Graph Adversarial Training: Dynamically Regularizing Based on Graph Structure

Fuli Feng, Xiangnan He, Jie Tang, Tat-Seng Chua

Abstract—Recent efforts show that neural networks are vulnerable to small but intentional perturbations on input features in visual classification tasks. Due to the additional consideration of connections between examples (*e.g.,* articles with citation link tend to be in the same class), graph neural networks could be more sensitive to the perturbations, since the perturbations from connected examples exacerbate the impact on a target example. *Adversarial Training* (AT), a dynamic regularization technique, can resist the worst-case perturbations on input features and is a promising choice to improve model robustness and generalization. However, existing AT methods focus on standard classification, being less effective when training models on graph since it does not model the impact from connected examples.

In this work, we explore adversarial training on graph, aiming to improve the robustness and generalization of models learned on graph. We propose *Graph Adversarial Training* (GraphAT), which takes the impact from connected examples into account when learning to construct and resist perturbations. We give a general formulation of GraphAT, which can be seen as a dynamic regularization scheme based on the graph structure. To demonstrate the utility of GraphAT, we employ it on a state-of-the-art graph neural network model — *Graph Convolutional Network* (GCN). We conduct experiments on two citation graphs (Citeseer and Cora) and a knowledge graph (NELL), verifying the effectiveness of GraphAT which outperforms normal training on GCN by 4.51% in node classification accuracy. Codes are available via: https://github.com/fulifeng/GraphAT.

Index Terms—Adversarial Training, Graph-based Learning, Graph Neural Networks

1 Introduction

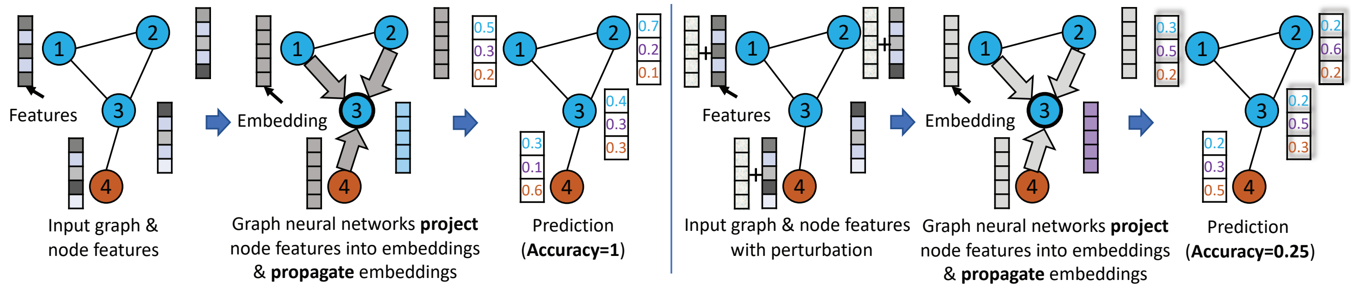
Graph-based learning makes predictions by accounting for both input features of examples and the relations between examples. It is remarkably effective for a wide range of applications, such as predicting the profiles and interests of social network users [1], [2], predicting the role of a protein in biological interaction graph [3], [4], and classifying contents like documents, videos, and webpages based on their interlinks [5]–[7]. In addition to the *supervised loss* on labeled examples, graph-based learning also optimizes the *smoothness* of predictions over the graph structure, that is, closely connected examples are encouraged to have similar predictions [8]–[11]. Recently, owing to the extraordinary representation ability, deep neural networks become prevalent models for graph-based learning [1], [7], [10]–[12].

Despite promising performance, we argue that graph neural networks are vulnerable to small but intentional perturbations on the input features [13], [14], and this could even be more serious than the standard neural networks that do not model the graph structure. The reasons are twofold: 1) graph neural networks also optimize the supervised loss on labeled data, thus it will face the same vulnerability issue as the standard neural networks [15], and 2) the additional smoothness constraint will exacerbate the impact of perturbations, since smoothing across connected nodes[1] would aggregate the impact of perturbations from nodes connected to the target node (i.e., the node that we apply perturbations with the aim of changing its prediction). Figure 1 illustrates the impact of perturbations on node features with an intuitive example of a graph with 4 nodes. A graph neural network model predicts node labels (3 in total) for clean input features and features with applied perturbations, respectively. Here perturbations are intentionally applied to the features of nodes 1, 2, 4. Consequently, the graph neural network model is fooled to make wrong predictions on nodes 1 and 2 as with standard neural networks. Moreover, by propagating the node embeddings, the model aggregates the influence of perturbations to node 3, from which its prediction is also affected. In real-world applications, small perturbations like the update of node features may frequently happen, but should not change the predictions much. As such, we believe that there is a strong need to stabilize the graph neural network models during training.

Adversarial Training (AT) is a dynamic regularization technique that proactively simulates the perturbations during the training phase [15]. It has been empirically shown to be able to stabilize neural networks, and enhance their robustness against perturbations in standard classification tasks [16]–[21]. Therefore, employing a similar approach to that of AT on a graph neural network model would also be helpful to the model's robustness. However, directly employing AT on graph neural network is insufficient, since

- *F. Feng and TS. Chua are with School of Computing, National University of Singapore, Computing 1, Computing Drive, 117417, Singapore. E-mail: fulifeng93@gmail.com, dcscts@nus.edu.sg.*
- *X. He (corresponding author) is with School of Information Science and Technology, University of Science and Technology of China, Hefei, China. E-mail: xiangnanhe@gmail.com.*
- *J. Tang is with Tsinghua University, Beijing 100084, China. E-mail: jietang@tsinghua.edu.cn.*

[1] In the following sections, we interchangeably use node and example.



(a) Standard training

(b) Adversarial training applying perturbations

Fig. 1: An intuitive example to illustrate the impact of applying perturbations to the input node features on the prediction of graph neural networks. Here the model implements the graph smoothness constraint via propagating node embeddings over the graph. On the right, the model propagates the applied perturbations on the connected nodes of the target node 3, leading to a wrong prediction. Moreover, the perturbations on node 1 and 2 directly lead to the wrong associated predictions like in the standard neural networks. (Better viewed in color.)

it treats examples as independent of each other and does not consider the impacts from connected examples. As such, we propose a new adversarial training method, named *Graph Adversarial Training* (GraphAT), which learns to construct and resist perturbations by taking the graph structure into account.

The key idea of GraphAT is that, when generating perturbations on a target example, it maximizes the divergence between the prediction of the target example and its connected examples. That is, the adversarial perturbations should attack the graph smoothness constraint as much as possible. Then, GraphAT updates model parameters by additionally minimizing a *graph adversarial regularizer*, reducing the prediction divergence between the perturbed target example and its connected examples. Through this way, GraphAT can resist the worst-case perturbations on graph-based learning and enhance model robustness. To efficiently calculate the adversarial perturbations, we further devise a linear approximation method based on back-propagation.

To demonstrate GraphAT, we employ it on a well-established graph neural network model, *Graph Convolutional Network* (GCN) [7], which implements the smoothness constraint by performing embedding propagation. We study the method's performance on node classification, one of the most popular tasks on graph-based learning. Extensive experiments on three public benchmarks (two citation graphs and a knowledge graph) verify the strengths of GraphAT — compared to normal training on GCN, GraphAT leads to 4.51% accuracy improvement. Moreover, the improvements on less popular nodes (with a small degree) are more significant, highlighting the necessity of performing AT with the graph structure considered.

