Robust Training of Graph Convolutional Networks via Latent Perturbation

Hongwei Jin[1] and Xinhua Zhang[1]

University of Illinois at Chicago, Chicago IL 60607, USA
{hjin25, zhangx}@uic.edu

Abstract. Despite the recent success of graph convolutional networks (GCNs) in modeling graph structured data, its vulnerability to adversarial attacks has been revealed and attacks on both node feature and graph structure have been designed. Direct extension of defense algorithms based on adversarial samples meets with immediate challenge because computing the adversarial network costs substantially. We propose addressing this issue by perturbing the latent representations in GCNs, which not only dispenses with generating adversarial networks, but also attains improved robustness and accuracy by respecting the latent manifold of the data. This new framework of latent adversarial training on graphs is applied to node classification, link prediction, and recommender systems. Our empirical experimental results confirm the superior robustness performance over strong baselines.

Keywords: Graph Neural Network · Adversarial Training · Representation Learning

1 Introduction

Neural networks have achieved great success on Euclidean data for image recognition, machine translation, and speech recognition, etc. However, modeling with non-Euclidean data—such as complex networks with geometric information and structural manifold—is more challenging in terms of data representation. Mapping non-Euclidean data to Euclidean, which is also referred to as embedding, is one of the most prevalent techniques. Recently, graph convolutional networks (GCNs) have received increased popularity in machine learning for structured data. The first phenomenal work of GCN was presented by Bruna *et al.* (2013), which developed a set of graph convolutional operations based on spectral graph theory. The conventional GCN was introduced by Kipf and Welling (2017) for the task of node classification, and they represent nodes by repeated multiplication of augmented normalized adjacency matrix and feature matrix, which can be interpreted as the first order approximation of localized spectral filters on graphs (Hammond *et al.*, 2011; Defferrard *et al.*, 2016). GCNs have been widely applied to a variety of machine learning tasks, including node classification (Kipf and Welling, 2017), graph clustering (Duvenaud *et al.*, 2015), link prediction (Kipf and Welling, 2016; Schlichtkrull *et al.*, 2018), recommender systems (Berg *et al.*, 2018), etc.

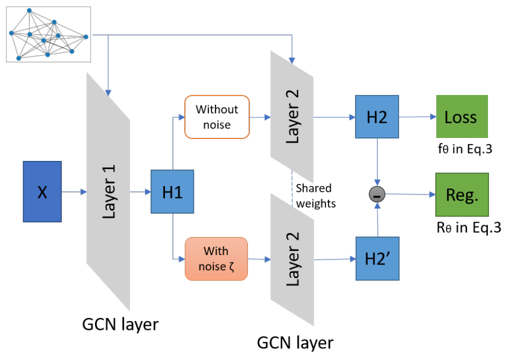

Fig. 1: LAT-GCN Framework

Despite the advantages in efficiently and effectively learning representations and predictions, GCNs have been shown vulnerable to adversarial attacks. Although adversarial learning has achieved significant progress in recent years (Szegedy *et al.*, 2014), the graph structure in GCNs constitutes an additional source of vulnerability. The conventional approaches based on adversarial samples (a.k.a. attacks) are typically motivated by adding imperceptible perturbation to images, followed by enforcing the invariance of prediction outputs (Kurakin *et al.*, 2017). This corresponds to perturbing node features in GCNs and has received very recent study. Feng *et al.* (2019) introduced the graph adversarial training as a dynamic regularization scheme based on graph structure. Deng *et al.* (2019) proposed a sample-based batch virtual adversarial training to promote the smoothness of model.

However, the graph topology itself can be subject to attacks such as adding or deleting edges or nodes. Zügner *et al.* (2018) and Dai *et al.* (2018) constructed effective structural attacks at both training time (poisoning) and testing time (evasion). Finding the adversarial input graph is indeed a combinatorial optimization problem that is typically NP-hard. Dai *et al.* (2018) proposed a reinforcement learning based attack that learns a generalizable attack policy to mis-classify a target in both graph classification and node classification. Zügner *et al.* (2018) introduced a surrogate model to approximate the perturbed graph structure and feature. Both of these methods considered attacks at the test stage, i.e., evasion attacks. By solving a bilevel problem from meta-learning, Zügner and Günnemann (2019) generated adversarial examples that are graph structures, and in contrast to the previous attacks, they do not need to specify the target. Their attack modifies the training data to undermine the performance, hence a poison attack. Xu *et al.* (2019) proposed two attack methods: PGD topology attack and min-max attack methods, which relaxed $\{0,1\}^n$ to $[0,1]^n$, leading to a continuous optimization. They also proposed a robust training model by solving a min-max problem.

Although the technique of adversarial sample can be directly applied to defend against structural attacks, an immediate obstacle arises from computational complexity. All the aforementioned structural attacks are much more computation intensive than conventional attacks in image classification. Therefore alternating between model optimization and adversarial sample generation can be prohibitively time consuming, making a new solution principle in demand.

The goal of this paper, therefore, is to develop a new adversarial training algorithm that defends against attacks on both node features *and* graph structure, while at the same time maintaining or even improving the generalization accuracy. Our intuition draws upon two prior works. Firstly, in the context of adversarial training on word inputs, Miyato *et al.* (2017) noted that words are discrete and are not amenable to infinitesimal perturbation. So they resorted to perturbing the word embeddings that are continuous.

A straightforward analogy of word embeddings in GCNs is the first layer output $\mathbf{H}^{(1)}$ after graph convolution, which blends the information of both node features and the graph. So we propose injecting adversarial perturbations to $\mathbf{H}^{(1)}$, as shown by $\boldsymbol{\zeta}$ in Figure 1. This leads to indirect perturbations to the graph, which implicitly enforces robustness to structural attacks. As shown in Section 3.1, this can be achieved via a regularization term on $\mathbf{H}^{(1)}$, completely circumventing the requirement of generating adversarial attacks on the graph structure. We will refer to the approach as latent adversarial training (LAT-GCN).

However, Miyato *et al.* (2017) also noted that "the perturbed embedding does not map to any word" and "we thus propose this approach exclusively as a means of regularizing a text classifier". To address the analogous concern in GCNs, we leverage the observation by Stutz *et al.* (2019) that adversarial examples can benefit both robustness and accuracy if they are on the manifold of low-dimensional embeddings. Using $\mathbf{H}^{(1)}$ as a proxy of the latent manifold, LAT-GCN manages to generate "on-manifold" perturbations, which, as our experiments show in Section 4, help to reduce the success rate of adversarial attacks for GCNs while preserving or improving the accuracy of the model.

2 Related Work

Vulnerability of deep neural networks has received intensive study in machine learning (Szegedy *et al.*, 2014; Huang *et al.*, 2017; Song *et al.*, 2018; Jia and Liang, 2017; He *et al.*, 2017). Some certifications of robustness have been established, such as Sinha *et al.* (2018), Raghunathan *et al.* (2018), and Wong and Kolter (2018). Recently, investigations have been made on the important trade-off between adversarial robustness and generalization performance. Tsipras *et al.* (2019) showed that robustness may be at odds with accuracy, and a principled trade-off was studied by Zhang *et al.* (2019), which decomposed the prediction error for adversarial examples (robust error) into classification error and boundary error, and a differentiable upper bound was derived by leveraging classification calibration. A number of works also analyzed the robust error *in terms of* generalization error (e.g., Schmidt *et al.*, 2018; Cullina *et al.*, 2018; Yin *et al.*, 2018).

However all these analyses are on *continuous* input domains, typically with "imperceptible perturbations". In contrast, our focus is on adversarial robustness with respect to *discrete* objects such as network structures. Although it is hard to extend the aforementioned theoretical trade-off to the new setting, we demonstrate empirically that our novel adversarial training algorithm is able to improve both robustness and accuracy for graph data learning algorithms that are based on node embeddings, e.g., through GCNs.

2.1 Adversarial attacks on link prediction

Since GCNs embed both the structure and feature information simultaneously, different adversarial attacks have been proposed recently. For the task of link prediction, Kipf and Welling (2016) introduced a GCN-based graph auto-encoder (GAE) which reconstructs the adjacency matrix from node embeddings produced by GCNs, and it outperformed spectral clustering (Tang and Liu, 2011) and DeepWalk (Perozzi *et al.*, 2014). Following GAE, Tran (2018) proposed an architecture to learn a joint representation of local graph structure and available node features for multi-task learning in both link prediction and semi-supervised node classification.

Unfortunately, GAEs are still short of robustness under adversarial link attacks. Chen *et al.* (2018b) proposed an iterative gradient attack based on the gradient information in the trained model, and GAEs were shown vulnerable to just a few link perturbations. In order to improve the robustness, Pan *et al.* (2019) designed an adversarial training method by virtually attacking the node features so that their latent representation matches a prior Gaussian distribution. Xu *et al.* (2019) proposed two attack methods: PGD topology attack and min-max attack methods, which relaxed $\{0,1\}^n$ to $[0,1]^n$, leading to a continuous optimization. They also proposed a robust training model by solving a min-max problem. All the above-mentioned approaches rely on adversarial examples to improve the robustness of the model for either node classification or link prediction. However, their methods need to generate the adversarial instance, either at the attack stage or at the training stage. In contract, LAT-GCN does not look for the exact adversarial examples, which is much less time consuming.

Naturally, the same idea can be applied to link prediction amounting to similar robustness and improvement in generalization accuracy, because GAE reconstructs the adjacency matrix based on *node embeddings*, which in turn employs GCNs as the encoder.

Parallel to our work, Wang *et al.* (2019) proposed DefNet to defend against adversarial attacks on GCNs in a conditional manner. It investigates vulnerabilities via graph encoder refining, and addresses discrete attacks via adversarial contractive learning. Both our method and theirs can reduce the attack success rate to about 60% on node classification tasks. However, our approach is much simpler and is more generally applicable to other tasks such as link prediction.

Notation. We denote the set of vertices for a graph G as $\mathcal{V}$, and denote the feature matrix as $\mathbf{X}\in\mathbb{R}^{n\times d}$, where $n=|\mathcal{V}|$ and d is the number of node features. Let $\mathcal{E}$ be the set of existing edges, and $\mathbf{A}\in\mathbb{R}^{n\times n}$ be the adjacency matrix, whose entries are 0 or 1 for unweighted graphs. After adding a self loop to each node, we have $\hat{\mathbf{A}}=\mathbf{A}+\mathbf{I}$, and we construct a diagonal matrix with $\hat{\mathbf{D}}_{ii}=\sum_j \hat{\mathbf{A}}_{ij}$. The augmented normalized adjacency matrix is then defined as $\tilde{\mathbf{A}}=\hat{\mathbf{D}}^{-1/2}\hat{\mathbf{A}}\hat{\mathbf{D}}^{-1/2}$.

3 Latent Adversarial Training of GCNs

The original GCN model employs a forward propagation of representation as:

$$\mathbf{H}^{(l+1)}=\sigma(\tilde{\mathbf{A}}\mathbf{H}^{(l)}\mathbf{W}^{(l+1)}),\quad l\geq 0, \tag{1}$$

where the initial node representation is $\mathbf{H}^{(0)} = \mathbf{X}$. Without loss of generality, let us consider a two-layer GCN, which is commonly used in practice. Then the standard GCN tries to find the optimal weights $\boldsymbol{\theta} := (\mathbf{W}^{(1)}, \mathbf{W}^{(2)})$ that minimize some loss $f_{\boldsymbol{\theta}}$ (e.g., cross-entropy loss) over the latent representation $\mathbf{H}^{(2)}$. That is, $\min_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}}(G, \mathbf{X})$.

In order to improve the robustness to the perturbation in input feature $\mathbf{X}$, Feng *et al.* (2019) and Deng *et al.* (2019) considered *generative adversarial training* over $\mathbf{X}$, which at a high level, optimizes $\min_{\boldsymbol{\theta}} \max_{\mathbf{r} \in C} f_{\boldsymbol{\theta}}(G, \mathbf{X} + \mathbf{r})$. Here $\mathbf{r}$ is the perturbation chosen from a constrained domain C.

Naturally, it is also desirable to defend against attacks on the graph topology of G. Such structural attacks have been studied in Dai *et al.* (2018); Zügner *et al.* (2018); Zügner and Günnemann (2019), but no corresponding defense algorithm has been proposed yet. Different from attacks on $\mathbf{X}$, here the attacks on G are discrete (hence not imperceptible per se), and finding the most effective attacks can be NP-hard. This creates new obstacles to generative adversarial training, and therefore we resort to a regularization approach based on the *latent layer* $\mathbf{H}^{(1)}$.

Specifically, considering that the information in G and $\mathbf{X}$ is summarized in the first layer output $\mathbf{H}^{(1)}$, we can adopt generative adversarial training directly on $\mathbf{H}^{(1)}$:

$$\min_{\boldsymbol{\theta}} \max_{\boldsymbol{\zeta} \in D} \; f_{\boldsymbol{\theta}}(\mathbf{H}^{(1)} + \boldsymbol{\zeta}), \tag{2}$$

where the symbol $f_{\boldsymbol{\theta}}$ is overloaded to denote the loss based on $\mathbf{H}^{(1)}$ with perturbation $\boldsymbol{\zeta}$. The benefits are two folds. Firstly, $\mathbf{H}^{(1)}$ is continuously valued, making it meaningful to apply small perturbations to it, which indirectly represent the perturbations in $\mathbf{X}$ *and* G. Secondly, Stutz *et al.* (2019) argued that perturbation at latent layers (such as $\mathbf{H}^{(1)}$) are more likely to generate "on-manifold" samples that additionally benefits generalization, while perturbations in the raw input space (e.g., $\mathbf{X}$) is likely to deviate from the original data manifold and hence irrelevant to generalization.

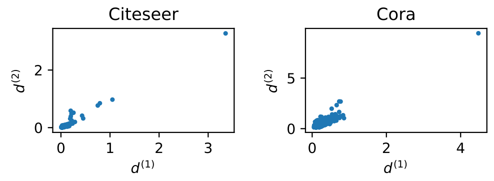

Fig. 2: Scatter plot of the L_2 displacement of hidden representation before and after Nettack, for both first and second layers. Each dot corresponds to a node. Citeseer: $\mathbf{d}^{(1)}$: 0.048 ± 0.084, $\mathbf{d}^{(2)}$: 0.038 ± 0.083, Cora: $\mathbf{d}^{(1)}$: 0.216 ± 0.122, $\mathbf{d}^{(2)}$: 0.486 ± 0.265. The single outlier (top right) corresponds to the targeted node.

Unfortunately, the perturbation $\boldsymbol{\zeta}$ in (2) is chosen *jointly* over *all* nodes in the graph setting, which is different from the common adversarial setting where each individual example seeks its own perturbation independently. As a result, the computational cost is higher, which is further exacerbated by the nested min-max optimization. To alleviate this problem, we adopt the standard regularization variant of adversarial training, which aims to promote the smoothness of model predictions with respect to the perturbations:

$$\min_{\boldsymbol{\theta}} \mathcal{L}_{\boldsymbol{\theta}}(\tilde{\mathbf{A}}, \mathbf{X}) := f_{\boldsymbol{\theta}}(\mathbf{H}^{(1)}) + \gamma \mathcal{R}_{\boldsymbol{\theta}}(\mathbf{H}^{(1)}), \tag{3}$$

where $\gamma \geq 0$ is a trade-off parameter, and the regularizer $\mathcal{R}_{\boldsymbol{\theta}}$ is defined as the Frobenius distance between the original second layer output and that after perturbing $\mathbf{H}^{(1)}$:

$$\mathcal{R}_{\boldsymbol{\theta}}(\mathbf{H}^{(1)}) = \max_{\boldsymbol{\zeta} \in D} \left\| \tilde{\mathbf{A}} \left(\mathbf{H}^{(1)} + \boldsymbol{\zeta} \right) \mathbf{W}^{(2)} - \tilde{\mathbf{A}} \mathbf{H}^{(1)} \mathbf{W}^{(2)} \right\|_F^2. \tag{4}$$

Here we constrain the perturbed noise to be imperceptible via $D := \{\boldsymbol{\zeta} : \|\boldsymbol{\zeta}_{i:}\|_2 \leq \epsilon, \forall i \in \{1, \ldots, n\}\}$, where $\boldsymbol{\zeta}_{i:}$ stands for the i-th row of $\boldsymbol{\zeta}$. This **row-wise** $L_{2,\infty}$ regularization on $\boldsymbol{\zeta}$ is critical for good performance. As we observed in practice, changing it to the Frobenius norm of $\boldsymbol{\zeta}$ leads to significantly inferior performance.

To gain a deeper insight into why this $L_{2,\infty}$ norm on $\boldsymbol{\zeta}$ outperforms Frobenius norm, we empirically examined the change of hidden representations after applying Nettack (Zügner *et al.*, 2018). With a target node randomly selected, we define

$$\mathbf{d}_i^{(l)} = \left\| \mathbf{H}_{i:}^{(l)} - \mathbf{H}_{i:}^{'(l)} \right\|_2 \quad \forall i \in \mathcal{V},$$

where $\mathbf{H}^{(l)}$ and $\mathbf{H}^{'(l)}$ are the hidden representations of the l-th layer upon the completion of vanilla GCN training, based on the graphs before and after applying Nettack, respectively. The changes in L_2 norm are demonstrated in Figure 2 where each dot corresponds to a node, and the outlier in the top-right corner corresponds to the target node. Interestingly, most nodes undergo very little change (including neighbors of the target), while only the target node suffers a large change. Such an imbalanced distribution indicates that it is more reasonable to consider the largest norm of changes, rather than their sum or average (as in the Frobenius norm).

Although (3) is still a min-max problem, the inner maximization problem in $\mathcal{R}_{\boldsymbol{\theta}}$ is now decoupled with the loss $f_{\boldsymbol{\theta}}$, hence solvable with much ease. Indeed, both the objective and the constraint are quadratic, permitting efficient computation of gradient and projection. We also note in passing that turning the adversarial objective (2) into a regularized objective (3) is a commonly adopted technique, and their relationship has been studied by, e.g., Shafieezadeh-Abadeh *et al.* (2017).

3.1 Optimization

Since the objective (3) intrinsically couples all nodes, we apply ADAM to find the optimal $\boldsymbol{\theta}$ using the entire dataset as the mini-batch (Kingma and Ba, 2015). The major complexity stems from $\nabla_{\boldsymbol{\theta}} \mathcal{R}_{\boldsymbol{\theta}}(\mathbf{H}^{(1)})$, which can be readily computed using the Danskin's theorem (Bertsekas, 1995). To this end, we first simplify (4) by

$$\mathcal{R}_{\boldsymbol{\theta}}(\mathbf{H}^{(1)}) = \max_{\boldsymbol{\zeta}} \left\| \tilde{\mathbf{A}} \boldsymbol{\zeta} \mathbf{W}^{(2)} \right\|_F^2 \quad \text{s.t.} \quad \|\boldsymbol{\zeta}_{i:}\| \leq \epsilon. \tag{5}$$

Once we find the optimal $\boldsymbol{\zeta}^*$, the gradient in $\boldsymbol{\theta}$ is simply $\nabla_{\boldsymbol{\theta}} \left\| \tilde{\mathbf{A}} \boldsymbol{\zeta}^* \mathbf{W}^{(2)} \right\|_F^2$. To find $\boldsymbol{\zeta}^*$, note the gradient in $\boldsymbol{\zeta}$ is

Algorithm 1 Latent adversarial training for GCN

```
input $\mathbf{A}, \mathbf{X}$
1: while not converged for (3) do
2:     while not converged for (5) do
3:         Apply ADAM to find $\boldsymbol{\zeta}^*$ using the gradient in $\boldsymbol{\zeta}$ computed from Eq (6).
4:     end while
5:     Take one step of ADAM in $\boldsymbol{\theta}$ with the gradient computed by $\nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}}(\mathbf{H}^{(1)}) + \gamma \nabla_{\boldsymbol{\theta}} \left\| \tilde{\mathbf{A}} \boldsymbol{\zeta}^* \mathbf{W}^{(2)} \right\|_F^2$.
6: end while
```

$$\nabla_{\boldsymbol{\zeta}} \operatorname{tr}(\tilde{\mathbf{A}} \boldsymbol{\zeta} \mathbf{W} \boldsymbol{\zeta}^\top \tilde{\mathbf{A}}^\top) = \left(\mathbf{W}^\top \boldsymbol{\zeta} + \mathbf{W} \boldsymbol{\zeta}^\top \right) \tilde{\mathbf{A}} \tilde{\mathbf{A}}^\top, \qquad (6)$$

where $\mathbf{W} = \mathbf{W}^{(2)} \mathbf{W}^{(2)}{}^\top$. Although the constraint $\|\boldsymbol{\zeta}_{i:}\| \leq \epsilon$ is convex and projection to it is trivial, the objective *maximizes* a convex function. So we simply use ADAM to approximately solve for $\boldsymbol{\zeta}$. Empirical results in Table 1 show that the additional optimization does not incur much computation. ADV-GCN is the classical adversarial training where we sampled 20 nodes from the training set, which were successively taken as the target for Nettack Zügner *et al.* (2018) to find 20 adversarial graphs. And MIN-MAX GCN is the adversarial training algorithm in Xu *et al.* (2019) which is based on a node attack algorithm CE-PDG with $\epsilon = 5\%$. Since the noise $\boldsymbol{\zeta}$ is applied to all the nodes, Eq (5) becomes more expensive to solve as the number of node grows. The overall procedure is summarized in Algorithm 1.

Table 1: CPU wall-clock time in seconds (200 epochs on CPU only)

	$\lvert\mathcal{V}\rvert$	$\lvert\mathcal{E}\rvert$	GCN	ADV-GCN	MIN-MAX GCN	LAT-GCN
Citeseer	2110	3668	7.01	13.3	4012.2	25.49
Cora	2485	5069	6.26	15.7	1823.7	23.90
PubMed	19717	44324	31.18	53.4	out of memory	133.64

4 Experiment 1: Node Classification

We tested the performance of LAT-GCN on a range of standard citation datasets for node classification, including CiteSeer, Cora, and PubMed (Sen *et al.*, 2008). The competing baselines include the vanilla GCN, FastGCN (Chen *et al.*, 2018a), SGCN (Chen *et al.*, 2017), SGC (Wu *et al.*, 2019) and GAT (Veličković *et al.*, 2018). All hyper-parameters in respective models followed from the original implementation, including step size, width of layers, etc. Since the optimal objective value in (5) is quadratic in ϵ, only the value of $\gamma\epsilon^2$ matters for LAT-GCN. So we fixed $\gamma = 0.1$, and only tuned ϵ.

Finally, all algorithms were applied in a transductive setting, where the graph was constructed by combining the nodes for training and testing. Accordingly, perturbation $\boldsymbol{\zeta}$ was applied to both training and test nodes in (4) for training LAT-GCN.

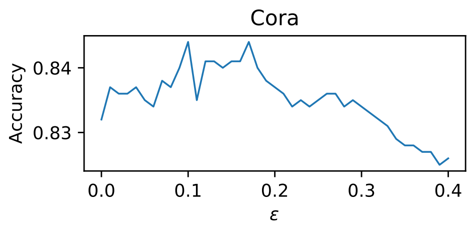

Fig. 3: Test accuracy of LAT-GCN on Cora as a function of ϵ

Table 2: Test accuracy $(\%)$ for node prediction

	CiteSeer	Cora	PubMed
GCN	$70.9 \pm .5$	$81.4 \pm .4$	**79.0** $\pm .4$
FastGCN	$68.8 \pm .6$	$79.8 \pm .3$	$77.4 \pm .3$
SGCN	$70.8 \pm .1$	$81.7 \pm .5$	**79.0** $\pm .4$
SGC	$71.9 \pm .1$	$81.0 \pm .0$	$78.9 \pm .0$
GAT	**72.5** $\pm .7$	**83.0** $\pm .7$	**79.0** $\pm .3$
LAT-GCN	$72.1 \pm .4$	$82.3 \pm .3$	$78.8 \pm .7$

Table 3: Test accuracy $(\%)$ with different settings of LAT-GCN. Subscript indicates the layer(s) perturbed in LAT-GCN.

	GCN	LAT-GCN$_1$	LAT-GCN$_2$	LAT-GCN$_{1,2}$
Citeseer	71.6	72.5	71.9	72.2
Cora	83.4	84.4	84.0	84.2
PubMed	82.8	85.6	84.2	84.4

Comparison of accuracy. We first compared the test accuracy as shown in Table 2. Each test was based on randomly partitioning the nodes into training and testing for 20 times. Clearly, LAT-GCN delivers similar accuracy compared with all the competing algorithms.

In order to study the influence of ϵ on the performance of LAT-GCN, we plotted in Figure 3 how the test accuracy changes with the value of ϵ. We again used the Cora dataset for training. Interestingly, the accuracy tends to increase as ϵ grows from 0 to 0.17, and then drops for larger ϵ. This is consistent with the observation in Stutz *et al.* (2019) where robustness is shown to be positively correlated with generalization, as long as the data points are not perturbed away from the latent manifold. In addition, we also measured the influence of attacking two layers instead of just the first one. Table 3 shows the test accuracy under the four combinations of settings. Perturbing both layers has similar performance as when only the first layer is perturbed. So for the benefit of computational cost, our model only perturbs the first layer.

We also varied the training set size in $\{10\%, 20\%, \ldots, 80\%\}$ of the entire dataset, and plotted how the hyperparameter ϵ affects the test accuracy. There is no perturbation when $\epsilon = 0$ or $\gamma = 0$.

Similar trends as above were observed in the results for Cora and CiteSeer, relegated to Figures 4 and 5, respectively.

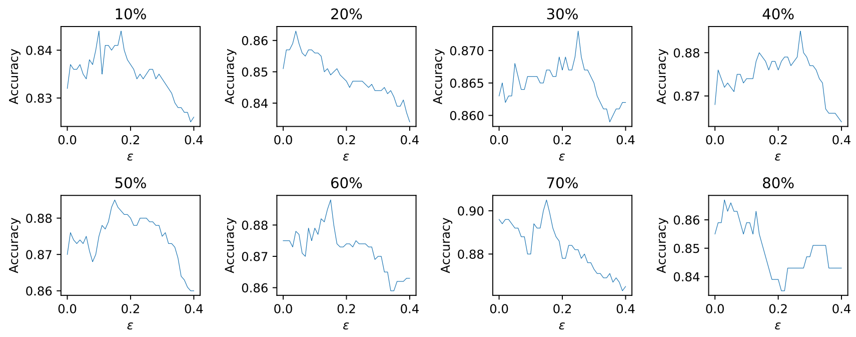

Fig. 4: Test accuracy of LAT-GCN on Cora as a function of ϵ, over different sizes of training data.

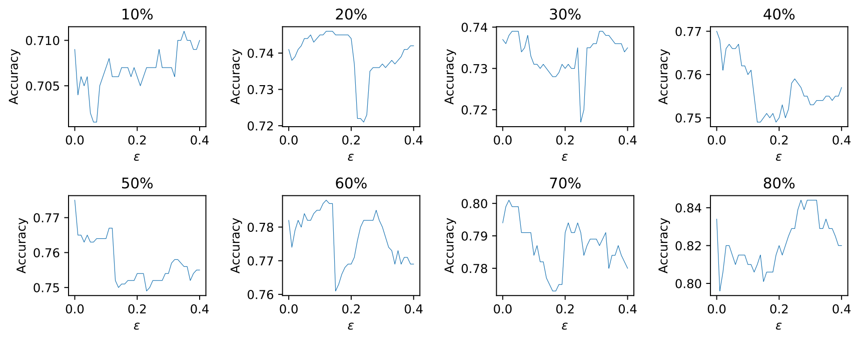

Fig. 5: Test accuracy of LAT-GCN on Citeseer as a function of ϵ, over different sizes of training data.

Comparison of robustness. We next evaluated the robustness of LAT-GCN under Nettack, an evasion attack on graphs that was specifically designed for GCNs (Zügner *et al.*, 2018). The goal is to perform small perturbations on graph $G = (\mathbf{A}, \mathbf{X})$, leading to a new graph $G' = (\mathbf{A}', \mathbf{X}')$, such that the classification performance drops as much as possible. Analogous to "imperceptible perturbations", Nettack imposes a budget on the number of allowed changes:

$$\sum_u \sum_i |\mathbf{X}_{ui} - \mathbf{X}'_{ui}| + \sum_{u<v} |\mathbf{A}_{uv} - \mathbf{A}'_{uv}| \leq \Delta.$$

Since the logits in prediction are complicated by the nonlinear activation function σ, Zügner *et al.* (2018) first introduced a surrogate model that replaces σ with the identity function, leading to a linearized GCN:

$$\mathcal{L}_s(\mathbf{A}, \mathbf{X}; \mathbf{W}, v_o) = \max_{c \neq c_o} \left[\hat{\mathbf{A}}^2 \mathbf{X}\mathbf{W}\right]_{v_o, c} - \left[\hat{\mathbf{A}}^2 \mathbf{X}\mathbf{W}\right]_{v_o, c_o},$$

where v_o is the target node of attack, and c_o is the original label of v_o. Now the aim becomes to solve $\operatorname{argmax}_{(\mathbf{A}', \mathbf{X}')} \mathcal{L}_s(\mathbf{A}', \mathbf{X}'; \mathbf{W}, v_o)$.

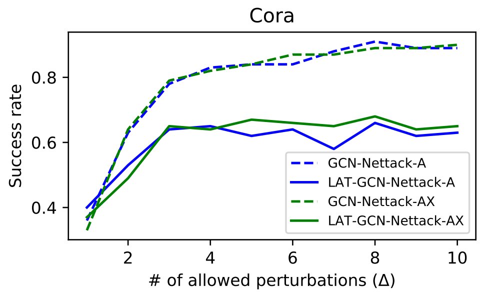

Fig. 6: Success rate on Cora with increasing value of Δ

However, this problem is still intractable because the domain is discrete. To simplify, they defined scoring functions that evaluate the surrogate loss obtained after adding/deleting an edge e or a feature f:

$$\mathcal{S}_{struct}(e; G, v_o) := \mathcal{L}_s(\mathbf{A}', \mathbf{X}; \mathbf{W}, v_o),$$
$$\mathcal{S}_{feature}(f; G, v_o) := \mathcal{L}_s(\mathbf{A}, \mathbf{X}'; \mathbf{W}, v_o).$$

Then Nettack proceeds iteratively, where each step greedily selects an edge e or a feature f which allow either $\mathcal{S}_{struct}$ or $\mathcal{S}_{feature}$ to be maximally increased. The algorithm terminates when the budget is depleted. Since our focus here is on structural attacks, we only considered attacking the structure $\mathbf{A}$, or attacking both structure $\mathbf{A}$ and feature $\mathbf{X}$. We will refer to our results as LAT-GCN-A and LAT-GCN-AX, respectively. ϵ was set to 0.1 for LAT-GCN.

We first compared LAT-GCN to vanilla GCN when the perturbation budget Δ in Nettack was increased from 1 to 10. The test procedure is detailed in Algorithm 2, where the performance metric is the attack success rate. Since Nettack is targeted, we randomly sampled 100 nodes from the test set as the target.

From Figure 6, it is clear that LAT-GCN enjoys significantly lower success rate than GCN on the Cora dataset, under both attack strategies.

Moreover, we compare the success rate with different robust training models. We compared our method (LAT-GCN) with the robust training method in Xu *et al.* (2019) (MIN-MAX GCN) which is based on a node attack algorithm CE-PDG with $\epsilon = 5\%$. We obtained robust models from both methods, and then evaluated them by running Algorithm 2. In particular, we set the hidden unit to 32 and $\epsilon = 5\%$ when training the robust models. We reported the results in Table 4 where LAT-GCN clearly outperforms other methods in robustness.

Table 4: Success Rate (%)

	GCN	ADV-GCN	MIN-MAX GCN	LAT-GCN
Cora	87	84	71	62
Citeseer	84	82	73	67
PubMed	85	79	out of memory	75

Table 5: AUC and AP scores on the test set for link prediction. GCN-1 and GCN-2 stand for GCN with one and two hidden layers, respectively. The second to the fourth columns under GCN-1 show the test performance when edges in $\mathcal{E}_{train}$ are added or removed randomly under a budget ρ in the perturbation of $\mathbf{A}$, i.e., $\|\mathbf{A}' - \mathbf{A}\|_1 \leq \rho |\mathcal{V}|^2$.

		SC	DW	ARGA	GCN-1				GCN-2	LAT-GCN
					$\rho = 0$	$\rho = 0.01$	$\rho = 0.05$	$\rho = 0.1$		
CiteSeer	AUC	0.805	0.805	0.924	0.692	0.713	0.694	0.635	0.8618	**0.945**
	AP	0.850	0.836	0.930	0.702	0.723	0.670	0.636	0.867	**0.952**
Cora	AUC	0.846	0.831	0.924	0.876	0.892	0.846	0.815	0.902	**0.931**
	AP	0.885	0.850	0.926	0.873	0.896	0.849	0.817	0.911	**0.943**
PubMed	AUC	0.842	0.844	**0.968**	0.954	0.949	0.900	0.879	0.962	0.957
	AP	0.878	0.841	**0.971**	0.954	0.950	0.905	0.882	0.963	0.958

5 Experiment 2: Link Prediction

Besides predicting the properties of each node, another important application of graph is to predict whether a pair of nodes should be connected by an edge. Applications include social network, recommender system, and knowledge graph.

Followed by the work from Kipf and Welling (2016), assuming we have a graph $G = (\mathcal{V}, \mathcal{E})$, our goal can be written
as reconstructing the adjacency matrix $\mathbf{P}$ by using GCN's output $\mathbf{Z}$:

$$\mathbf{P} = \sigma\left(\mathbf{Z}\mathbf{Z}^\top\right), \quad (\sigma \text{ applied element-wise}) \tag{7}$$

$$\mathbf{Z} = \text{GCN}(\mathbf{A}, \mathbf{X}) = \tilde{\mathbf{A}}\sigma\left(\tilde{\mathbf{A}}\mathbf{X}\mathbf{W}^{(1)}\right)\mathbf{W}^{(2)}, \tag{8}$$

where σ can be the sigmoid function to represent the probability of connection, or simply a sign function. Intuitively, since each row of $\mathbf{Z}$ corresponds to the embedding of a node, (7) uses their pairwise inner product to determine whether they are to be connected.

We partition the set of edges $\mathcal{E}$ into $\mathcal{E}_{train}$, $\mathcal{E}^+_{val}$, $\mathcal{E}^-_{val}$, $\mathcal{E}^+_{test}$, $\mathcal{E}^-_{test}$, where $+$ and $-$ represent the existent and non-existent edges. In order to ensure proper node embedding, we enforce that $\mathcal{E}_{train}$ covers all the nodes in the link prediction task.

Similar to node prediction, adversarial training can be achieved by incorporating a regularizer which penalizes the change of the prediction under the perturbation, i.e.,

$$f(\mathbf{A}, \mathbf{X}) = \mathcal{L}_{\boldsymbol{\theta}}(\mathbf{P}, \mathbf{A}) + \lambda \max_{\boldsymbol{\zeta} \in D} \left\|\sigma\left(\mathbf{Z}\mathbf{Z}^\top\right) - \sigma\left(\mathbf{Z}_\zeta \mathbf{Z}_\zeta^\top\right)\right\|_F^2,$$

where $\mathbf{Z}_\zeta = \tilde{\mathbf{A}}\left(\sigma\left(\tilde{\mathbf{A}}\mathbf{X}\mathbf{W}^{(1)}\right) + \boldsymbol{\zeta}\right)\mathbf{W}^{(2)}$ and $\|\boldsymbol{\zeta}_{i:}\| \leq \epsilon$ for all $i \in |\mathcal{V}|$. It is natural to base the regularizer on the variation of $\mathbf{Z}\mathbf{Z}^\top$ because ultimately the adjacency matrix

Table 6: Success rate for Nettack-link and random attack under a budget of Δ edges for structure attack

Model	Attacker	CiteSeer				Cora				PubMed			
	Δ	1	5	10	20	1	5	10	20	1	5	10	20
GCN	Nettack-link	0.12	0.33	0.54	0.86	0.09	0.27	0.45	0.83	0.10	0.24	0.41	0.73
LAT-GCN-A	Nettack-link	0.15	0.25	0.43	0.69	0.08	0.21	0.38	0.65	0.08	0.21	0.34	0.67
GCN	random	0.05	0.13	0.17	0.23	0.01	0.08	0.17	0.31	0.00	0.03	0.11	0.18
LAT-GCN-A	random	0.06	0.14	0.15	0.20	0.02	0.09	0.17	0.29	0.00	0.05	0.09	0.20

Algorithm 2 Evaluation of robustness

```
input $\mathbf{A}, \mathbf{X}, \Delta$ (budget of Nettack)
1: $s_{\text{GCN}} = s_{\text{LAT-GCN}} = 0$
2: $\mathcal{T} :=$ sample 100 nodes from test set to attack
3: for $u \in \mathcal{T}$ (target of attack) & $g \in \{\text{GCN, LAT-GCN}\}$ do
4:     $c_g := evaluate(u, g(\mathbf{A}, \mathbf{X}))$
5:     if $c_g = c_{true}(u)$ then
6:         $\mathbf{A}', \mathbf{X}' \leftarrow Nettack(\mathbf{A}, \mathbf{X}, u, \Delta)$
7:         $s_g := s_g + \delta(c_g \neq c_{true}(u))$
8:     end if
9: end for
output success rates: $s_{\text{GCN}} / |\mathcal{T}|$, $s_{\text{LAT-GCN}} / |\mathcal{T}|$.
```

$\mathbf{P}$ is based on $\mathbf{ZZ}^\top$. In practice, however, we noticed that the performance does not change noticeably if we directly regularize the variation of $\mathbf{ZZ}^\top$:

$$f(\mathbf{A}, \mathbf{X}) = \mathcal{L}_{\boldsymbol{\theta}}(\mathbf{P}, \mathbf{A}) + \lambda \max_{\boldsymbol{\zeta} \in D} \left\| \mathbf{ZZ}^\top - \mathbf{Z}_\zeta \mathbf{Z}_\zeta^\top \right\|_F^2. \tag{9}$$

Here $\mathbf{Z}_\zeta$ simplifies to $\tilde{\mathbf{A}}\mathbf{X}\mathbf{W}^{(1)} + \boldsymbol{\zeta}$ when a one-layer GCN is used. Similar to (4), the inner maximization over $\boldsymbol{\zeta}$ can be solved by ADAM as in Section 3.1. We will again refer to the adversarial objective (9) as LAT-GCN. Although GCNs used in link prediction are often referred to as graph auto-encoders (GAE), we will stick with the term "GCN" to simplify terminology. Indeed, many of the discussions and approaches based on GCNs in this paper are applicable to more general graph neural networks.

Comparison of accuracy. We compared one-layer LAT-GCN against vanilla GCN and three more baselines. Spectral clustering (Tang and Liu, 2011) is an effective approach to learn node embedding. DeepWalk (Perozzi *et al.*, 2014) is also a popular approach to represent nodes into a continuous vector space. Adversarial regularized auto-encoder (Pan *et al.*, 2019) generates adversarial examples from node embedding. Table 5 shows that with a slight perturbation on the latent layer in GCN, the LAT-GCN outperforms GCN in area under ROC curve (AUC) and average precision (AP) on CiteSeer and Cora datasets. Here the results are reported based on the optimal tuned hyperparameter, with $\epsilon = 0.1$ and $\gamma = 0.02$ for both CiteSeer and Cora. Moreover, Figure 7 shows how the varied value of ϵ impacts the performance of LAT-GCN (γ fixed to 0.1). Clearly only small values of ϵ can be helpful.

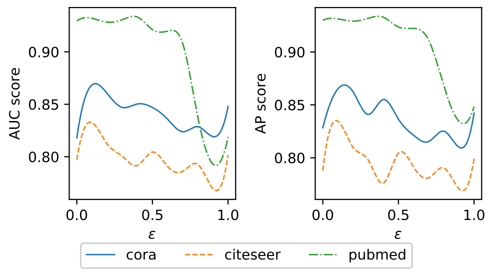

Fig. 7: AUC and AP curves of LAT-GCN with varied values of ϵ

Although LAT-GCN does not perform the best on PubMed, our further experiment suggests that robustness is probably not positively correlated with generalization performance on this dataset. In particular, analogous to standard data augmentation, we randomly added or removed $\rho\,|\mathcal{V}|^2$ number of edges in $\mathcal{E}_{train}$, where the budget $\rho \in \{0.01, 0.05, 0.1\}$. So the resulting unnormalized adjacency matrix $\mathbf{A}'$ deviates from $\mathbf{A}$ in L_1 norm by at most $\rho\,|\mathcal{V}|^2$. We call it random attack (RA). Interestingly, as shown in the four columns underneath GCN-1 (one-layer GCN), increasing ρ from 0 to 0.01 does improve the test AUC and AP for CiteSeer and Cora, but further increasing ρ starts to be harmful. In contrast, the performance on PubMed decays monotonically as ρ grows, suggesting that enforcing robustness to adversarial attacks on this dataset may be conflicting with generalization.

Comparison of robustness. Since no specialized algorithm is available that adversarially attacks link prediction under GCNs, we designed a novel extension of Nettack, called Nettack-link, for this setting in addition to random attacks. Given a targeted link (u, v), denote $\delta_{u,v} = 1$ if (u, v) is in $\mathcal{E}$, and -1 otherwise. Nettack-link considers the change of $\mathbf{P}_{u,v}$ under perturbations of node features and graph structure, i.e.,

$$s_{struct}(e; \mathbf{A}, (u, v)) := \delta_{u,v} \cdot \left(\log \mathbf{P}_{u,v} - \log \mathbf{P}'_{u,v}\right),$$

for perturbing another edge e. Here we can restrict the candidate edge e to one-hop or two-hop away from either u or v. By greedily picking an edge e whose addition/removal maximizes $s_{struct}(e; \mathbf{A}, (u, v))$, the algorithm can progressively find Δ number of edges to attack the target link prediction (u, v). To facilitate an efficient computation which avoids computing $s_{struct}(e; \mathbf{A}, (u, v))$ from scratch every time, we followed the linearization recipe from Zügner *et al.* (2018), and identified an incremental procedure.

Table 6 shows the success rate of the two attacks, where both the set of existing and non-existing edges get a budget of Δ edges to remove or add, respectively. The test protocol followed Algorithm 2, except that we randomly picked 100 (existent or nonexistent) edges to attack in step 2. Clearly Nettack-list is much more effective than random attack. Both GCN and LAT-GCN suffer an increased rate of being successfully attacked as the budget grows, but LAT-GCN is more robust than GCN with a lower success rate in general.

6 Experiment 3: Recommendation

Finally we studied another effective application of GCNs: recommendation by matrix completion (GC-MC, Berg *et al.*, 2018). Compared with the general structured graph above, here the graphs are bipartite with two groups of nodes such as users and items, and links only span between nodes of different groups. To model the R levels of ratings (or other discrete recommendation levels), a set of adjacency matrices $\{\mathbf{A}_1, \cdots, \mathbf{A}_R\}$ are constructed for each rating type $r \in R$, where $\mathbf{A}_r$ is sized N_u-by-N_v for N_u users and N_v items, respectively. The (u, v)-th entry of $\mathbf{A}_r$ is 1 if and only if the user u gives a rating of r to the item v, and 0 else. Normalization of $\mathbf{A}_r$ follows as above.

After GCN propagation is completed on each graph separately, the representation of an item or user is formed by aggregating its corresponding embeddings in the R graphs, using concatenation or summation. The final rating for a (user, item) pair can then be predicted based on the inner product between the embeddings of the user and item, akin to link prediction.

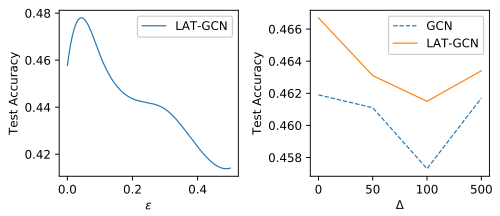

Fig. 8: Accuracy for MovieLen 100k with respect to ϵ in LAT-GCN (left), and the attack budget Δ (right)

Since there is no customized attacking algorithm for this setting, we resorted to random attack where a rating is randomly selected and flipped to another level; for simplicity, we did not consider just removing or adding ratings. The fact that random attack is not targeted allows us to present robustness and generalization performance in one plot. We followed the same setting as the GC-MC model, including the learning rate, dimensions of hidden layers, etc. Summation was applied to aggregate embeddings from five graphs, and we tested on the Movielens 100k dataset[1].

Figure 8 (left) shows that the test accuracy increases when LAT-GCN imposes a small perturbation on the GCN layer in training, but eventually the test accuracy decays. This is similar to Figure 7. Furthermore, we varied the budget Δ for randomly flipping the ratings in the training data. As Figure 8 (right) shows, the test accuracy of LAT-GCN (with ϵ fixed to 0.1) keeps higher than that of vanilla GCN for a range of Δ.

7 Conclusion

In this work, we proposed a new regularization technique for GCN which not only improves generalization, but also defends against attacks in both node feature and graph structure. Superior empirical performance is achieved on node classification and link

[1] `https://grouplens.org/datasets/movielens/`

prediction problems. The method, which is based on perturbing latent representations, can be extended to adversarial training of other graph learning problems, such as recommender systems, and dynamic and multi-modal graphs.

Acknowledgements The Titan V used for this research was donated by the NVIDIA Corporation. We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan V GPU used for this research.

References

R. v. d. Berg, T. N. Kipf, and M. Welling. Graph convolutional matrix completion. In *KDD*, 2018.

D. P. Bertsekas. *Nonlinear Programming*. Athena Scientific, Belmont, MA, 1995.

J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun. Spectral networks and locally connected networks on graphs. In *ICLR*, 2013.

J. Chen, J. Zhu, and L. Song. Stochastic training of graph convolutional networks with variance reduction. In *ICML*, 2017.

J. Chen, T. Ma, and C. Xiao. Fastgcn: fast learning with graph convolutional networks via importance sampling. In *ICLR*, 2018.

J. Chen, Z. Shi, Y. Wu, X. Xu, and H. Zheng. Link prediction adversarial attack. *arXiv preprint arXiv:1810.01110*, 2018.

D. Cullina, A. N. Bhagoji, and P. Mittal. PAC-learning in the presence of adversaries. In *NeurIPS*, 2018.

H. Dai, H. Li, T. Tian, X. Huang, L. Wang, J. Zhu, and L. Song. Adversarial attack on graph structured data. In *ICML*, pages 1123–1132, 2018.

M. Defferrard, X. Bresson, and P. Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *NeurIPS*, pages 3844–3852, 2016.

Z. Deng, Y. Dong, and J. Zhu. Batch virtual adversarial training for graph convolutional networks. *arXiv preprint arXiv:1902.09192*, 2019.

D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams. Convolutional networks on graphs for learning molecular fingerprints. In *NeurIPS*, pages 2224–2232, 2015.

F. Feng, X. He, J. Tang, and T.-S. Chua. Graph adversarial training: Dynamically regularizing based on graph structure. *arXiv preprint arXiv:1902.08226*, 2019.

D. K. Hammond, P. Vandergheynst, and R. Gribonval. Wavelets on graphs via spectral graph theory. *Applied and Computational Harmonic Analysis*, 30(2):129–150, 2011.

W. He, J. Wei, X. Chen, N. Carlini, and D. Song. Adversarial example defenses: Ensembles of weak defenses are not strong. *arXiv preprint arXiv:1706.04701*, 2017.

S. Huang, N. Papernot, I. Goodfellow, Y. Duan, and P. Abbeel. Adversarial attacks on neural network policies. *arXiv preprint arXiv:1702.02284*, 2017.

R. Jia and P. Liang. Adversarial examples for evaluating reading comprehension systems. In *ICLR*, 2017.

D. P. Kingma and J. L. Ba. Adam: A method for stochastic optimization. *ICLR*, 2015.

T. N. Kipf and M. Welling. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016.

T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.

A. Kurakin, I. Goodfellow, and S. Bengio. Adversarial machine learning at scale. In *ICLR*, 2017.

T. Miyato, A. M. Dai, and I. Goodfellow. Adversarial training methods for semi-supervised text classification. In *ICLR*, 2017.

S. Pan, R. Hu, S.-f. Fung, G. Long, J. Jiang, and C. Zhang. Learning graph embedding with adversarial training methods. *arXiv preprint arXiv:1901.01250*, 2019.

B. Perozzi, R. Al-Rfou, and S. Skiena. Deepwalk: Online learning of social representations. In *KDD*, pages 701–710. ACM, 2014.

A. Raghunathan, J. Steinhardt, and P. Liang. Certified defenses against adversarial examples. In *ICLR*, 2018.

M. Schlichtkrull, T. N. Kipf, P. Bloem, R. Van Den Berg, I. Titov, and M. Welling. Modeling relational data with graph convolutional networks. In *ESWC*, pages 593–607. Springer, 2018.

L. Schmidt, S. Santurkar, D. Tsipras, K. Talwar, and A. Madry. Adversarially robust generalization requires more data. In *NeurIPS*, 2018.

P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008.

S. Shafieezadeh-Abadeh, D. Kuhn, and P. M. Esfahani. Regularization via mass transportation. *arXiv preprint arXiv:1710.10016*, 2017.

A. Sinha, H. Namkoong, and J. Duchi. Certifying some distributional robustness with principled adversarial training. In *ICLR*, 2018.

Y. Song, T. Kim, S. Nowozin, S. Ermon, and N. Kushman. Pixeldefend: Leveraging generative models to understand and defend against adversarial examples. In *ICLR*, 2018.

D. Stutz, M. Hein, and B. Schiele. Disentangling adversarial robustness and generalization. *CVPR*, 2019.

C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. In *ICLR*, 2014.

L. Tang and H. Liu. Leveraging social media networks for classification. *Data Mining and Knowledge Discovery*, 23(3):447–478, 2011.

P. V. Tran. Learning to make predictions on graphs with autoencoders. In *5th IEEE International Conference on Data Science and Advanced Analytics*, 2018.

D. Tsipras, S. Santurkar, L. Engstrom, A. Turner, , and A. Madry. Robustness may be at odds with accuracy. In *ICLR*, 2019.

P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio. Graph attention networks. In *ICLR*, 2018.

S. Wang, Z. Chen, J. Ni, X. Yu, Z. Li, H. Chen, and P. S. Yu. Adversarial defense framework for graph neural network. *arXiv preprint arXiv:1905.03679*, 2019.

E. Wong and J. Z. Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *ICML*, 2018.

F. Wu, T. Zhang, A. H. d. Souza Jr, C. Fifty, T. Yu, and K. Q. Weinberger. Simplifying graph convolutional networks. *arXiv preprint arXiv:1902.07153*, 2019.

K. Xu, H. Chen, S. Liu, P.-Y. Chen, T.-W. Weng, M. Hong, and X. Lin. Topology attack and defense for graph neural networks: an optimization perspective. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, pages 3961–3967. AAAI Press, 2019.

D. Yin, K. Ramchandran, and P. Bartlett. Rademacher complexity for adversarially robust generalization. *arXiv preprint arXiv:1810.11914*, 2018.

H. Zhang, Y. Yu, J. Jiao, E. P. Xing, L. E. Ghaoui, and M. I. Jordan. Theoretically principled trade-off between robustness and accuracy. In *ICML*, 2019.

D. Zügner and S. Günnemann. Certifiable robustness and robust training for graph convolutional networks. In *KDD*, pages 246–256. ACM, 2019.

D. Zügner, A. Akbarnejad, and S. Günnemann. Adversarial attacks on neural networks for graph data. In *KDD*, pages 2847–2856. ACM, 2018.