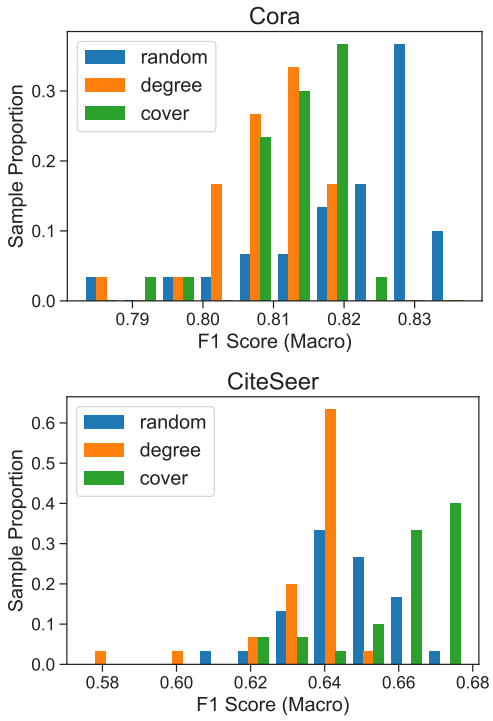


**Figure 5: Distribution of F1 scores (macro aggregated) using all 3 training set selection methods on both datasets before perturbation. Performance is for a GCN with one hidden layer. The alternative methods sometimes reduce performance, suggesting there may be a tradeoff between robustness and classification performance.**

demonstrated that such examples can be generated using the limited-memory BFGS (L-BFGS) algorithm, which identifies an adversarial example in an incorrect class with minimal $L_2$ norm to the true data. Later, Goodfellow et al. proposed the fast gradient sign method (FGSM), where the attacker starts with a clean image and takes small, equal-sized steps in each dimension (i.e., alters each pixel by the same amount) in the direction maximizing the loss [7]. Another proposed attack—the Jacobian-based Saliency Map Attack (JSMA)—iteratively modifies the pixel with the largest impact on the loss [15]. DeepFool, like L-BFGS, minimizes the $L_2$ distance from the true instance while crossing a boundary into an incorrect class, but does so quickly by approximating the classifier as linear, stepping to maximize the loss, then correcting for the true classification surface [13]. Like Nettack, these methods all try to maintain closeness to the original data ($L_2$ norm for L-BFGS and DeepFool, $L_0$ norm for JSMA, and $L_\infty$ norm for FGSM).

The Nettack paper only compares to one of these methods: FGSM. In future work, it would be worthwhile to also compare to JSMA and DeepFool. These both have similarities to Nettack: JSMA has a similar greedy strategy of iteratively choosing a dimension to maximally increase the loss, and DeepFool uses a linear approximation of the classifier. In addition, another attack against vertex classification has been introduced [5]. This method uses reinforcement learning to identify modifications to graph structure for an evasion attack. A comparison between these methods would be valuable.

Defenses to these attacks have been proposed, although several prove to be insufficient against stronger attacks. Defensive distillation is one such defense, in which a classifier is trained with high "temperature" in the softmax, which is reduced for classification [16]. While this was effective against the methods from [7, 13, 15, 18], it was shown in [3] that modifying the attack by changing the constraint function (which ensures the adversarial example is in a given class) renders this defense ineffective. As more defenses have been proposed, such as pixel deflection [17] and randomization techniques [20], but many such methods are still found to be vulnerable to attacks [1, 2]. Future work will also consider defenses that appear promising (e.g., [4, 19]) if they can be applied in the graph domain.

## 6 CONCLUSIONS

This paper describes work in progress aiming to make vertex classification robust to the adversarial poisoning technique Nettack. We consider two possible approaches to enhance robustness to Nettack: varying the classifier used and varying the training set selection method. When transferring Nettack to classifiers other than a GCN—specifically classifiers that decouple structure from attributes—we note that there is a much more gradual decline in the classification margin than when using the GCN. In particular, using ReFeX to embed the structure of the graph leads to much greater robustness among vertices that are correctly classified. There is, however, a reduction in overall performance, and achieving a balance between performance and robustness will be essential future work. When considering alternative ways to select training data, we also see an improvement in robustness, both in terms of increasing the budget required for a successful attack (often by a factor of 2 or more) and, in cases where this is not possible, making the classifier much less confident in its incorrect predictions. Both of these developments are highly encouraging as we explore possibilities to enhance vertex classification robustness.

The work we have documented here points to several avenues of investigation we intend to pursue. First and foremost, developing methods that achieve the performance of the GCN on the overall dataset with the robustness of ReFeX on the targets is the ultimate goal, and investigating a classification ensemble that incorporates all of these classifiers may be a way to achieve this. Similarly, developing a hybrid method of selecting the training data that uses a combination of randomly selected data and nodes that cover the test set could prove useful if it enables both large classification margins and robustness to perturbations. Of course, we must also consider how an attacker may take these alternative contexts into account, and update Nettack accordingly. Finally, changing the way in which performance is evaluated may be interesting: splitting nodes based on degree rather than margin and considering the margin change for all nodes instead of those correctly classified. In addition, expanding the notion of an "unnoticeable" or subtle attack to include preserving triangle count would be helpful, as the attacks appear to shift the triangle distribution (see Figure 6). Robustness to adversarial activity has driven fascinating research in the image

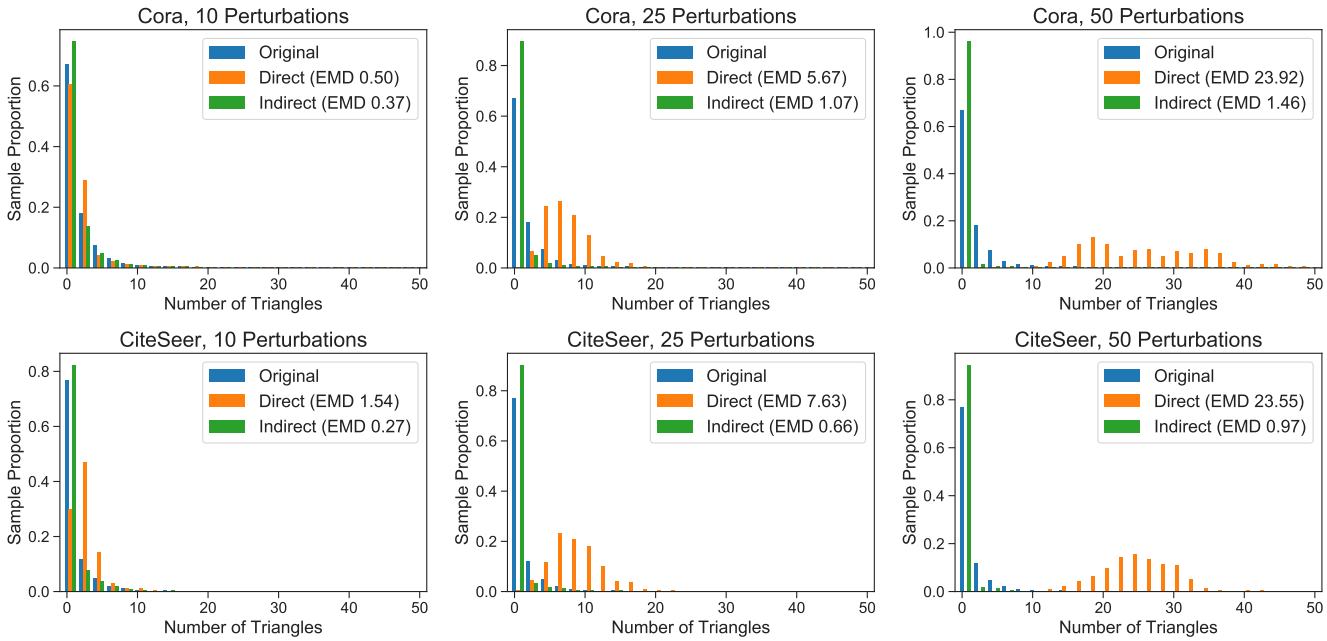


**Figure 6: Distribution of triangle counts. In both Cora (top) and CiteSeer (bottom), the number of triangles goes up for direct attacks and down for indirect attacks. Thus, the subtlety of Nettack's perturbations is questionable. The legend is annotated with the earth mover's distance (EMD) of the perturbed triangle distribution from the original triangle distribution.**

classification domain. We look forward to new discoveries as the same is done for vertex classification.

## 7 ACKNOWLEDGEMENTS

The authors would like to thank Qi (Rose) Yu and Rui Wang for helpful feedback on the training modification portion of this work.

This research was sponsored in part by the Combat Capabilities Development Command Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF-13-2-0045 (ARL Cyber Security CRA) and by the United States Air Force under Air Force Contract No. FA8702-15-D-0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Combat Capabilities Development Command Army Research Laboratory, the United States Air Force, or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes not withstanding any copyright notation here on.

## REFERENCES

[1] Anish Athalye and Nicholas Carlini. 2018. On the Robustness of the CVPR 2018 White-Box Adversarial Example Defenses. *CoRR* abs/1804.03286 (2018).
[2] Anish Athalye, Nicholas Carlini, and David Wagner. 2018. Obfuscated Gradients Give a False Sense of Security: Circumventing Defenses to Adversarial Examples. In *ICML*. 274–283.
[3] N. Carlini and D. Wagner. 2017. Towards Evaluating the Robustness of Neural Networks. In *IEEE Symp. on Security and Privacy (SP)*. 39–57.
[4] Francesco Croce, Maksym Andriushchenko, and Matthias Hein. 2019. Provable Robustness of ReLU networks via Maximization of Linear Regions. In *AISTATS*. 2057–2066.
[5] Hanjun Dai, Hui Li, Tian Tian, Xin Huang, Lin Wang, Jun Zhu, and Le Song. 2018. Adversarial Attack on Graph Structured Data. In *ICML*. 1115–1124.
[6] Nikhil Desai. 2014. Beyond Community Detection - RolX. https://github.com/Lab41/Circulo/blob/master/circulo/algorithms/rolx.py. (2014).
[7] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and Harnessing Adversarial Examples. In *ICLR*. http://arxiv.org/abs/1412.6572
[8] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable Feature Learning for Networks. In *KDD*. 855–864.
[9] William L. Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *NeurIPS (formerly known as NIPS)*. 1025–1035.
[10] Keith Henderson, Brian Gallagher, Lei Li, Leman Akoglu, Tina Eliassi-Rad, Hanghang Tong, and Christos Faloutsos. 2011. It's Who You Know: Graph Mining Using Recursive Structural Features. In *KDD*. 663–671.
[11] Robin Jia and Percy Liang. 2017. Adversarial Examples for Evaluating Reading Comprehension Systems. In *EMNLP*. 2021–2031.
[12] John Moore and Jennifer Neville. 2017. Deep collective inference. In *AAAI*. 2364–2372.
[13] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. 2016. DeepFool: A Simple and Accurate Method to Fool Deep Neural Networks. In *CVPR*. 2574–2582.
[14] J. Neville, B. Gallagher, and T. Eliassi-Rad. 2009. Evaluating Statistical Tests for Within-Network Classifiers of Relational Data. In *IEEE ICDM*. 397–406.
[15] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami. 2016. The Limitations of Deep Learning in Adversarial Settings. In *IEEE European Symp. on Security and Privacy (EuroSP)*. 372–387.
[16] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami. 2016. Distillation as a Defense to Adversarial Perturbations Against Deep Neural Networks. In *IEEE Symp. on Security and Privacy (SP)*. 582–597.
[17] Aaditya Prakash, Nick Moran, Solomon Garber, Antonella DiLillo, and James Storer. 2018. Deflecting Adversarial Attacks With Pixel Deflection. In *CVPR*. 8571–8580.
[18] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2014. Intriguing properties of neural networks. In *ICLR*. http://arxiv.org/abs/1312.6199
[19] Eric Wong and Zico Kolter. 2018. Provable Defenses against Adversarial Examples via the Convex Outer Adversarial Polytope. In *ICML*. 5286–5295.
[20] Cihang Xie, Jianyu Wang, Zhishuai Zhang, Zhou Ren, and Alan Yuille. 2018. Mitigating Adversarial Effects Through Randomization. In *ICLR*. https://openreview.net/forum?id=Sk9yuql0Z
[21] Daniel Zügner, Amir Akbarnejad, and Stephan Günnemann. 2018. Adversarial Attacks on Neural Networks for Graph Data. In *KDD*. 2847–2856.