The main contributions of this paper are summarized as:

- We formulate *Graph Adversarial Training*, a new optimization method for graph neural networks that can enhance the model's robustness against perturbations on node input features.
- We devise a *graph adversarial regularizer* that dynamically encourages the model to generate similar predictions on the perturbed target example and its connected examples, and develop an efficient algorithm to construct perturbations.
- We demonstrate the effectiveness of GraphAT on GCN, conducting experiments on three datasets which show that our method achieves state-of-the-art performance for node classification. Codes are available via: https://github.com/fulifeng/GraphAT.

In the remainder of this paper, we first discuss related work in Section 2, followed by the problem formulation and preliminaries in Section 3. In Section 4 and 5, we elaborate the method and experimental results, respectively. We conclude the paper and envision future directions in Section 6.

2 Related Work

In this section, we discuss the existing research on graph-based learning and adversarial learning, which are closely related to this work.

2.1 Graph-based Learning

Graph, a natural representation of relational data, in which nodes and edges represent entities and their relations, is widely used in the analysis of social networks, transaction records, biological interactions, collections of interlinked documents, web pages, and multimedia contents, *etc.*. On such graphs, one of the most popular tasks is *node classification* targeting to predicting the label of nodes in the graph by accounting for node features and the graph structure. The existing work on node classification mainly fall into two broad categories: *graph Laplacian regularization* and *graph embedding-based methods*. Methods lying in the former category explicitly encode the graph structure as a regularization term to smooth the predictions over the graph, *i.e.*, the regularization incurs a large penalty when similar nodes (*e.g.*, closely connected) are predicted with different labels [8], [9], [22]–[24].

Recently, graph embedding-based methods, which learn node embeddings that encodes the graph data, have become promising solution. Most of embedding-based methods fall into two broad categories: *skip-gram based methods* and *convolution based methods*, depending on how the graph data are modeled. The skip-gram based methods learn node embeddings via using the embedding of a node to predict node context that are generated by performing random walk on the graph so as the embeddings of ”connected” nodes are associated to each other [2], [5], [6], [12]. Inspired by the idea of convolution in computer vision, which aggregates contextual signals in a local window, convolution based methods iteratively aggregate representation of neighbor nodes to learn a node embedding [3], [4], [7], [11], [25]–[30].

In both of the two categories, methods leveraging the advanced representation ability of deep neural networks (*neural graph-based learning methods*) have shown remarkably effective in solving the node classification task. However, the neural graph-based learning models are vulnerable to intentionally designed perturbations indicating the unstability in generalization [13], [31], and little attention has been paid to enhance the *robustness* of these methods, which is the focus of this work.

2.2 Adversarial Learning

2.2.1 Adversarial Training

In order tackle the vulnerability to intentional perturbations of deep neural networks, researchers proposed adversarial training which is an alternative minimax process [32]. The adversarial training methods augment the training process by dynamically generating adversarial examples from clean examples with perturbations maximally attacking the training objective, and then learn over these adversarial examples by minimizing an additional regularization term [15], [17], [33]–[38]. The adversarial training methods mainly fall into *supervised* and *semi-supervised* ones regarding the target of the training objective. In supervised learning tasks such as visual recognition [15], supervised loss [15], [33], [34] and its surrogates [36]–[38] over adversarial examples are designed as the target of the maximization and minimization. For semi-supervised learning where partial examples are labeled, divergence of predictions for inputs around each examples is adopted as the target. Generally speaking, the philosophy of adversarial training methods is to smooth the prediction around individual inputs in a dynamical fashion.

Our work is inspired by these adversarial training methods. In addition to the local smoothness of individual examples, our method further accounts for relation between examples (*i.e.,* the graph structure) in the target of the minimax process so as to learn robust classifiers predicting smoothly over the graph structure. To the best of our knowledge, this is the first attempt to incorporate graph structure in adversarial training.

Another emerging research topic related to our work is generating adversarial perturbations attacking neural graph-based learning models where [31] and [13] are the only published work. However, methods in [31] and [13] are not suitable for constructing adversarial examples in graph adversarial training. This is because these methods generate a new graph as the adversarial example for each individual node, *i.e.,* they would generate N graphs when the number of nodes is N leading to unaffordable memory overhead. In this work, we devise an efficient method to generate adversarial examples for graph adversarial training.

2.2.2 Generative Adversarial Networks

Generative adversarial networks (GAN) is a machine learning framework with two different networks as a generator and a discriminator playing minimax game on generating and detecting fake examples. Recently, several GAN-based models are proposed to learn graph embeddings, which either generate fake nodes and edges to augment embedding learning [39], [40] or smooth the leaned embeddings to follow a prior distribution [41]–[44]. However, using two different networks inevitably doubles the computation of model training and the labor of parameter tuning of GAN-based methods. Moreover, for different applications, one may need to build GAN from scratch, whereas our method is a generic solution can be seamlessly applied to enhance the existing graph neural network models with less computing and tuning overhead.

TABLE 1: Terms and notations.

Symbol	Definition
$\boldsymbol{X} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_N]^T \in \mathbb{R}^{N\times F}$	features of N nodes.
$\boldsymbol{A}, \boldsymbol{D} \in \mathbb{R}^{N\times N}$	adjacency matrix and degree matrix of a graph.
$\boldsymbol{Y} = [\boldsymbol{y}_1, \boldsymbol{y}_2, \cdots, \boldsymbol{y}_M]^T \in \mathbb{R}^{M\times L}$	labels of M nodes.
$\boldsymbol{r}_i^g$	graph adversarial perturbation of node i.
$\boldsymbol{r}_i^v$	virtual graph adversarial perturbation of node i.
$\boldsymbol{W}^l, \boldsymbol{b}^l$	weights and bias to be learned at layer l.
$\hat{\boldsymbol{y}}_{\boldsymbol{i}} = f(\boldsymbol{x}_i, G\vert\boldsymbol{\Theta})$	prediction function.
$\boldsymbol{\Theta}$	model parameters of the prediction function f.

3 Preliminaries

We first introduce some notations used in the following sections. We use bold capital letters (e.g. $\boldsymbol{X}$) and bold lowercase letters (e.g. $\boldsymbol{x}$) to denote matrices and vectors, respectively. Note that all vectors are in a column form if not otherwise specified, and X_{ij} denotes the entry of matrix $\boldsymbol{X}$ at the row i and column j. In Table 1, we summarize some of the terms and notations.

3.1 Graph Representation

The nodes and edges of a graph represent the entities of interest and their relations, respectively. First, the edges in a graph with N nodes are typically represented as an adjacency matrix $\boldsymbol{A} \in \mathbb{R}^{N\times N}$. In this work, we mainly study unweighted graphs where $\boldsymbol{A}$ is a binary matrix. $A_{ij} = 1$ if there is an edge between node i and j, otherwise $A_{ij} = 0$. Moreover, we use a diagonal matrix $\boldsymbol{D} \in \mathbb{R}^{N\times N}$ to denote the degrees of nodes, *i.e.,* $D_{ii} = \sum_{j=1}^{N} A_{ij}$. For an attributed graph, where each node is associated with a feature vector, we use a matrix $\boldsymbol{X} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_N]^T \in \mathbb{R}^{N\times F}$ to represent the feature vectors of all nodes, where F is the dimension of the features. Finally, an attributed graph is denoted as $G = (\boldsymbol{A},\ \boldsymbol{D},\ \boldsymbol{X})$.

3.2 Node Classification

On graph data, node classification is one of the most popular tasks. In the general problem setting of node classification, a graph G with N nodes is given, associated with labels ($\boldsymbol{Y}$) of a some portion of nodes [7], [11], [12]. This setting is transductive since testing nodes are observed (only features and associated edges) during training, and is the focus of this work. Here, $\boldsymbol{Y} = [\boldsymbol{y}_1, \boldsymbol{y}_2, \cdots, \boldsymbol{y}_M]^T \in \mathbb{R}^{M\times L}$ are the labels, where M and L are the numbers of labeled nodes and node classes, respectively, and $\boldsymbol{y}_i$ is the one-hot encoding of node i's label. Note that, without loss of generality, we index the labeled nodes and unlabeled nodes in the range of $[1, M]$ and $(M, N]$, respectively. The target of node classification is to learn a prediction function (classifier) $\hat{\boldsymbol{y}}_i = f(\boldsymbol{x}_i, G|\boldsymbol{\Theta})$, to forecast the label of the node, where $\boldsymbol{\Theta}$ includes model parameters to be learned.

3.3 Graph-based Learning

Graph-based learning methods have been shown remarkably effective on solving the node classification task [8], [9], [22], [23]. Generally, most of the graph-based learning models jointly optimize two objectives: 1) *supervised loss* on labeled nodes and 2) *graph smoothness constraint*, which can be summarized as:

$$\Gamma = \Omega + \lambda \Phi, \tag{1}$$

where Ω is a classification loss (*e.g.*, log loss, hinge loss, and cross-entropy loss) that measures the discrepancy between prediction and ground-truth of labeled nodes. Φ encourages *smoothness* of predictions over the graph structure, which is based on the assumption that closely connected nodes tend to have similar predictions. For instance, Φ could be a *graph Laplacian term*, $\sum_{i,j=1}^{N} A_{ij} \|\hat{\boldsymbol{y}}_i - \hat{\boldsymbol{y}}_j\|^2$, which directly regulates the predictions of connected nodes to be similar [8], [9], [22], [23]. The assumption could also be implicitly implemented by iteratively propagating *node embeddings* through the graph so that connected nodes obtain close embeddings and are predicted similarly [3], [7], [10], [11]. Here, λ is a hyperparameter to balance the two terms.

4 Methodology

In this section, we first introduce the formulation of *Graph Adversarial Training*, followed by the introduction of *GraphVAT*, an extension of GraphAT, which further incorporates the virtual adversarial regularization [35]. We then present two solutions for the node classification task, *GraphAT* and *GraphVAT*, which employ GraphAT and GraphVAT to train GCN [7], respectively. Finally, we analyze the time complexity of the two solutions and present the important implementation details.

4.1 Graph Adversarial Training

Recent advances of *Adversarial Training* has been successful in learning deep neural network-based classifiers, making them robust against perturbations for a wide range of standard classification tasks such as visual recognition [15], [16], [35] and text classification [17]. Generally, applying AT would regulate the model parameters to smooth the output distribution [35]. Specifically, for each clean example in the dataset, AT encourages the model to assign similar outputs to the artificial input (*i.e.*, the *adversarial example*) derived from the clean example. Inspired by the philosophy of standard AT, we develop graph adversarial training, which trains graph neural network modules in the manner of generating adversarial examples and optimizing additional regularization terms over the adversarial examples, so as to prevent the adverse effects of perturbations. Here the focus is to prevent perturbations propagated through node connections (as illustrated in Figure 1), *i.e.*, accounting for graph structure in adversarial training.

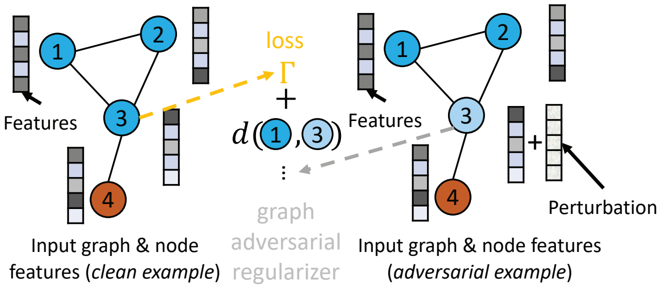


Fig. 2: The training process of GraphAT: 1) constructing graph adversarial example and 2) updating model parameters by minimizing loss and graph adversarial regularizer.

Generally, the formulation of GraphAT is:

$$\begin{aligned}
&\textbf{min: } \Gamma_{GAT} = \Gamma + \beta \sum_{i=1}^{N} \sum_{j \in \mathcal{N}_i} d(f(\boldsymbol{x}_i + \boldsymbol{r}_i^g, G|\boldsymbol{\Theta}), f(\boldsymbol{x}_j, G|\boldsymbol{\Theta})),\\
&\textbf{max: } \boldsymbol{r}_i^g = \arg \max_{\boldsymbol{r}_i, \|\boldsymbol{r}_i\| \le \epsilon} \sum_{j \in \mathcal{N}_i} d(f(\boldsymbol{x}_i + \boldsymbol{r}_i, G|\hat{\boldsymbol{\Theta}}), f(\boldsymbol{x}_j, G|\hat{\boldsymbol{\Theta}})),
\end{aligned} \tag{2}$$

where Γ_{GAT} is the training objective function with two terms: the standard objective function of the origin graph-based learning model (*e.g.*, Equation 1) and *graph adversarial regularizer*. The second term encourages the graph adversarial examples to be classified similarly as connected examples where $\boldsymbol{\Theta}$ denotes the parameters to be learned, and d is a nonnegative function that measures the divergence (*e.g.*, Kullback-Leibler divergence [45]) between two predictions. $\boldsymbol{r}_i^g$ denotes the graph adversarial perturbation, which is applied to the input feature of the clean example i to construct a graph adversarial example.

The graph adversarial perturbation is calculated by maximizing the graph adversarial regularizer under current value of model parameters. That is to say, the graph adversarial perturbation is the direction of changes on the input feature, which can maximally attack the graph adversarial regularizer, *i.e.*, the worst case of perturbations propagated from neighbor nodes. ϵ is a hyperparameter controling the magnitude of perturbations, which is typically set as small values so that the feature distribution of adversarial examples is close to that of clean examples.

Generally, similar to the standard AT, each iteration of GraphAT can also be viewed as playing a *minimax game*:

- **Maximization**: GraphAT generates graph adversarial perturbations from clean examples, which break the smoothness between connected nodes to the maximum extent. and then constructs graph adversarial examples by adding the perturbations to the input of associated clean examples.
- **Minimization**: GraphAT minimizes the objective function of the graph neural network with an additional regularizer over graph adversarial examples, by encouraging smoothness between predictions of adversarial examples and connected examples. As such, the model becomes robust against perturbations propagated through the graph.

Figure 2 illustrates the process of GraphAT. While the traditional graph-based regularizations (*e.g.*, the graph Laplacian term) also encourage the smoothness of predictions over the graph structure, GraphAT is believed to be a more advanced regulation for two reasons: 1) the regularization performed by GraphAT is dynamic since the adversarial examples are adaptively generated according to the current parameters and predictions of the model whereas the standard graph-based regularizations are static; and 2) GraphAT to some extent augments the training data, since the generated adversarial examples have not occurred in the training data, which is beneficial to model generalization.

Approximation. It is non-trivial to obtain the closed-form solution of $\boldsymbol{r}_i^g$. Inspired by the *linear approximation* method proposed in [15] for standard adversarial training, we also design a linear approximation method to calculate the graph adversarial perturbations in GraphAT, of which the formulation is:

$$\boldsymbol{r}_i^g \approx \epsilon \frac{\boldsymbol{g}}{\|\boldsymbol{g}\|}, \text{ where } \boldsymbol{g} = \nabla_{\boldsymbol{x}_i} \sum_{j \in \mathcal{N}_i} D(f(\boldsymbol{x}_i, G|\hat{\boldsymbol{\Theta}}), f(\boldsymbol{x}_j, G|\hat{\boldsymbol{\Theta}})), \tag{3}$$

where $\boldsymbol{g}$ is the gradient *w.r.t.* the input $\boldsymbol{x}_i$. For graph neural network models, the gradient can be efficiently calculated by one backpropagation. Note that $\hat{\boldsymbol{\Theta}}$ is a constant set denoting the current model parameters.

4.2 Virtual Graph Adversarial Training

Considering that node classification is a task of semi-supervised learning by nature, we further devise an extended version of GraphAT, named GraphVAT which additionally smooths the distribution of predictions around each clean example to further enhance the model robustness. Inspired by the idea of virtual adversarial training [35], we further add a virtual adversarial regularizer into the training objective function and construct virtual adversarial examples to attack the local smoothness of predictions. Compared to standard AT which only considers labeled clean examples, virtual adversarial training additionally encourages the model to make consistent predictions around the unlabeled clean examples. The formulation of GraphVAT is:

$$\begin{aligned}
&\textbf{min: } \Gamma_{GVAT} = \Gamma + \underbrace{\alpha \sum_{i=1}^{N} d(f(\boldsymbol{x}_i + \boldsymbol{r}_i^v, G|\boldsymbol{\Theta}), \tilde{\boldsymbol{y}}_i)}_{\text{virtual adversarial regularizer}} + \\
&\qquad \underbrace{\beta \sum_{i=1}^{N} \sum_{j \in \mathcal{N}_i} d(f(\boldsymbol{x}_i + \boldsymbol{r}_i^g, G|\boldsymbol{\Theta}), f(\boldsymbol{x}_j, G|\boldsymbol{\Theta}))}_{\text{graph adversarial regularizer}}, \\
&\textbf{max: } \boldsymbol{r}_i^v = \arg \max_{\boldsymbol{r}_i', \|\boldsymbol{r}_i'\| \le \epsilon'} d(f(\boldsymbol{x}_i + \boldsymbol{r}_i', G|\hat{\boldsymbol{\Theta}}), \tilde{\boldsymbol{y}}_i),
\end{aligned} \tag{4}$$

where $\boldsymbol{r}_i'$ denotes the virtual adversarial perturbation, the direction that leads to the largest change on the model prediction of $\boldsymbol{x}_i$. For labeled nodes and unlabeled nodes, $\tilde{\boldsymbol{y}}_i$ denotes ground truth label and model prediction, respectively. That is,

$$\tilde{\boldsymbol{y}}_i = \begin{cases} \hat{\boldsymbol{y}}_i, & i \le M \text{ (labeled node)}, \\ f(\boldsymbol{x}_i, G|\hat{\boldsymbol{\Theta}}), & M < i \le N \text{ (unlabeled node)}. \end{cases}$$

Note that GraphVAT can be seen as jointly playing two minimax games with three players, where the two maximum players generate virtual adversarial examples and graph adversarial examples, respectively. That is, in each iteration, two types of perturbations and the associated adversarial examples are generated to attack 1) the smoothness of prediction around individual clean example; and 2) the smoothness of connected examples, respectively. By minimizing the additional regularizers over these adversarial examples, the learned model is encouraged to be more smooth and robust, achieving good generalization.

Approximation. For labeled nodes, $\boldsymbol{r}_i'$ can be easily evaluated via linear approximation [15], *i.e.*, calculating the gradient of $d(f(\boldsymbol{x}_i, G|\hat{\boldsymbol{\Theta}}), \tilde{\boldsymbol{y}}_i)$ *w.r.t.* $\boldsymbol{x}_i$. For unlabeled nodes, such approximation is infeasible since the gradient will always be zero. This is because $d(f(\boldsymbol{x}_i, G|\hat{\boldsymbol{\Theta}}), \tilde{\boldsymbol{y}}_i)$ achieves the minimum value (0) at $\boldsymbol{x}_i$ (note that $\tilde{\boldsymbol{y}}_i = f(\boldsymbol{x}_i, G|\hat{\boldsymbol{\Theta}})$ for unlabeled data). Realizing that the first-order gradient is always zero, we estimate $\boldsymbol{r}_i'$ from the second-order Taylor approximation of $d(f(\boldsymbol{x}_i + \boldsymbol{r}_i', G|\hat{\boldsymbol{\Theta}}), \tilde{\boldsymbol{y}}_i)$. That is, $\boldsymbol{r}_i^v \approx \arg\max_{\boldsymbol{r}_i', \|\boldsymbol{r}_i'\| \le \epsilon'} \frac{1}{2} \boldsymbol{r}_i'^T \boldsymbol{H} \boldsymbol{r}_i'$ where $\boldsymbol{H}$ is the Hessian matrix of $d(f(\boldsymbol{x}_i + \boldsymbol{r}_i', G|\hat{\boldsymbol{\Theta}}), \tilde{\boldsymbol{y}}_i)$. For the consideration of efficiency, we calculate $\boldsymbol{r}_i^v$ via the power iteration approximation [35]:

$$\boldsymbol{r}_i^v \approx \epsilon' \frac{\boldsymbol{g}}{\|\boldsymbol{g}\|}, \text{ where } \boldsymbol{g} = \nabla_{\boldsymbol{r}_i} d(f(\boldsymbol{x}_i + \boldsymbol{r}_i, G|\hat{\boldsymbol{\Theta}}, \tilde{\boldsymbol{y}}_i)) \,|_{\boldsymbol{r}_i = \xi \boldsymbol{d}} \;, \tag{5}$$

where $\boldsymbol{d}$ is a random vector. Detailed derivation of the method is referred to [35].

4.3 Graph Convolution Network

Inspired by the extraordinary representation ability, many neural networks have been used as the predictive model $f(\boldsymbol{x}_i, G|\boldsymbol{\Theta})$ [1], [7], [10], [11]. Under the transductive setting, Graph Convolutional Network [7] is a state-of-the-art model. Specifically, GCN stacks multiple graph convolution layers, which is formulated:

$$\boldsymbol{H}^l = \sigma\left(\widetilde{\boldsymbol{D}}^{-\frac{1}{2}} \widetilde{\boldsymbol{A}} \widetilde{\boldsymbol{D}}^{-\frac{1}{2}} \left(\boldsymbol{H}^{l-1} \boldsymbol{W}^l + \boldsymbol{b}^l\right)\right). \tag{6}$$

Specifically, the l-th graph convolution layer conducts three operations to project $\boldsymbol{H}^{l-1} \in \mathbb{R}^{N \times D^{l-1}}$ (the output of the $(l-1)$-th layer or the node features $\boldsymbol{X}$) into $\boldsymbol{H}^l \in \mathbb{R}^{N \times D^l}$, where D^{l-1} and D^l are the output dimension of layer $l-1$ and l, respectively.

- Similar as the fully connected layer, the graph convolution layer first *projects* the input ($\boldsymbol{H}^{l-1}$) into latent representations with $\boldsymbol{W}^l \in \mathbb{R}^{D^{l-1} \times D^l}$ and $\boldsymbol{b}^l \in \mathbb{R}^{D^l}$.
- It then *propagates* the latent representations ($\boldsymbol{H}^{l-1}\boldsymbol{W}^l + \boldsymbol{b}^l$) through the normalizied adjacency matrix $\widetilde{\boldsymbol{D}}^{-\frac{1}{2}} \widetilde{\boldsymbol{A}} \widetilde{\boldsymbol{D}}^{-\frac{1}{2}}$ with self-connections, where $\widetilde{\boldsymbol{D}} = \boldsymbol{D} + \boldsymbol{I}$ and $\widetilde{\boldsymbol{A}} = \boldsymbol{A} + \boldsymbol{I}$ ($\boldsymbol{I} \in \mathbb{R}^{N \times N}$ is an identity matrix). Here, the representation of node i in $\boldsymbol{H}$ is the aggregation of latent representations in ($\boldsymbol{H}^{l-1}\boldsymbol{W}^l + \boldsymbol{b}^l$) of nodes connected to i (including itself due to the self-connection).
- Finally, a non-linear activation function σ (*e.g.*, the sigmoid, hyperbolic tangent, and rectifier functions) is applied to allow non-linearity.

The original objective function of GCN is,

$$\sum_{i=1}^{M} cross\text{-}entropy(f(\boldsymbol{x}_i, G|\boldsymbol{\Theta}), \boldsymbol{y}_i) + \lambda \|\boldsymbol{\Theta}\|_F^2, \qquad (7)$$

where the second term is L_2-norm to prevent overfitting. To train GCN with our proposed GraphAT and GraphVAT, we set the Γ term in Equation 2 and 4 as the cross-entropy loss in Equation 7, which are minimized to update the parameter of GCN, respectively.

4.4 Time Complexity and Implementation

Time Complexity. As compared to GCN with standard training, the additional computation of GraphAT is twofold: 1) generating graph adversarial perturbations ($\{\boldsymbol{r}_i^g, i < N\}$) with Equation 3 and 2) calculating the value of graph adversarial regularizer ($\sum_{i=1}^{N} \sum_{j \in \mathcal{N}_i} D(f(\boldsymbol{x}_i + \boldsymbol{r}_i^g, G|\boldsymbol{\Theta}), f(\boldsymbol{x}_j, G|\boldsymbol{\Theta}))$). Considering that they can be accomplished with a back-propagation (to calculate $\boldsymbol{r}_i^g$) and a forward-propagation (to calculate $f(\boldsymbol{x}_i + \boldsymbol{r}_i^g, G|\boldsymbol{\Theta})$), the computation overhead of GraphAT is acceptable. Additionally, GraphVAT computes virtual adversarial perturbations and virtual adversarial regularizer, which can also be finished by performing one back-propagation and one forward-propagation [35]. It indicates that the overhead of GraphVAT is still acceptable [35].

Implementation. Noting that number of connected nodes varies a lot across the nodes in the graph, we sample K neighbors for each node to generate adversarial examples and calculate the graph adversarial regularizer to facilitate the calculation. Here, the following sampling strategies [28] are considered:

- **Uniform:** neighbors are selected uniformly.
- **Degree:** the probability of selecting a node is proportional to the normalized node degree.
- **Degree-Reverse:** on the contrary, the probability is the reciprocal of node degree (also normalized to sum to unity).
- **PageRank:** it performs PageRank [46] on the graph and takes the normalized pagerank score as the sampling probability.

Note that advanced but complex sampling strategies (*e.g.*, the one proposed in [28]) are not considered due to efficiency consideration.

5 Experiments

5.1 Experimental Settings

5.1.1 Datasets

We follow the same experimental settings as in [7] and conduct experiments on two types of node classification datasets: citation network datasets (Citeseer and Cora [47]) and knowledge graph (NELL [12])[2], of which the statistics are summarized in Table 2.

- In the citation networks, nodes and edges represent documents and citation links between documents, respectively. Note that the direction of edge is omitted since a citation is assumed to have equally impacts on the prediction of the associated two documents. Each document is associated with a normalized bag-of-words feature vector and a class label. During training, we use features of all nodes, but only 20 labels per class. 500 and 1,000 of the remaining nodes are used as validation and testing, respectively.
- NELL is a bipartite graph of 55,864 relation nodes and 9,891 entity nodes, extracted from a knowledge graph which is a set of triplets in the format of (e_1, r, e_2). Here e_1 and e_2 are entities, and r is the connected relation between them. Following [7], each relation r is split into two relation nodes (r_1 and r_2), from which two edges (e_1, r_1) and (e_2, r_2) are constructed. Entity nodes and relation nodes are described by bag-of-words feature vectors (normalized) and one-hot encodings, respectively. Note that we pad zero values to align the feature vectors of entity and relation nodes. Here only labels of entity nodes are available and only 0.001 of entities under each class are labeled during training.

TABLE 2: Statistics of the experiment datasets.

Dataset	#Nodes	#Edges	#Classes	#Features	Label rate
Citeseer	3,312	4,732	6	3,703	0.036
Cora	2,708	5,429	7	1,433	0.052
NELL	65,755	266,144	210	5,414	0.001

5.1.2 Baselines

We compare the following baselines:

- **LP** [8]: label propagation ignores node features and only propagates labels over the graph structure.
- **DeepWalk** [5]: it is a skip-gram based graph embedding method, which learns the embedding of a node by predicting its contexts that are generated by performing random walk on the graph.
- **SemiEmb** [48]: it learns node embeddings from node features and leverages Laplacian regularization to encourage connected nodes have close embeddings.
- **Planetoid** [12]: similar as DeepWalk, this method learns node embeddings by predicting node context, but additionally accounts for node features.
- **GCN** [7]: it stacks two graph convolution layers to project node features into labels. It propagates node representations and predictions over the graph structure to smooth the output.
- **GraphSGAN** [40]: this is a semi-supervised generative adversarial network which encodes the density signal of the graph structure during generation of fake nodes.

Note that **LP**, **DeepWalk**, **SemiEmb**, and **Planetoid** are also baselines in the paper of **GCN**, we exactly follow their settings in [7]. In addition, the setting of **GraphSGAN** is same as the original paper.

5.1.3 Parameter Settings

We implement GraphAT and GraphVAT, which train GCN with different versions of graph adversarial training, respectively, with Tensorflow. GraphAT has six hyperparameters in total: 1) D^1, the size of hidden layer (GCN); 2) λ, the weight for L_2-norm (GCN); 3) dropout ratio (GCN); 4) ϵ, the scale of graph adversarial perturbations (GraphAT); 5) β, the weight for graph adversarial regularizer (GraphAT); and 6) K, the number of sampled neighbors (GraphAT).

[2] https://github.com/kimiyoung/planetoid.

TABLE 3: Performance of the compared methods on the three datasets *w.r.t.* accuracy.

Category	Method	Citeseer	Cora	NELL
Graph	LP	45.3	68.0	26.5
	DeepWalk	43.2	67.2	58.1
+Node Features	SemiEmb	59.6	59.0	26.7
	Planetoid	64.7	75.7	61.9
	GCN	69.3	81.4	61.2
+Adversarial	GraphSGAN	73.1	**83.0**	—
	GraphVAT	**73.7**	82.6	**64.7**

For fair comparison, we set D^1, λ as the optimal values of standard GCN. But we set dropout ratio as zero in GraphAT for stable training. For the remaining three parameters, ϵ, β, and K, we performed grid-search within the ranges of [0.01, 0.05, 0.1, 0.5, 1], [0.01, 0.05, 0.1, 0.5, 1, 5], [1, 2, 3], respectively.

For GraphVAT, for simplicity, we set the six hyperparameters common to that of GraphAT using the optimal values found for GraphAT. Here we only tune its three additional hyperparameters: 1) ϵ', the scale of virtual adversarial perturbations; 2) α, the weight for virtual adversarial regularizer; and 3) ξ, the scale to calculate approximation. In particular, we perform grid-search within the ranges of [0.01, 0.05, 0.1, 0.5, 1], [0.001, 0.005, 0.01, 0.05, 0.1, 0.5], [1e-6, 1e-5, 1e-4], respectively. It should be noted that the *Uniform* strategy is adopted to sample neighbor nodes if not other specified.

5.2 Performance Comparison

5.2.1 Model Comparison

We first investigate how effective is the proposed *graph adversarial training* via comparing the performance of GraphVAT (the extended version of GraphAT) with state-of-the-art node classification methods. Table 3 shows the classification performance of the compared methods on the three datasets regarding accuracy. The performance of LP, DeepWalk, SemiEmb, and Planetoid are taken from the GCN paper [7] since we follow its settings exactly. From the results, we have the following observations:

- GraphVAT significantly outperforms the standard GCN, exhibiting relative improvements of 6.35%, 1.47%, and 5.72% on the Citeseer, Cora, and NELL datasets, respectively. As the only difference between GraphVAT and GCN is applying the proposed graph adversarial training, the improvements are attributed to the proposed training method which would enhance the stabilization and generalization of GCN. Besides, the results validate that GraphVAT is effective in solving the node classification task.
- GraphVAT achieves comparable performance as that of GraphSGAN, which is the state-of-the-art method of node classification. It demonstrates the efficacy of the proposed method. However, our method could offer a more feasible solution for two reasons: 1) GraphSGAN is based on the standard generative adversarial network, which explicitly plays a mini-max game between a discriminator and a generator (two different networks). This, inevitably, will lead to doubling of the computation of model training and the labor of parameter tuning. 2) For different node classification applications, GraphSGAN needs to be built from scratch, whereas our GraphVAT is a generic solution that can be seamlessly applied to enhance the existing models of the applications.
- GraphVAT and GraphSGAN achieve better results in all the cases as compared to the other baselines. On the Citeseer, Cora, and NELL datasets, the relative improvements are at least 6.35%, 1.97%, and 4.52%, respectively. This indicates the effectiveness of adversarial learning, *i.e.*, dynamically playing a mini-max game either implicitly (GraphVAT) and explicitly (GraphSGAN) in the training phase. Moreover, the results are consistent with findings in previous work [15], [35], [42], [49].
- Among the baselines, 1) the methods that jointly account for the graph structure and node features (in the category of *+Node Features*) outperform LP and DeepWalk that only consider graph structure. This suggests further exploration of how to combine the connection patterns and node features more appropriately. 2) As compared to SemiEmb that is a shallow model, Planetoid and GCN achieves significant improvements (from 8.56% to 131.8%) in all cases. The improvement is reasonable and attributed to the strong representation ability of neural networks. As such, methods targeting to enhance the graph neural network models, such as the graph adversarial training, will be meaningful and influential in future.

5.2.2 Performance *w.r.t.* Node Degree

We next study how the graph adversarial training performs on nodes with different densities of connections so as to understand where this regularization technique can be more suitably applied. We empirically split the nodes into three groups according to node degree (*i.e.*, the number of neighbors), where node degrees are in ranges of $[1, 2]$, $[3, 5]$, $[6, N]$. Figure 3 illustrates the distribution of nodes over the three groups. As can be seen, in all the three datasets, a great number of nodes are sparsely connected (with degrees smaller than three), and only about ten percent of the nodes are densely connected with degrees bigger than five. By separately counting the accuracy of GCN and GraphVAT over nodes in different groups, we obtain the group-oriented performance on the three datasets, which is depicted in Figure 4. From the results, we observe that:

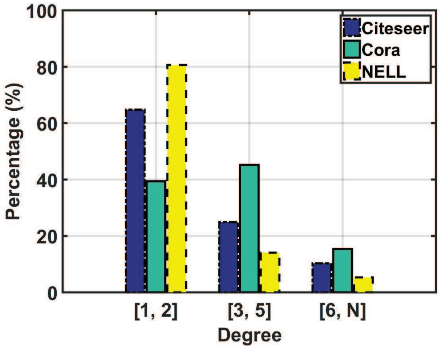
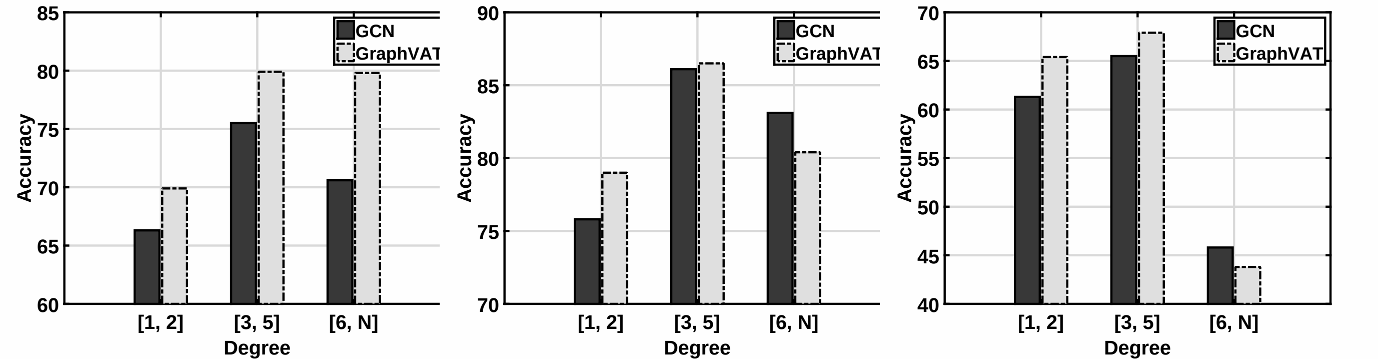


Fig. 3: Percentage of nodes with degrees in different groups in the three datasets.

- In all the three datasets, both GCN and GraphVAT achieves the best performance on the group of $[3, 5]$. The relatively worse performance on the group of $[1, 2]$ could be attributed to that the nodes in that group are sparsely



(a) Citeseer (b) Cora (c) NELL

Fig. 4: Performance of GCN and GraphVAT on nodes with different degrees in Citeseer (a), Cora (b), and NELL (c).

TABLE 4: Effect of graph adversarial regularization and virtual graph adversarial regularization.

Category	Method	Citeseer	Cora	NELL
Standard Training	GCN	69.3	81.4	61.2
Adversarial Training	GCN-VAT	72.4	79.3	63.3
	GraphAT	73.4	82.5	62.3
	GraphVAT	**73.7**	**82.6**	**64.7**

connected and lacks sufficient signals propagated from the neighbors, which are helpful for the classification [7], [8], [50]. In addition, we postulate the reason for the worse performance over nodes with degrees in $[6, N]$ as such nodes are harder to classify. This is because such nodes typically represent more general entities, such as those having connections to other entities with different types of relations and are thus harder to be accurately classified into a specific category.

- In most cases (except the $[6, N]$ group of Cora and NELL), GraphVAT outperforms GCN, which indicates that graph adversarial training would benefit the prediction of nodes with different degrees and is roughly not sensitive to the density of graph. For one of the exceptions (the $[6, N]$ group of NELL), we speculate that the reason is the under-fitting of standard GCN on such nodes (note that the performance of GCN on $[6, N]$ is 27.7% on average worse than the other two groups), where additional regularization performed by graph adversarial training worsens the under-fitting problem.
- GraphVAT significantly and consistently outperforms GCN on the group of $[1, 2]$, with an average improvement of 5.45%. The result indicates that the graph adversarial training would be more effective on sparse part of the graph. It should be noted that most of the graphs are sparse in real world applications [51]. As such this result further demonstrates the potential of the proposed methods in real world applications.

5.2.3 Method Ablation

Recall that we design two versions of graph adversarial training: 1) basic GraphAT (Equation 2) and 2) incorporating virtual adversarial training (Equation 4). To evaluate the contribution of these two types of regularizations, we compare the performance of the following solutions built upon GCN:

- **GCN**: It learns the parameters of GCN with standard training, *i.e.,* it optimizes Equation 7.
- **GCN-VAT**: Virtual adversarial training, which performs perturbations by considering node features only, is employed to train GCN, *i.e.,* optimizing Equation 4 with $\beta = 0$.
- **GraphAT**: It trains GCN by the basic GraphAT, of which the perturbations only focus on graph structure. That is optimizing Equation 4 with $\alpha = 0$.
- **GraphVAT**: It accounts for both the virtual and graph adversarial regularizations during the training of GCN.

Table 4 shows the performance of the compared methods on the three datasets *w.r.t.* accuracy. As can be seen:

- In most of the cases, GCN performs worse than the other approaches, which indicates that adversarial training could enhance the node classification model as compared to the standard training. That is, by intentionally and dynamically generating perturbations and optimizing additional regularizers, the trained model could by more accurate.
- GraphVAT achieves the best performance in all cases, *i.e.,* GraphVAT outperforms both GCN-VAT and GraphAT. It shows that perturbations targeting at both the individual nodes (virtual adversarial perturbations) and neighbor nodes (graph adversarial perturbations) benefit the training of graph neural network model. Moreover, it suggests that it is beneficial to jointly consider both the node features and graph structure in adversarial training of graph neural networks.
- Compared to GCN-VAT, GraphAT achieves improvements of 1.38% and 4.04% on the Citeseer and Cora datasets, which signifies the benefit of accounting for the graph structure in adversarial training of graph neural networks. However, on the NELL dataset, the performance of GraphAT is 1.58% worse than GCN-VAT. We speculate that the decrease is mainly because NELL is a bipartite graph where the neighbors of an entity node are all relation nodes without bag-of-words descriptions and labels. Therefore, as compared to standard graph with homogeneous nodes, the generated graph adversarial perturbations according to the predictions of connected relation nodes are less effective. It should be noted that, by resisting such perturbations, GraphAT still implicitly encourages smooth predictions of entity nodes connected

TABLE 5: Performance comparison of GraphAT with different neighbor sampling strategies during adversarial example generation. RI denotes the relative improvement over the Uniform strategy.

Sampling Strategy	Citeseer	Cora	NELL	RI
Uniform	73.4	82.5	**62.3**	-
Degree	73.0	82.9	61.8	-0.9%
Degree-Reverse	**73.8**	82.4	62.1	0.1%
PageRank	72.6	**83.1**	62.0	-0.8%

by the same relation node, which could be the reason why GraphAT outperforms standard GCN on NELL.

5.2.4 Effect of Sampling Strategies

As mentioned in Section 4.4, different strategies could be adopted to sample neighbor nodes for the generation of graph adversarial perturbations and the calculation of graph adversarial regularizer. Here, we investigate the effect of sampling strategies via comparing the results of GraphAT performing different samplings. Note that we select GraphAT rather than GraphVAT for the reason that we focus on investigating properties of the proposed graph adversarial training.

Table 5 shows the corresponding performance, from which we can observe that the performance of different sampling strategies are comparable to each other. As compared to *Uniform*, the relative improvement (RI) achieved by the other strategies is within a range of [-0.9%, 0.1%]. As such, *Uniform* would be a suitable selection since it will not bring any additional computation as compared to the other approaches.

5.3 Effect of Hyperparameters

5.3.1 Sensitivity

We then investigate how the value of hyperparameters effects the performance of the proposed GraphAT. Given a hyperparameter, we evaluate model performance when adjusting its value and fixing the other hyperparameters with optimal values. Since focusing on graph adversarial regularization, we mainly study: a) weight of graph adversarial regularizer (β), b) scale of graph adversarial perturbations (ϵ), and c) number of sampled neighbors (k), and use GraphAT to report the performance. It should be noted that, in the following, we focus on the citation graphs and omit results on NELL, which is a bipartite graph rather than a standard graph.

Figure 5 illustrates the performance of GraphAT on the validation and testing of the three datasets when varying the value of β, ϵ, and k. From the figures, we have the following observations:

- Under most cases, the performance of GraphAT changes smoothly near the optimal value of the selected hyperparameter, which indicates that GraphAT is not sensitive to hyperparameters. The only exception is that GraphAT performs significantly worse when $k = 3$ and $k = 5$ as compared to the performance with other values of k. We check the training procedure and observe that both of them are caused by triggering early stopping at the early stage of the training (dozens of epochs), which is occasional and would converge to an expected performance if disable early stopping.
- For each parameter, a) GraphAT achieves best performance with β is around 0.1, which roughly balances the contribution of the supervised loss and the graph adversarial regularizer (note that the supervised loss decreases fast at the early epochs). Larger value of β (stronger regularization) will harm GraphAT since the model could suffer from the underfitting issue. b) GraphAT performs well when ϵ is in the range of [1e-4, 1e-2], but the performance decreases significantly as increasing ϵ. This result supports the assumption that perturbations have to be in a small scale so that the constructed adversarial examples have similar feature distributions as the real data. c) GraphAT performs best when $k = 1$ or $k = 2$, which is somehow coherent with the result in Figure 4 that graph adversarial training are more effective to nodes with degree in [1, 2]. This result is appealing since the computation cost linearly increases as sampling more neighbors.

TABLE 6: Performance of GraphAT as tuning all hyperparameters (*i.e.*, β, ϵ, and k) and tuning ϵ with fixed $\beta = 1.0$ and $k = 1$.

Hyperparameter	Citeseer	Cora
$\{\beta,\ \epsilon,\ k\}$	73.4	**82.5**
$\{\epsilon\}$	**73.6**	**82.5**

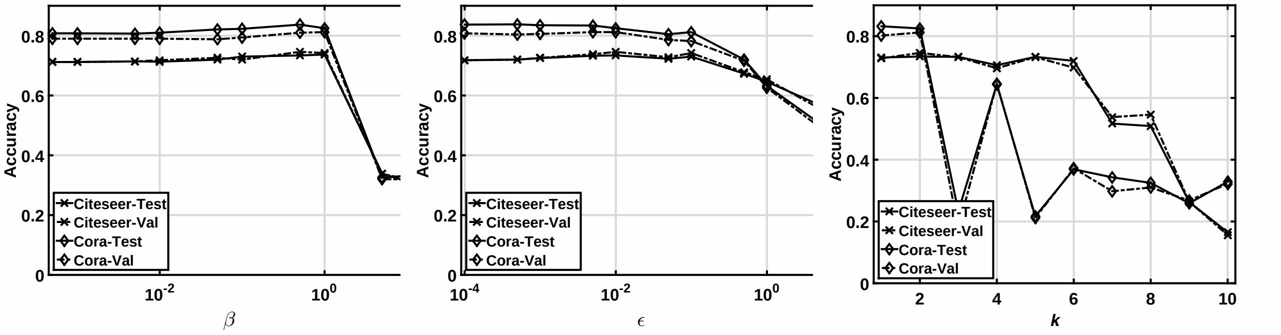
5.3.2 Tuning ϵ Only

Considering that the number of candidate combinations exponentially increases with the number of hyperparameters, we explore whether comparable performance could be achieved when tune one hyperparameter alone and fix the others with empirical values. It should be noted that previous work [35] has shown that tuning ϵ' alone could suffice for achieving satisfactory performance of VAT. Similarly, we tune ϵ with $\beta = 1$ and $k = 1$ and summarize the performance of GraphAT in Table 6. As can be seen, on the citation graphs, tuning ϵ alone achieves satisfactory performance. As such, the overhead of additional hyperparameters of the proposed GraphAT could be ignored.

5.4 Impact of Graph Adversarial Training

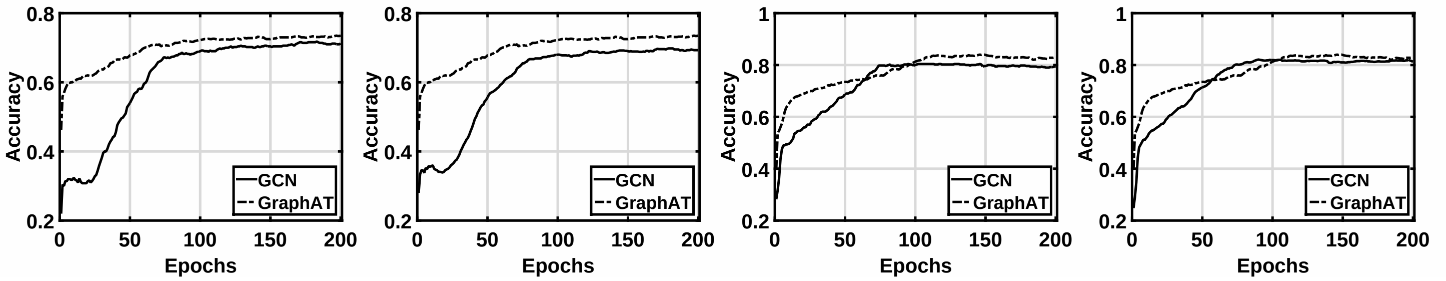
5.4.1 Training Process

We next study the effect of GraphAT on the training process of GCN. Specifically, we observe the performance of GCN and GraphAT on the validation and testing of Citeseer and Cora after every training epoch, which is depicted in Figure 6. As can be seen, 1) On the two datasets, the performance of GCN and GraphAT becomes stable after 100 epochs, which indicates that graph adversarial training will not affect the convergence speed of GCN. 2) It is interesting to see that the performance of GraphAT increases faster than standard GCN during the initial several epochs. Note that the supervised loss is typically much larger (about 1e5 times) than the value of graph adversarial regularizer at the initial epochs. This is because all nodes are assigned predictions close to random at the beginning which leads to tiny divergence between connected nodes. As such, the



(a) Weight of GraphAT regularization (b) Scale of perturbation (c) Number of sampled neighbors

Fig. 5: Performance of GraphAT with different values of hyperparameters: (a) weight of graph adversarial regularizer (β), (b) scale of graph adversarial perturbations (ϵ), and (c) number of sampled neighbors (k) on the validation and testing of the three datasets (When investigating the effect of a hyperparameter, the other two are set as the optimal values).



(a) Validation (Citeseer) (b) Testing (Citeseer) (c) Validation (Cora) (d) Testing (Cora)

Fig. 6: Training curves of the GCN and GraphAT on the validation and testing of Citeseer and Cora.

TABLE 7: The impact of adding graph adversarial perturbations to GCN and GraphAT. The number shows the relative decrease of testing accuracy.

Method	Citeseer	Cora
GCN	-21.1%	-6.6%
GraphAT	-4.1%	-1.6%

acceleration of performance increase at the initial epochs is believed to be the effect of data augmentation (additional adversarial examples) rather then a better regularization.

5.4.2 Robustness against Adversarial Perturbations

Recall that our target is to enhance the robustness of graph neural networks. Table 7 shows relative performance decrease of GCN and GraphAT on adversarial examples as compared to clean examples. As can be seen, by training GCN with graph adversarial training, the model becomes less sensitive to graph adversarial perturbations. Graph adversarial perturbations in the scale of 0.01 (*i.e.*, $\epsilon = 0.01$) decreases accuracy of GCN by 13.9% on average, while the number is only 2.9% for GraphAT. It validates that the graph adversarial training technique could enhance the robustness of a GCN model.

5.4.3 Effect of Graph Adversarial Training on Divergence of Neighbor Nodes

We retrospect that the intuition of graph adversarial regularizer is to encourage connected nodes to be predicted similarly. Table 8 shows the effect of applying graph adversarial training to train GCN, from which we can see that graph adversarial training reduces the divergence between connected nodes as expected. These results show that the predictions of GraphAT are more smooth over the graph structure, which indicates the stronger generalization ability and robustness of the trained model.

TABLE 8: Average Kullback-Leibler divergence between connected node pairs calculated from predictions of GCN and GraphAT (small value indicates close predictions).

Method	Citeseer		Cora	
	Test	All	Test	All
GCN	0.132	0.137	0.345	0.333
GraphAT	0.127	0.130	0.308	0.299

6 Conclusion

In this work, we proposed a new learning method, named *graph adversarial training*, which additional accounts for relation between examples as compared to standard adversarial training. By iteratively generating adversarial examples attacking the graph smoothness constraint and learning over adversarial examples, the proposed method encourages the smoothness of predictions over the given graph, a property indicating good generalization of the model. As can be seen as a dynamic regularization technique, our method is generic to be applied to train most graph neural network models. We trained one well-established model, GCN, with the proposed method to solve the node classification task. By conducting experiments on three benchmark datasets, we demonstrated that training GCN with our method is remarkably effective, achieving an average improvement of

4.51%. