Adversarial Attacks on Neural Networks for Graph Data

Daniel Zügner   Amir Akbarnejad   Stephan Günnemann
Technical University of Munich, Germany
{zuegnerd,amir.akbarnejad,guennemann}@in.tum.de

ABSTRACT

Deep learning models for graphs have achieved strong performance for the task of node classification. Despite their proliferation, currently there is no study of their robustness to adversarial attacks. Yet, in domains where they are likely to be used, e.g. the web, adversaries are common. Can deep learning models for graphs be easily fooled? In this work, we introduce the first study of adversarial attacks on attributed graphs, specifically focusing on models exploiting ideas of graph convolutions. In addition to attacks at test time, we tackle the more challenging class of poisoning/causative attacks, which focus on the training phase of a machine learning model. We generate adversarial perturbations targeting the *node's features* and the *graph structure*, thus, taking the dependencies between instances in account. Moreover, we ensure that the perturbations remain *unnoticeable* by preserving important data characteristics. To cope with the underlying discrete domain we propose an efficient algorithm Nettack exploiting incremental computations. Our experimental study shows that accuracy of node classification significantly drops even when performing only few perturbations. Even more, our attacks are transferable: the learned attacks generalize to other state-of-the-art node classification models and unsupervised approaches, and likewise are successful even when only limited knowledge about the graph is given.

KEYWORDS

Adversarial machine learning, graph mining, network mining, graph convolutional networks, semi-supervised learning

1 INTRODUCTION

Graph data is the core for many high impact applications ranging from the analysis of social and rating networks (Facebook, Amazon), over gene interaction networks (BioGRID), to interlinked document collections (PubMed, Arxiv). One of the most frequently applied
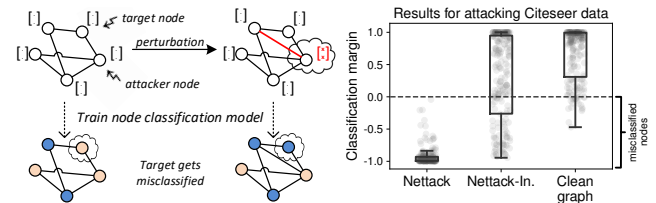
Figure 1: Small perturbations of the graph structure and node features lead to misclassification of the target.

tasks on graph data is *node classification*: given a single large (attributed) graph and the class labels of a few nodes, the goal is to predict the labels of the remaining nodes. For example, one might wish to classify the role of a protein in a biological interaction graph [17], predict the customer type of users in e-commerce networks [12], or assign scientific papers from a citation network into topics [19].

While many classical approaches have been introduced in the past to tackle the node classification problem [8, 21], the last years have seen a tremendous interest in methods for *deep learning on graphs* [5, 7, 25]. Specifically, approaches from the class of graph convolutional networks [19, 28] have achieved strong performance in many graph-learning tasks including node classification.

The strength of these methods — beyond their non-linear, hierarchical nature – relies on their use of the graphs' relational information to perform classification: instead of only considering the instances individually (nodes and their features), the relationships between them are exploited as well (the edges). Put differently: the instances are not treated independently; we deal with a certain form of non-i.i.d. data where so-called network effects such as homophily [21] support the classification.

However, there is one big catch: Many researchers have noticed that deep learning architectures for classical learning tasks can easily be fooled/attacked [14, 30] . Even only slight, deliberate perturbations of an instance – also known as *adversarial perturbations/examples* – can lead to wrong predictions. Such negative results significantly hinder the applicability of these models, leading to unintuitive and unreliable results, and they additionally open the door for attackers that can exploit these vulnerabilities. So far, however, the question of adversarial perturbations for deep learning methods on graphs has not been addressed. This is highly critical, since especially in domains where graph-based learning is used (e.g. the web) adversaries are common and false data is *easy to inject*: spammers add wrong information to social networks; fraudsters frequently manipulate online reviews and product websites [18].

In this work, we close this gap and we investigate whether such manipulations are possible. *Can deep learning models for attributed graphs be easily fooled? How reliable are their results?*

The answer to this question is indeed not foreseeable: On one hand the relational effects might improve robustness since predictions are not based on individual instances only but based on

various instances jointly. On the other hand, the propagation of information might also lead to cascading effects, where manipulating a single instance affects many others. Indeed, compared to the existing works on adversarial attacks, our work significantly differs in various aspects.

Opportunities: (1) Since we are operating on an attributed graph, adversarial perturbations can manifest in two different ways: by changing the nodes' features or the graph structure. Manipulating the graph, i.e. the dependency structure between instances, has not been studied so far, but is a highly likely scenario in real-life. For example, one might add or remove (fake) friendship relations to a social network. (2) While existing works were limited to manipulating an instance itself to enforce its wrong prediction[1], the relational effects give us more power: by manipulating one instance, we might specifically misguide the prediction for another instance. Again, this scenario is highly realistic. Think about a fraudster who hijacks some accounts, which he then manipulates to enforce a wrong prediction for *another* account he has *not* under control. Thus, in graph-based learning scenarios we can distinguish between (i) nodes which we aim to misclassify, called *targets*, and (ii) nodes which we can directly manipulate, called *attackers*. Figure 1 illustrates the goal of our work and shows the result of our method on the Citeseer network. Clearly, compared to classical attacks to learning models, graphs enable much richer potential for perturbations. But likewise, constructing them is far more challenging.

Challenges: (1) Unlike, e.g., images consisting of continuous features, the graph structure – and often also the nodes' features – is discrete. Therefore, gradient based approaches [14, 23] for finding perturbations are not suited. How to design efficient algorithms that are able to find adversarial examples in a discrete domain? (2) Adversarial perturbations are aimed to be unnoticeable (by humans). For images, one often enforces, e.g., a maximum deviation per pixel value. How can we capture the notion of 'unnoticeable changes' in a (binary, attributed) graph? (3) Last, node classification is usually performed in a *transductive* learning setting. Here, the train and test data are used jointly to learn a new classification model before the predictions are performed on the specific test data. This means, that the predominantly performed evasion attacks – where the parameters of the classification model are assumed to be static – are not realistic. The model has to be (re)trained on the manipulated data. Thus, graph-based learning in a transductive setting is inherently related to the challenging poisoning/causative attacks [3].

Given these challenges, we propose a principle for adversarial perturbations of attributed graphs that aim to fool state-of-the art deep learning models for graphs. In particular, we focus on semi-supervised classification models based on graph convolutions such as GCN [19] and Column Network (CLN) [28] – but we will also showcase our methods' potential on the unsupervised model DeepWalk [27]. By default, we assume an attacker with knowledge about the full data, which can, however, only manipulate parts of it. This assumption ensures reliable vulnerability analysis in the worst case. But even when only parts of the data are known, our attacks are still successful as shown by our experiments. Overall, our contributions are:

- *Model:* We propose a model for adversarial attacks on attributed graphs considering node classification. We introduce new types of attacks where we explicitly distinguish between the attacker and the target nodes. Our attacks can manipulate the graph structure and node features while ensuring unnoticeable changes by preserving important data characteristics (e.g. degree distribution, co-occurence of features).

- *Algorithm:* We develop an efficient algorithm NETTACK for computing these attacks based on linearization ideas. Our methods enables incremental computations and exploits the graph's sparsity for fast execution.

- *Experiments:* We show that our model can dramatically worsen classification results for the target nodes by only requiring few changes to the graph. We furthermore show that these results transfer to other established models, hold for various datasets, and even work when only parts of the data are observed. Overall, this highlights the need to handle attacks to graph data.

2 PRELIMINARIES

We consider the task of (semi-supervised) node classification in a single large graph having binary node features. Formally, let $G = (A, X)$ be an attributed graph, where $A \in \{0, 1\}^{N \times N}$ is the adjacency matrix representing the connections and $X \in \{0, 1\}^{N \times D}$ represents the nodes' features. We denote with $x_v \in \{0, 1\}^D$ the D-dim. feature vector of node v. W.l.o.g. we assume the node-ids to be $\mathcal{V} = \{1, \ldots, N\}$ and the feature-ids to be $\mathcal{F} = \{1, ..., D\}$.

Given a subset $\mathcal{V}_L \subseteq \mathcal{V}$ of labeled nodes, with class labels from $C = \{1, 2, \ldots, c_K\}$, the goal of node classification is to learn a function $g : \mathcal{V} \to C$ which maps each node $v \in \mathcal{V}$ to one class in C.[2] Since the predictions are done for the *given* test instances, which are already known before (and also used during) training, this corresponds to a typical *transductive* learning scenario [8].

In this work, we focus on node classification employing graph convolution layers. In particular, we will consider the well established work [19]. Here, the hidden layer $l + 1$ is defined as

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

where $\tilde{A} = A + I_N$ is the adjacency matrix of the (undirected) input 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 $\sigma(\cdot)$ is an activation function (usually ReLU). In the first layer we have $H^{(0)} = X$, i.e. using the nodes' features as input. Since the latent representations H are (recursively) relying on the neighboring ones (multiplication with $\tilde{A}$), all instances are coupled together. Following the authors of [19], we consider GCNs with a single hidden layer:

$$Z = f_\theta(A, X) = \text{softmax} \left(\hat{A} \, \sigma \left(\hat{A} X W^{(1)} \right) W^{(2)} \right), \tag{2}$$

where $\hat{A} = \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}}$. The output Z_{vc} denotes the probability of assigning node v to class c. Here, we used θ do denote the set of all parameters, i.e. $\theta = \{W^{(1)}, W^{(2)}\}$. The optimal parameters θ are

---

[1] Due to the independence assumption, a misclassification for instance i can only be achieved by manipulating instance i itself for the commonly studied evasion (test-time) attacks. For the less studied poisioning attacks we might have indirect influence.

[2] Please note the difference to (structured) learning settings where we have *multiple but independent* graphs as training input with the goal to perform a prediction for each *graph*. In this work, the prediction is done per *node* (e.g. a person in a social network) – and especially we have *dependencies* between the nodes/data instances via the edges.

then learned in a semi-supervised fashion by minimizing cross-entropy on the output of the labeled samples $\mathcal{V}_L$, i.e. minimizing

$$L(\theta; A, X) = - \sum_{v \in \mathcal{V}_L} \ln Z_{v, c_v} \quad , \quad Z = f_\theta(A, X) \qquad (3)$$

where c_v is the given label of v from the training set. After training, Z denotes the class probabilities for every instance in the graph.

3 RELATED WORK

In line with the focus of this work, we briefly describe deep learning methods for graphs aiming to solve the node classification task.

Deep Learning for Graphs. Mainly two streams of research can be distinguished: (i) *node embeddings* [5, 7, 16, 27] – that often operate in an unsupervised setting – and (ii) *architectures employing layers specifically designed for graphs* [19, 25, 28]. In this work, we focus on the second type of principles and additionally show that our adversarial attack transfers to node embeddings as well. Regarding the developed layers, most works seek to adapt conventional CNNs to the graph domain: called graph convolutional layers or neural message passing [11, 13, 19, 25, 28]. Simply speaking, they reduce to some form of aggregation over neighbors as seen in Eq. (2). A more general setting is described in [13] and an overview of methods given in [7, 25].

Adversarial Attacks. Attacking machine learning models has a long history, with seminal works on, e.g., SVMs or logistic regression [23]. In contrast to outliers, e.g. present in attributed graphs [4], adversarial examples are created deliberately to mislead machine learning models and often are designed to be unnoticeable. Recently, deep neural networks have shown to be highly sensitive to these small adversarial perturbations to the data [14, 30]. Even more, the adversarial examples generated for one model are often also harmful when using another model: known as transferability [32]. Many tasks and models have been shown to be sensitive to adversarial attacks; however, all assume the data instances to be independent. Even [33], which considers relations between different tasks for multi-task relationship learning, still deals with the classical scenario of i.i.d. instances within each task. For interrelated data such as graphs, where the data instances (i.e. nodes) are not treated independently, no such analysis has been performed yet.

Taxonomies characterizing the attack have been introduced in [3, 26]. The two dominant types of attacks are poisoning/causative attacks which target the training data (specifically, the model's training phase is performed *after* the attack) and evasion/exploratory attacks which target the test data/application phase (here, the learned model is assumed fixed). Deriving effective poisoning attacks is usually computationally harder since also the subsequent learning of the model has to be considered. *This categorization is not optimally suited for our setting*. In particular, attacks on the test data are causative as well since the test data is used while training the model (transductive, semi-supervised learning). Further, even when the model is fixed (evasion attack), manipulating one instance might affect all others due to the relational effects imposed by the graph structure. Our attacks are powerful even in the more challenging scenario where the model is retrained.

Generating Adversarial Perturbations. While most works have focused on generating adversarial perturbations for evasion attacks, poisoning attacks are far less studied [20, 23, 33] since

they require to solve a challenging bi-level optimization problem that considers learning the model. In general, since finding adversarial perturbations often reduces to some non-convex (bi-level) optimization problem, different approximate principles have been introduced. Indeed, almost all works exploit the gradient or other moments of a given differentiable (surrogate) loss function to guide the search in the neighborhood of legitimate perturbations [14, 15, 20, 23, 26]. For discrete data, where gradients are undefined, such an approach is suboptimal.

Hand in hand with the attacks, the robustification of machine learning models has been studied – known as adversarial machine learning or robust machine learning. Since this is out of the scope of the current paper, we do not discuss these approaches here.

Adversarial Attacks when Learning with Graphs. Works on adversarial attacks for graph learning tasks are almost non-existing. For graph clustering, the work [9] has measured the changes in the result when injecting noise to a bi-partite graph that represent DNS queries. Though, they do not focus on generating attacks in a principled way. Our work [6] considered noise in the graph structure to improve the robustness when performing spectral clustering. Similarly, to improve robustness of collective classification via associative Markov networks, the work [31] considers adversarial noise in the features. They only use label smoothness and assume that the attacker can manipulate the features of every instance. Overall, no work so far has considered poisoning/training-time attacks on neural networks for attributed graphs.

4 ATTACK MODEL

Given the node classification setting as described in Sec. 2, our goal is to perform small perturbations on the graph $G^{(0)} = (A^{(0)}, X^{(0)})$, leading to the graph $G' = (A', X')$, such that the classification performance drops. Changes to $A^{(0)}$, are called **structure attacks**, while changes to $X^{(0)}$ are called **feature attacks**.

Target vs. Attackers. Specifically, our goal is to attack a specific target node $v_0 \in \mathcal{V}$, i.e. we aim to change v_0's prediction. Due to the non-i.i.d. nature of the data, v_0's outcome not only depends on the node itself, but also on the other nodes in the graph. Thus, we are not limited to perturbing v_0 but we can achieve our aim by changing other nodes as well. Indeed, this reflects real world scenarios much better since it is likely that an attacker has access to a few nodes only, and not to the entire data or v_0 itself. Therefore, besides the target node, we introduce the attacker nodes $\mathcal{A} \subseteq \mathcal{V}$. The perturbations on $G^{(0)}$ are constrained to these nodes, i.e. it must hold

$$X'_{ui} \neq X_{ui}^{(0)} \Rightarrow u \in \mathcal{A} \quad , \quad A'_{uv} \neq A_{uv}^{(0)} \Rightarrow u \in \mathcal{A} \vee v \in \mathcal{A} \quad (4)$$

If the target $v_0 \notin \mathcal{A}$, we call the attack an **influencer attack**, since v_0 gets not manipulated directly, but only indirectly via some influencers. If $\{v_0\} = \mathcal{A}$, we call it a **direct attack**.

To ensure that the attacker can not modify the graph completely, we further limit the number of allowed changes by a budget Δ:

$$\sum_u \sum_i |X_{ui}^{(0)} - X'_{ui}| + \sum_{u < v} |A_{uv}^{(0)} - A'_{uv}| \leq \Delta \qquad (5)$$

More advanced ideas will be discussed in Sec. 4.1. For now, we denote with $\mathcal{P}_{\Delta, \mathcal{A}}^{G0}$ the set of all graphs G' that fulfill Eq. (4) and (5). Given this basic set-up, our problem is defined as:

PROBLEM 1. *Given a graph $G^{(0)} = (A^{(0)}, X^{(0)})$, a target node v_0, and attacker nodes $\mathcal{A}$. Let c_{old} denote the class for v_0 based on the graph $G^{(0)}$ (predicted or using some ground truth). Determine*

$$\underset{(A',X') \in \mathcal{P}_{\Delta,\mathcal{A}}^{G0}}{\arg \max} \; \underset{c \neq c_{old}}{\max} \; \ln Z_{v_0,c}^* - \ln Z_{v_0,c_{old}}^*$$

subject to $Z^* = f_{\theta^*}(A', X')$ with $\theta^* = \arg \min_{\theta} L(\theta; A', X')$

That is, we aim to find a perturbed graph G' that classifies v_0 as c_{new} and has maximal 'distance' (in terms of log-probabilities/logits) to c_{old}. Note that for the perturbed graph G', the optimal parameters θ^* are used, matching the transductive learning setting where the model is learned on the given data. Therefore, we have a bi-level optimization problem. As a simpler variant, one can also consider an evasion attack assuming the parameters are static and learned based on the old graph, $\theta^* = \arg \min_{\theta} L(\theta; A^{(0)}, X^{(0)})$.

4.1 Unnoticeable Perturbations

Typically, in an adversarial attack scenario, the attackers try to modify the input data such that the changes are *unnoticable*. Unlike to image data, where this can easily be verified visually and by using simple constraints, in the graph setting this is much harder mainly for two reasons: (i) the graph structure is discrete preventing to use infinitesimal small changes, and (ii) sufficiently large graphs are not suitable for visual inspection.

How can we ensure unnoticeable perturbations in our setting? In particular, we argue that only considering the budget Δ might not be enough. Especially if a large Δ is required due to complicated data, we still want realistically looking perturbed graphs G'. Therefore, our core idea is to allow only those perturbations that preserve specific inherent properties of the input graph.

Graph structure preserving perturbations. Undoubtedly, the most prominent characteristic of the graph structure is its degree distribution, which often resembles a power-law like shape in real networks. If two networks show very different degree distributions, it is easy to tell them apart. Therefore, we aim to only generate perturbations which follow similar power-law behavior as the input.

For this purpose we refer to a statistical two-sample test for power-law distributions [2]. That is, we estimate whether the two degree distributions of $G^{(0)}$ and G' stem from the same distribution or from individual ones, using a likelihood ratio test.

More precisely, the procedure is as follows: We first estimate the scaling parameter α of the power-law distribution $p(x) \propto x^{-\alpha}$ referring to the degree distribution of $G^{(0)}$ (equivalently for G'). While there is no exact and closed-form solution to estimate α in the case of discrete data, [10] derived an approximate expression, which for our purpose of a graph G translates to

$$\alpha_G \approx 1 + |\mathcal{D}_G| \cdot \left[\sum_{d_i \in \mathcal{D}_G} \log \frac{d_i}{d_{\min} - \frac{1}{2}} \right]^{-1} \quad (6)$$

where d_{min} denotes the minimum degree a node needs to have to be considered in the power-law test and $\mathcal{D}_G = \{d_v^G \mid v \in \mathcal{V}, d_v^G \geq d_{min}\}$ is the multiset containing the list of node degrees, where d_v^G is the degree of node v in G. Using this, we get estimates for the values $\alpha_{G^{(0)}}$ and $\alpha_{G'}$. Similarly, we can estimate α_{comb} using the combined samples $\mathcal{D}_{comb} = \mathcal{D}_{G^{(0)}} \cup \mathcal{D}_{G'}$.

Given the scaling parameter α_x, the log-likelihood for the samples $\mathcal{D}_x$ can easily be evaluated as

$$l(\mathcal{D}_x) = |\mathcal{D}_x| \cdot \log \alpha_x + |\mathcal{D}_x| \cdot \alpha_x \cdot \log d_{\min} + (\alpha_x + 1) \sum_{d_i \in \mathcal{D}_x} \log d_i \quad (7)$$

Using these log-likelihood scores, we set up the significance test, estimating whether the two samples $\mathcal{D}_{G^{(0)}}$ and $\mathcal{D}_{G'}$ come from the same power law distribution (null hypotheses H_0) as opposed to separate ones (H_1). That is, we formulate two competing hypotheses

$$l(H_0) = l(\mathcal{D}_{comb}) \quad \text{and} \quad l(H_1) = l(\mathcal{D}_{G^{(0)}}) + l(\mathcal{D}_{G'}) \quad (8)$$

Following the likelihood ratio test, the final test statistic is

$$\Lambda(G^{(0)}, G') = -2 \cdot l(H_0) + 2 \cdot l(H_1). \quad (9)$$

which for large sample sizes follows a χ^2 distribution with one degree of freedom [2].

A typical p-value for rejecting the null hypothesis H_0 (i.e. concluding that both samples come from different distributions) is 0.05, i.e., statistically, in one out of twenty cases we reject the null hypothesis although it holds (*type I error*). In our adversarial attack scenario, however, we argue that a human trying to find out whether the data has been manipulated would be far more conservative and ask the other way: Given that the data was manipulated, what is the probability of the test falsely not rejecting the null hypothesis (*type II error*).

While we cannot compute the type II error in our case easily, type I and II error probabilities have an inverse relation in general. Thus, by selecting a very conservative p-value corresponding to a high type I error, we can reduce the probability of a type II error. We therefore set the critical p-value to 0.95, i.e. if we were to sample two degree sequences from the same power law distribution, we were to reject the null hypothesis in 95% of the times and could then investigate whether the data has been compromised based on this initial suspicion. On the other hand, if our modified graph's degree sequence passes this very conservative test, we conclude that the changes to the degree distribution are *unnoticeable*.

Using the above p-value in the χ^2 distribution, we only accept perturbations $G' = (A', X')$ where the degree distribution fulfills

$$\Lambda(G^{(0)}, G') < \tau \approx 0.004 \quad (10)$$

Feature statistics preserving perturbations. While the above principle could be applied to the nodes' features as well (e.g. preserving the distribution of feature occurrences), we argue that such a procedure is too limited. In particular, such a test would not well reflect the correlation/co-occurence of different features: If two features have never occured together in $G^{(0)}$, but they do once in G', the distribution of feature occurences would still be very similar. Such a change, however, is easily noticable. Think, e.g., about two words which have never been used together but are suddenly used in G'. Thus, we refer to a test based on feature co-occurrence.

Since designing a statistical test based on the co-occurences requires to model the joint distribution over features – intractable for correlated multivariate binary data [24] – we refer to a deterministic test. In this regard, setting features to 0 is uncritical since it does not introduce new co-occurences. The question is: Which features of a node u can be set to 1 to be regarded unnoticeable?

To answer this question, we consider a probabilistic random walker on the co-occurence graph $C = (\mathcal{F}, E)$ of features from $G^{(0)}$, i.e. $\mathcal{F}$ is the set of features and $E \subseteq \mathcal{F} \times \mathcal{F}$ denotes which features have occurred together so far. We argue that adding a feature i is unnoticeable if the probability of reaching it by a random walker starting at the features originally present for node u and performing one step is significantly large. Formally, let $S_u = \{j \mid X_{uj} \neq 0\}$ be the set of all features originally present for node u. We consider addition of feature $i \notin S_u$ to node u as unnoticeable if

$$p(i \mid S_u) = \frac{1}{|S_u|} \sum_{j \in S_u} 1/d_j \cdot E_{ij} > \sigma. \qquad (11)$$

where d_j denotes the degree in the co-occurrence graph C. That is, given that the probabilistic walker has started at any feature $j \in S_u$, after performing one step it would reach the feature i at least with probability σ. In our experiments we simply picked σ to be half of the maximal achievable probably, i.e. $\sigma = 0.5 \cdot \frac{1}{|S_u|} \sum_{j \in S_u} 1/d_j$.

The above principle has two desirable effects: First, features i which have co-occurred with many of u's features (i.e. in other nodes) have a high probability; they are less noticeable when being added. Second, features i that only co-occur with features $j \in S_u$ that are not specific to the node u (e.g. features j which co-occur with almost every other feature; stopwords) have low probability; adding i would be noticeable. Thus, we obtain the desired result.

Using the above test, we only accept perturbations $G' = (A', X')$ where the feature values fulfill

$$\forall u \in \mathcal{V} : \forall i \in \mathcal{F} : X'_{ui} = 1 \Rightarrow i \in S_u \vee p(i|S_u) > \sigma \qquad (12)$$

In summary, to ensure unnoticeable perturbations, we update our problem definition to:

PROBLEM 2. *Same as Problem 1 but replacing $\mathcal{P}^{G0}_{\Delta,\mathcal{A}}$ with the more restricted set $\hat{\mathcal{P}}^{G0}_{\Delta,\mathcal{A}}$ of graphs that additionally preserve the degree distribution (Eq. 10) and feature co-occurence (Eq. 12).*

5 GENERATING ADVERSARIAL GRAPHS

Solving Problem 1/2 is highly challenging. While (continuous) bi-level problems for attacks have been addressed in the past by gradient computation based on first-order KKT conditions [20, 23], such a solution is not possible in our case due to the data's discreteness and the large number of parameters θ. Therefore, we propose a sequential approach, where we first attack a *surrogate model*, thus, leading to an attacked graph. This graph is subsequently used to train the final model. Indeed, this approach can directly be considered as a check for transferability since we do not specifically focus on the used model but only on a surrogate one.

Surrogate model. To obtain a tractable surrogate model that still captures the idea of graph convolutions, we perform a linearizion of the model from Eq. 2. That is, we replace the nonlinearity $\sigma(.)$ with a simple linear activation function, leading to:

$$Z' = \text{softmax}\left(\hat{A}\,\hat{A} X W^{(1)} W^{(2)}\right) = \text{softmax}\left(\hat{A}^2 X W\right) \qquad (13)$$

Since $W^{(1)}$ and $W^{(2)}$ are (free) parameters to be learned, they can be absorbed into a single matrix $W \in \mathbb{R}^{D \times K}$.

Since our goal is to maximize the difference in the log-probabilities of the target v_0 (given a certain budget Δ), the instance-dependent normalization induced by the softmax can be ignored. Thus, the

log-probabilities can simply be reduced to $\hat{A}^2 X W$. Accordingly, given the trained surrogate model on the (uncorrupted) input data with learned parameters W, we define the surrogate loss

$$\mathcal{L}_s (A, X; W, v_0) = \max_{c \neq c_{old}} [\hat{A}^2 X W]_{v_0 c} - [\hat{A}^2 X W]_{v_0 c_{old}} \qquad (14)$$

and aim to solve $\arg\max_{(A', X') \in \hat{\mathcal{P}}^{G0}_{\Delta,\mathcal{A}}} \mathcal{L}_s (A', X'; W, v_0)$.

While being much simpler, this problem is still intractable to solve exactly due to the discrete domain and the constraints. Thus, in the following we introduce a scalable greedy approximation scheme. For this, we define scoring functions that evaluate the surrogate loss from Eq. (14) obtained *after* adding/removing a feature $f = (u, i)$ or edge $e = (u, v)$ to an arbitrary graph $G = (A, X)$:

$$s_{struct}(e; G, v_0) := \mathcal{L}_s(A', X; W, v_0) \qquad (15)$$

$$s_{feat}(f; G, v_0) := \mathcal{L}_s(A, X'; W, v_0) \qquad (16)$$

where $A' := A \pm e$ (i.e. $a'_{uv} = a'_{vu} = 1 - a_{uv}$)[3] and $X' := X \pm f$ (i.e. $x'_{ui} = 1 - x_{ui}$).

Approximate Solution. Algorithm 1 shows the pseudo-code. In detail, following a locally optimal strategy, we sequentially 'manipulate' the most promising element: either an entry from the adjacency matrix or a feature entry (taking the constraints into account). That is, given the current state of the graph $G^{(t)}$, we compute a candidate set C_{struct} of allowable elements (u, v) whose change from 0 to 1 (or vice versa; hence the $\pm$ sign in the pseudocode) does not violate the constraints imposed by $\hat{\mathcal{P}}^{G0}_{\Delta,\mathcal{A}}$. Among these elements we pick the one which obtains the highest difference in the log-probabilites, indicated by the score function $s_{struct}(e; G^{(t)}, v_0)$. Similar, we compute the candidate set C_{feat} and the score function $s_{feat}(f; G^{(t)}, v_0)$ for every allowable feature manipulation of feature i and node u. Whichever change obtains the higher score is picked and the graph accordingly updated to $G^{(t+1)}$. This process is repeated until the budget Δ has been exceeded.

To make Algorithm 1 tractable, two core aspects have to hold: (i) an efficient computation of the score functions s_{struct} and s_{feat}, and (ii) an efficient check which edges and features are compliant with our constraints $\hat{\mathcal{P}}^{G0}_{\Delta,\mathcal{A}}$, thus, forming the sets C_{struct} and C_{feat}. In the following, we describe these two parts in detail.

5.1 Fast computation of score functions

Structural attacks. We start by describing how to compute s_{struct}. For this, we have to compute the class prediction (in the surrogate model) of node v_0 *after* adding/removing an edge (m, n). Since we are now optimizing w.r.t. A, the term XW in Eq. (14) is a constant – we substitute it with $C := XW \in \mathbb{R}^{N \times K}$. The log-probabilities of node v_0 are then given by $g_{v_0} = [\hat{A}^2]_{v_0} \cdot C \in \mathbb{R}^{1 \times K}$ where $[\hat{A}^2]_{v_0}$ denotes a row vector. Thus, we only have to inspect how this row vector changes to determine the optimal edge manipulation.

Naively recomputing $[\hat{A}^2]_{v_0}$ for every element from the candidate set, though, is not practicable. An important observation to alleviate this problem is that in the used two-layer GCN the prediction for each node is influenced by its two-hop neighborhood only. That is, the above row vector is zero for most of the elements. And

---
[3]Please note that by modifying a single element $e = (u, v)$ we always change two entries, a_{uv} and a_{vu}, of A since we are operating on an undirected graph.

Algorithm 1: NETTACK: Adversarial attacks on graphs

Input: Graph $G^{(0)} \leftarrow (A^{(0)}, X^{(0)})$, target node v_0,
    attacker nodes $\mathcal{A}$, modification budget Δ

Output: Modified Graph $G' = (A', X')$

Train surrogate model on $G^{(0)}$ to obtain W // Eq. (13);

$t \leftarrow 0$;

while $|A^{(t)} - A^{(0)}| + |X^{(t)} - X^{(0)}| < \Delta$ **do**

$\quad C_{struct} \leftarrow$ candidate_edge_perturbations$(A^{(t)}, \mathcal{A})$;

$\quad e^* = (u^*, v^*) \leftarrow \underset{e \in C_{struct}}{\arg\max} \; s_{struct}\left(e; G^{(t)}, v_0\right)$;

$\quad C_{feat} \leftarrow$ candidate_feature_perturbations$(X^{(t)}, \mathcal{A})$;

$\quad f^* = (u^*, i^*) \leftarrow \underset{f \in C_{feat}}{\arg\max} \; s_{feat}\left(f; G^{(t)}, v_0\right)$;

$\quad$ **if** $s_{struct}(e^*; G^{(t)}, v_0) > s_{feat}(f^*; G^{(t)}, v_0)$ **then**

$\quad\quad G^{(t+1)} \leftarrow G^{(t)} \pm e^*$;

$\quad$ **else** $G^{(t+1)} \leftarrow G^{(t)} \pm f^*$;

$\quad t \leftarrow t + 1$;

return $: G^{(t)}$

// Train final graph model on the corrupted graph $G^{(t)}$;

---

even more important, we can derive an incremental update – we don't have to recompute the updated $[\hat{A}^2]_{v_0}$ from scratch.

THEOREM 5.1. *Given an adjacency matrix A, and its corresponding matrices $\tilde{A}, \hat{A}^2, \tilde{D}$. Denote with A' the adjacency matrix when adding or removing the element $e = (m, n)$ from A. It holds:*

$$[\hat{A}'^2]_{uv} = \frac{1}{\sqrt{\tilde{d}'_u \tilde{d}'_v}} \left(\sqrt{\tilde{d}_u \tilde{d}_v}[\hat{A}^2]_{uv} - \frac{\tilde{a}_{uv}}{\tilde{d}_u} - \frac{a_{uv}}{\tilde{d}_v} + \frac{a'_{uv}}{\tilde{d}'_v} + \frac{\tilde{a}'_{uv}}{\tilde{d}'_u} - \right.$$
$$\left. - \frac{a_{um}a_{mv}}{\tilde{d}_m} + \frac{a'_{um}a'_{mv}}{\tilde{d}'_m} - \frac{a_{un}a_{nv}}{\tilde{d}_n} + \frac{a'_{un}a'_{nv}}{\tilde{d}'_n} \right) \quad (17)$$

where $\tilde{d}', a',$ and $\tilde{a}',$ are defined as (using the Iverson bracket $\mathbb{I}$):

$$\tilde{d}'_k = \tilde{d}_k + \mathbb{I}[k \in \{m, n\}] \cdot (1 - 2 \cdot a_{mn})$$
$$a'_{kl} = a_{kl} + \mathbb{I}[\{k, l\} = \{m, n\}] \cdot (1 - 2 \cdot a_{kl})$$
$$\tilde{a}'_{kl} = \tilde{a}_{kl} + \mathbb{I}[\{k, l\} = \{m, n\}] \cdot (1 - 2 \cdot \tilde{a}_{kl})$$

PROOF. Let S and S' be defined as $S = \sum_{k=1}^{N} \frac{a_{uk}a_{kv}}{\tilde{d}_k}$ and $S' = \sum_{k=1}^{N} \frac{a'_{uk}a'_{kv}}{\tilde{d}'_k}$. We have $[\hat{A}]_{uv} = \frac{\tilde{a}_{uv}}{\sqrt{\tilde{d}_u \tilde{d}_v}}$. If $u \neq v$, then

$$[\hat{A}^2]_{uv} = \sum_{k=1}^{N} [\hat{A}]_{uk}[\hat{A}]_{kv} = \frac{\tilde{a}_{uv}}{\tilde{d}_u \sqrt{\tilde{d}_u \tilde{d}_v}} + \frac{\tilde{a}_{uv}}{\tilde{d}_v \sqrt{\tilde{d}_u \tilde{d}_v}} + \frac{1}{\sqrt{\tilde{d}_u \tilde{d}_v}} S.$$

Having the above equation for $\hat{A}'^2$, we get

$$[\hat{A}'^2]_{uv}\left(\sqrt{\tilde{d}'_u \tilde{d}'_v}\right) - [\hat{A}^2]_{uv}\left(\sqrt{\tilde{d}_u \tilde{d}_v}\right) =$$
$$\left[\frac{\tilde{a}'_{uv}}{\tilde{d}'_u} - \frac{\tilde{a}_{uv}}{\tilde{d}_u}\right] + \left[\frac{\tilde{a}'_{uv}}{\tilde{d}'_v} - \frac{\tilde{a}_{uv}}{\tilde{d}_v}\right] + (S' - S).$$

After replacing $S' - S = -\frac{a_{um}a_{mv}}{\tilde{d}_m} + \frac{a'_{um}a'_{mv}}{\tilde{d}'_m} - \frac{a_{un}a_{nv}}{\tilde{d}_n} + \frac{a'_{un}a'_{nv}}{\tilde{d}'_n}$ in the above equation, it is straightforward to derive Eq. 17. Deriving this equation for the case $u = v$ is similar. Eq. 17 encompasses both cases. □

Eq. (17) enables us to update the entries in $\hat{A}^2$ in *constant time*; and in a sparse and incremental manner. Remember that all $\tilde{a}_{uv}$, a_{uv}, and a'_{uv} are either 1 or 0, and their corresponding matrices are sparse. Given this highly efficient update of $[\hat{A}^2]_{v_0}$ to $[\hat{A}'^2]_{v_0}$, the updated log-probabilities and, thus, the final score according to Eq. (15) can be easily computed.

Feature attacks. The feature attacks are much easier to realize. Indeed, by fixing the class $c \neq c_{old}$ with currently largest log-probability score $[\hat{A}^2 XW]_{v_0 c}$, the problem is linear in X and every entry of X acts independently. Thus, to find the best node and feature (u^*, i^*) we only need to compute the gradients

$$\Upsilon_{ui} = \frac{\partial}{\partial X_{ui}} \left([\hat{A}^2 XW]_{v_0 c} - [\hat{A}^2 XW]_{v_0 c_{old}}\right)$$
$$= [\hat{A}^2]_{v_0 u} \left([W]_{ic} - [W]_{ic_{old}}\right)$$

and subsequently pick the one with the highest absolute value that points into an allowable direction (e.g. if the feature was 0, the gradient needs to point into the positives). The value of the score function s_{feat} for this best element is then simply obtained by adding $|\Upsilon_{ui}|$ to the current value of the loss function:

$$\mathcal{L}_s(A, X; W, v_0) + |\Upsilon_{ui}| \cdot \mathbb{I}[(2 \cdot X_{ui} - 1) \cdot \Upsilon_{ui} < 0]$$

All this can be done in *constant time* per feature. The elements where the gradient points outside the allowable direction should not be perturbed since they would only hinder the attack – thus, the old score stays unchanged.

5.2 Fast computation of candidate sets

Last, we have to make sure that all perturbations are valid according to the constraints $\hat{\mathcal{P}}_{\Delta, \mathcal{A}}^{G0}$. For this, we defined the sets C_{struct} and C_{feat}. Clearly, the constraints introduced in Eq. 4 and 5 are easy to ensure. The budget constraint Δ is fulfilled by the process of the greedy approach, while the elements which can be perturbed according to Eq. 4 can be precomputed. Likewise, the node-feature combinations fulfilling the co-occurence test of Eq. 12 can be precomputed. Thus, the set C_{feat} only needs to be instantiated once.

The significance test for the degree distribution, however, does not allow such a precomputation since the underlying degree distribution dynamically changes. How can we efficiently check whether a potential perturbation of the edge (m, n) still preserves a similar degree distribution? Indeed, since the individual degrees only interact additively, we can again derive a *constant time* incremental update of our test statistic Λ.

THEOREM 5.2. *Given graph $G = (A, X)$ and the multiset $\mathcal{D}_G$ (see below Eq. 6). Denote with $R^G = \sum_{d_i \in \mathcal{D}_G} \log d_i$ the sum of log degrees. Let $e = (m, n)$ be a candidate edge perturbation, and d_m and d_n the degrees of the nodes in G. For $G' = G \pm e$ we have:*

$$\alpha_{G'} = 1 + n^e \left[R^{G'} - n^e \log\left(d_{\min} - \tfrac{1}{2}\right)\right]^{-1} \quad (18)$$

$$l\left(\mathcal{D}_{G'}\right) = n^e \log \alpha_{G'} + n^e \alpha_{G'} \log d_{\min} + \left(\alpha_{G'} + 1\right) R^{G'} \quad (19)$$
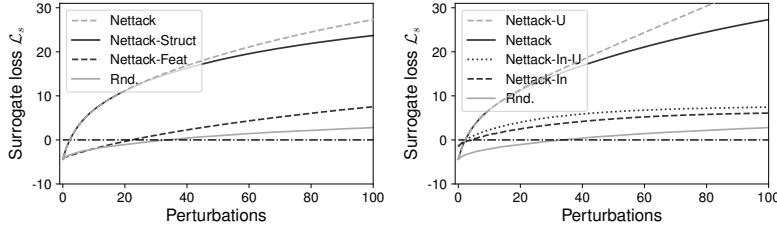
Figure 2: Average surrogate loss for increasing number of perturbations. Different variants of our method on the Cora data. Larger is better.
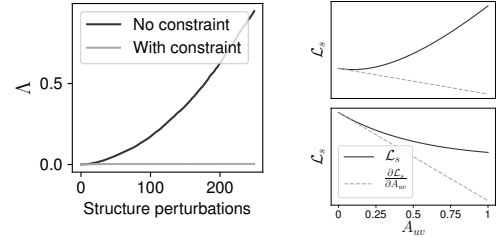
Figure 3: Change in test statistic Λ (degree distr.)

Figure 4: Gradient vs. actual loss

where

$$x = 1 - 2 \cdot a_{mn} \qquad (20)$$

$$n^e = |\mathcal{D}_G| + (\mathbb{I}[d_m + 1 - a_{mn} = d_{\min}] + \mathbb{I}[d_n + 1 - a_{mn} = d_{\min}]) \cdot x$$

$$R^{G'} = R^G - \mathbb{I}[d_m \geq d_{\min}] \log d_m + \mathbb{I}[d_m + x \geq d_{\min}] \log(d_m + x)$$
$$- \mathbb{I}[d_n \geq d_{\min}] \log d_n + \mathbb{I}[d_n + x \geq d_{\min}] \log(d_n + x).$$

PROOF. Firstly, we show that if we incrementally compute n^e according to the update equation of Theorem 5.2, n^e will be equal to $|\mathcal{D}_{G'}|$. The term $\mathbb{I}[d_m + 1 - a_{mn} = d_{min}] \cdot x$ will be activated (i.e. non-zero) only in two cases: 1) $a_{mn} = 1$ (i.e. $G' = G - e$), and $d_m = d_{min}$, then $x < 0$ and the update equation actually removes node m from $\mathcal{D}_G$. 2) $a_{mn} = 0$ (i.e. $G' = G + e$), and $d_m = d_{min} - 1$, then $x > 0$ and the update equation actually adds node m to $\mathcal{D}_G$. A similar argumentation is applicable for node n. Accordingly, we have that $n^e = |\mathcal{D}_{G'}|$.

Similarly, one can show the valid incremental update for $R^{G'}$ considering that only nodes with degree larger than d_{min} are considered and that $d_m + x$ is the new degree. Having incremental updates for n^e and $R^{G'}$, the updates for $\alpha_{G'}$ and $l(\mathcal{D}_{G'})$ follow easily from their definitions. □

Given $G^{(t)}$, we can now incrementally compute $l(\mathcal{D}_{G_e^{(t)}})$, where $G_e^{(t)} = G^{(t)} \pm e$. Equivalently we get incremental updates for $l(\mathcal{D}_{comb})$ after an edge perturbation. Since all r.h.s. of the equations above can be computed in constant time, also the test statistic $\Lambda(G^{(0)}, G_e^{(t)})$ can be computed in *constant time*. Overall, the set of valid candidate edge perturbations at iteration t is $C_{struct} = \{e = (m, n) \mid \Lambda(G^{(0)}, G_e^{(t)}) < \tau \land (m \in \mathcal{A} \lor n \in \mathcal{A})\}$. Since $R^{G^{(t)}}$ can be incrementally updated to $R^{G^{(t+1)}}$ once the best edge perturbation has been performed, the full approach is highly efficient.

5.3 Complexity

The candidate set generation (i.e. which edges/features are allowed to change) and the score functions can be incrementally computed and exploit the graph's sparsity, thus, ensuring scalability. The runtime complexity of the algorithm can easily be determined as:

$$O(\Delta \cdot |\mathcal{A}| \cdot (N \cdot th_{v_0} + D))$$

where th_{v_0} indicates the size of the two-hop neighborhood of the node v_0 during the run of the algorithm.

In every of the Δ many iterations, each attacker evaluates the potential edge perturbations (N at most) and feature perturbations (D

at most). For the former, this requires to update the two-hop neighborhood of the target due to the two convolution layers. Assuming the graph is sparse, th_{v_0} is much smaller than N. The feature perturbations are done in constant time per feature. Since all constraints can be checked in constant time they do not affect the complexity.

6 EXPERIMENTS

We explore how our attacks affect the surrogate model, and evaluate transferability to other models and for multiple datasets. For repeatability, NETTACK's source code is available on our website: https://www.kdd.in.tum.de/nettack.

Dataset	N_{LCC}	E_{LCC}
CORA-ML [22]	2,810	7,981
CITESEER [29]	2,110	3,757
POL. BLOGS [1]	1,222	16,714

Table 1: Dataset statistics. We only consider the largest connected component (LCC).

Setup. We use the well-known CORA-ML and CITESEER networks as in [5], and POLBLOGS [1]. The dataset characteristics are shown in Table 1. We split the network in labeled (20%) and unlabeled nodes (80%). We further split the labeled nodes in equal parts *training* and *validation* sets to train our surrogate model. That is, we remove the labels from the validation set in the training procedure and use them as the stopping criterion (i.e., stop when validation error increases). The labels of the unlabeled nodes are never visible to the surrogate model during training.

We average over five different random initializations/ splits, where for each we perform the following steps. We first train our surrogate model on the labeled data and among all nodes from the test set that have been *correctly* classified, we select (i) the 10 nodes with highest margin of classification, i.e. they are clearly correctly classified, (ii) the 10 nodes with lowest margin (but still correctly classified) and (iii) 20 more nodes randomly. These will serve as the target nodes for our attacks. Then, we corrupt the input graph using the model proposed in this work, called NETTACK for direct attacks, and NETTACK-IN for influence attacks, respectively (picking 5 random nodes as attackers from the neighborhood of the target).

Since no other competitors exist, we compare against two baselines: (i) Fast Gradient Sign Method (FGSM) [14] as a direct attack on v_0 (in our case also making sure that the result is still binary). (ii) RND is an attack in which we modify the structure of the graph. Given our target node v_0, in each step we randomly sample nodes u for which $c_{v_0} \neq c_u$ and add the edge u, v to the graph structure, assuming unequal class labels are hindering classification.

Class: neural networks				Class: theory				Class: probabilistic models			
constrained		**unconstrained**		**constrained**		**unconstrained**		**constrained**		**unconstrained**	
probabilistic	25	efforts	2	driven	3	designer	0	difference	2	calls	1
probability	38	david	0	increase	8	assist	0	solve	3	chemical	0
bayesian	28	averages	2	heuristic	4	disjunctive	7	previously	12	unseen	1
inference	27	accomplished	3	approach	56	interface	1	control	16	corporation	3
probabilities	20	generality	1	describes	20	driven	3	reported	1	fourier	1
observations	9	expectation	10	performing	7	refinement	0	represents	8	expressed	2
estimation	35	specifications	0	allow	11	refines	0	steps	5	robots	0
distributions	21	family	10	functional	2	starts	1	allowing	7	achieving	0
independence	5	uncertain	3	11	3	restrict	0	task	17	difference	2
variant	9	observations	9	acquisition	1	management	0	expressed	2	requirement	1

Table 2: Top-10 feature perturbations per class on Cora

6.1 Attacks on the surrogate model

We start by analyzing different variants of our method by inspecting their influence on the surrogate model. In Fig. 2 (left) we plot the surrogate loss when performing a specific number of perturbations. Note that once the surrogate loss is positive, we realized a successful misclassification. We analyze NETTACK, and variants where we only manipulate features or only the graph structure. As seen, perturbations in the structure lead to a stronger change in the surrogate loss compared to feature attacks. Still, combining both is the most powerful, only requiring around 3 changes to obtain a misclassification. For comparison we have also added RND, which is clearly not able to achieve good performance.

In Fig. 2 (right) we analyze our method when using a direct vs. influencer attack. Clearly, direct attacks need fewer perturbations – still, influencer attacks are also possible, posing a high risk in real life scenarios. The figure also shows the result when *not* using our constraints as proposed in Section 4.1, indicated by the name NETTACK-U. As seen, even when using our constraints, the attack is still succesfull. Thus, unnoticable perturbations can be generated.

It is worth mentioning that the constraints are indeed necessary. Figure 3 shows the test statistic Λ of the resulting graph with or without our constraints. As seen the constraint we impose has an effect on our attack; if not enforced, the power law

Figure 5: Runtime

distribution of the corrupted graph becomes more and more dissimilar to the original graph's. Similarly, Table 2 illustrates the result for the feature perturbations. For CORA-ML, the features correspond to the presence of *words* in the abstracts of papers. For each class (i.e. set of nodes with same label), we plot the top-10 features that have been manipulated by the techniques (these account for roughly 50% of all perturbations). Further, we report for each feature its original occurence within the class. We see that the used features are indeed different – even more, the unconstrained version often uses words which are 'unlikely' for the class (indicated by the small numbers). Using such words can easily be noticed as manipulations, e.g. 'david' in neural networks or 'chemical' in probabilistic models. Our constraint ensures that the changes are more subtle.

Overall, we conclude that attacking the features and structure simultaneously is very powerful; and the introduced constraints do not hinder the attack while generating more realistic perturbations. Direct attacks are clearly easier than influencer attacks.
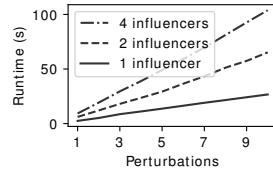
Lastly, even though not our major focus, we want to analyze the required runtime of NETTACK. In line with the derived complexity, in Fig. 5 we see that our algorithm scales linearly with the number of perturbations to the graph structure and the number of influencer nodes considered. Please note that we report runtime for *sequential* processing of candidate edges; this can however be trivially parallelized. Similar results were obtained for the runtime w.r.t. the graph size, matching the complexity analysis.

6.2 Transferability of attacks

After exploring how our attack affects the (fixed) surrogate model, we will now find out whether our attacks are also successful on established deep learning models for graphs. For this, we pursue the approach from before and use a budget of $\Delta = d_{v_0} + 2$, where d_{v_0} is the degree of target node we currently attack. This is motivated by the observation that high-degree nodes are more difficult to attack than low-degree ones. In the following we always report the score $X = Z^*_{v_0, c_{old}} - \max_{c \neq c_{old}} Z^*_{v_0, c}$ using the ground truth label c_{old} of the target. We call X the classification margin. *The smaller X, the better.* For values smaller than 0, the targets get misclassified. Note that this could even happen for the clean graph since the classification itself might not be perfect.

Evasion vs. Poisoning Attack. In Figure 6a we evaluate NETTACK's performance for two attack types: evasion attacks, where the model parameters (here of GCN [19]) are kept fix based on the clean graph; and poisoning attacks, where the model is retrained after the attack (averaged over 10 runs). In the plot, every dot represents one target node. As seen, direct attacks are extremely succesful – even for the challening poisoning case almost every target gets misclassified. We therefore conclude that our surrogate model and loss are a sufficient approximation of the true loss on the non-linear model *after* re-training on the perturbed data. Clearly, influencer attacks (shown right of the double-line) are harder but they still work in both cases. Since poisoining attacks are in general harder and match better the transductive learning scenario, we report in the following only these results.

Comparison. Figure 6b and 6c show that the corruptions generated by NETTACK transfer to different (semi-supervised) graph convolutional methods: GCN [19] and CLN [28]. Most remarkably, even the unsupervised model DeepWalk [27] is strongly affected by our perturbations (Figure 6d). Since DW only handles unattributed graphs, only structural attacks were performed. Following [27], node classification is performed by training a logistic regression on
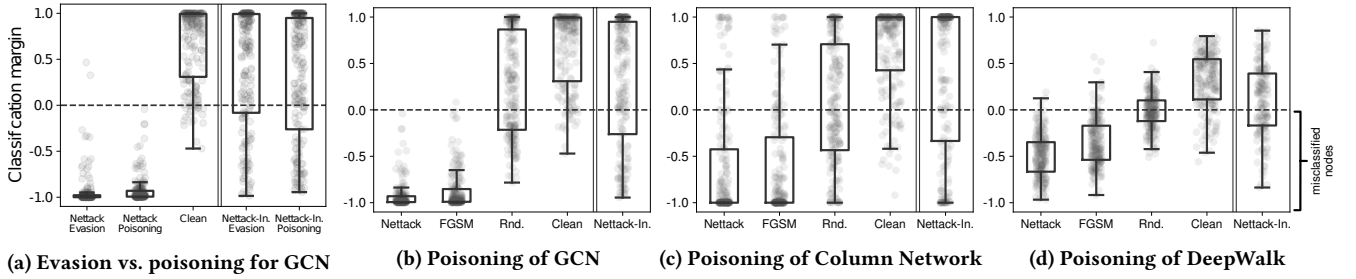
(a) Evasion vs. poisoning for GCN   **(b) Poisoning of GCN**   **(c) Poisoning of Column Network**   **(d) Poisoning of DeepWalk**

Figure 6: Results on Cora data using different attack algorithms. Clean indicates the original data. Lower scores are better.

the learned embeddings. Overall, we see that direct attacks pose a much harder problem than influencer attacks. In these plots, we also compare against the two baselines RND and FGSM, both operating in the direct attack setting. As shown, NETTACK outperforms both. Again note: All these results are obtained using a challenging poisoning attack (i.e. retraining of the model).

In Table 3 we summarize the results for different datasets and classification models. Here, we report the fraction of target nodes that get *correctly classified*. Our adversarial perturbations on the surrogate model are transferable to all three models an on all datasets we evaluated. Not surprisingly, influencer attacks lead to a lower decrease in performance compared to direct attacks.

We see that FGSM performs worse than NETTACK, and we argue that this comes from the fact that gradient methods are not optimal for discrete data. Fig. 4 shows why this is the case: we plot the gradient vs. the actual change in loss when changing elements in A. Often the gradients do not approximate the loss well – in (b) and (c) even the signs do not match. One key advantage of NETTACK is that we can *precisely* and efficiently compute the change in $\mathcal{L}_s$.

Last, we also analyzed how the structure of the target, i.e. its degree, affects the performance.

	[1;5]	[6;10]	[11;20]	[21;100]	[100; ∞)
Clean	0.878	0.823	1.0	1.0	1.0
NETTACK	0.003	0.009	0.014	0.036	0.05

The table shows results for different degree ranges. As seen, high degree nodes are slightly harder to attack: they have both, higher classification accuracy in the clean graph and in the attacked graph.

Limited Knowledge. In the previous experiments, we have assumed *full knowledge* of the input graph, which is a reasonable assumption for a worst-case attack. In Fig. 7 we analyze the result when having limited knowledge: Given a target node v_0, we provided our model only *subgraphs* of increasing size relative to the size of the Cora graph. We constructed these subgraphs by selecting nodes with increasing distance from v_0, i.e. we first selected 1-hop neighbors, then 2-hop neighbors and so on, until we have reached

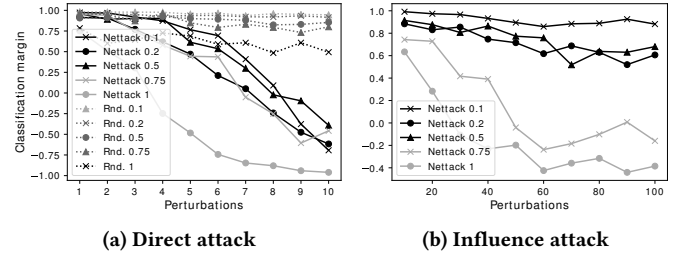**(a) Direct attack**   **(b) Influence attack**

Figure 7: Attacks with limited knowledge about the data

the desired graph size. We then perturbed the subgraphs using the attack strategy proposed in this paper. These perturbations are then taken over to *full* graph, where we trained GCN. *Note that NETTACK has always only seen the subgraph; and its surrogate model is also only trained based on it.*

Fig. 7 shows the result for a direct attack. As seen, even if only 10% of the graph is observed, we can still significantly attack it. Clearly, if the attacker knows the full graph, the fewest number of perturbations is required. For comparison we include the RND attack, also only operating on the subgraphs. In Fig. 7 we see the influence attack. Here we require more perturbations and 75% of the graph size for our attack to succeed. Still, this experiment indicates that full knowledge is not required.

7 CONCLUSION

We presented the first work on adversarial attacks to (attributed) graphs, specifically focusing on the task of node classification via graph convolutional networks. Our attacks target the nodes' features and the graph structure. Exploiting the relational nature of the data, we proposed direct and influencer attacks. To ensure unnoticeable changes even in a discrete, relational domain, we proposed to preserve the graph's degree distribution and feature co-occurrences. Our developed algorithm enables efficient perturbations in a discrete domain. Based on our extensive experiments we can conclude that even the challenging poisoning attack is successful possible with our approach. The classification performance is consistently reduced, even when only partial knowledge of the graph is available or the attack is restricted to a few influencers. Even more, the attacks generalize to other node classification models.

Studying the robustness of deep learning models for graphs is an important problem, and this work provides essential insights for deeper study. As future work we aim to derive extensions of existing models to become more robust against attacks, and we aim to study tasks beyond node classification.

Attack method	Cora			Citeseer			Polblogs		
	GCN	CLN	DW	GCN	CLN	DW	GCN	CLN	DW
Clean	0.90	0.84	0.82	0.88	0.76	0.71	0.93	0.92	0.63
NETTACK	**0.01**	**0.17**	**0.02**	**0.02**	**0.20**	**0.01**	**0.06**	**0.47**	**0.06**
FGSM	0.03	0.18	0.10	0.07	0.23	0.05	0.41	0.55	0.37
RND	0.61	0.52	0.46	0.60	0.52	0.38	0.36	0.56	0.30
NETTACK-IN	0.67	0.68	0.59	0.62	0.54	0.48	0.86	0.62	0.91

Table 3: Overview of results. Smaller is better.

REFERENCES

[1] Lada A Adamic and Natalie Glance. 2005. The political blogosphere and the 2004 US election: divided they blog. In *International workshop on Link discovery*. 36–43.

[2] Alessandro Bessi. 2015. Two samples test for discrete power-law distributions. *arXiv preprint arXiv:1503.00643* (2015).

[3] Battista Biggio, Giorgio Fumera, and Fabio Roli. 2014. Security evaluation of pattern classifiers under attack. *IEEE TKDE* 26, 4 (2014), 984–996.

[4] Aleksandar Bojchevski and Stephan Günnemann. 2018. Bayesian Robust Attributed Graph Clustering: Joint Learning of Partial Anomalies and Group Structure. In *AAAI*. 2738–2745.

[5] Aleksandar Bojchevski and Stephan Günnemann. 2018. Deep Gaussian Embedding of Graphs: Unsupervised Inductive Learning via Ranking. In *ICLR*.

[6] Aleksandar Bojchevski, Yves Matkovic, and Stephan Günnemann. 2017. Robust Spectral Clustering for Noisy Data: Modeling Sparse Corruptions Improves Latent Embeddings. In *SIGKDD*. 737–746.

[7] Hongyun Cai, Vincent W Zheng, and Kevin Chang. 2018. A comprehensive survey of graph embedding: problems, techniques and applications. *IEEE TKDE* (2018).

[8] Olivier Chapelle, Bernhard Schölkopf, and Alexander Zien. 2006. *Semi-Supervised Learning. Adaptive Computation and Machine Learning series.* The MIT Press.

[9] Yizheng Chen, Yacin Nadji, Athanasios Kountouras, Fabian Monrose, Roberto Perdisci, Manos Antonakakis, and Nikolaos Vasiloglou. 2017. Practical Attacks Against Graph-based Clustering. *arXiv preprint arXiv:1708.09056* (2017).

[10] Aaron Clauset, Cosma Rohilla Shalizi, and Mark EJ Newman. 2009. Power-law distributions in empirical data. *SIAM review* 51, 4 (2009), 661–703.

[11] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *NIPS*. 3837–3845.

[12] Dhivya Eswaran, Stephan Günnemann, Christos Faloutsos, Disha Makhija, and Mohit Kumar. 2017. ZooBP: Belief Propagation for Heterogeneous Networks. *PVLDB* 10, 5 (2017), 625–636.

[13] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. 2017. Neural Message Passing for Quantum Chemistry. In *ICML*. 1263–1272.

[14] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and harnessing adversarial examples. In *ICLR*.

[15] Kathrin Grosse, Nicolas Papernot, Praveen Manoharan, Michael Backes, and Patrick McDaniel. 2017. Adversarial Examples for Malware Detection. In *European Symposium on Research in Computer Security*. 62–79.

[16] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *SIGKDD*. 855–864.

[17] William L Hamilton, Rex Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *NIPS*.

[18] Bryan Hooi, Neil Shah, Alex Beutel, Stephan Günnemann, Leman Akoglu, Mohit Kumar, Disha Makhija, and Christos Faloutsos. 2016. BIRDNEST: Bayesian Inference for Ratings-Fraud Detection. In *SIAM SDM*. 495–503.

[19] Thomas N Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. In *ICLR*.

[20] Bo Li, Yining Wang, Aarti Singh, and Yevgeniy Vorobeychik. 2016. Data poisoning attacks on factorization-based collaborative filtering. In *NIPS*. 1885–1893.

[21] Ben London and Lise Getoor. 2014. Collective Classification of Network Data. *Data Classification: Algorithms and Applications* 399 (2014).

[22] Andrew Kachites McCallum, Kamal Nigam, Jason Rennie, and Kristie Seymore. 2000. Automating the construction of internet portals with machine learning. *Information Retrieval* 3, 2 (2000), 127–163.

[23] Shike Mei and Xiaojin Zhu. 2015. Using Machine Teaching to Identify Optimal Training-Set Attacks on Machine Learners. In *AAAI*. 2871–2877.

[24] Ahmed Mohamed Mohamed El-Sayed. 2016. Modeling Multivariate Correlated Binary Data. *American Journal of Theoretical and Applied Statistics* 5, 4 (2016), 225–233.

[25] Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodola, Jan Svoboda, and Michael M Bronstein. 2017. Geometric deep learning on graphs and manifolds using mixture model CNNs. In *CVPR*, Vol. 1. 3.

[26] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. 2016. The limitations of deep learning in adversarial settings. In *IEEE European Symposium on Security and Privacy*. 372–387.

[27] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *SIGKDD*. 701–710.

[28] Trang Pham, Truyen Tran, Dinh Q. Phung, and Svetha Venkatesh. 2017. Column Networks for Collective Classification. In *AAAI*. 2485–2491.

[29] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. 2008. Collective classification in network data. *AI magazine* 29, 3 (2008), 93.

[30] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Google Inc, Joan Bruna, Dumitru Erhan, Google Inc, Ian Goodfellow, and Rob Fergus. 2014. Intriguing properties of neural networks. In *ICLR*.

[31] Mohamad Ali Torkamani and Daniel Lowd. 2013. Convex adversarial collective classification. In *ICML*. 642–650.

[32] Florian Tramèr, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. 2017. The Space of Transferable Adversarial Examples. *arXiv preprint arXiv:1704.03453* (2017).

[33] Mengchen Zhao, Bo An, Yaodong Yu, Sulin Liu, and Sinno Jialin Pan. 2018. Data Poisoning Attacks on Multi-Task Relationship Learning. In *AAAI*. 2628–2635.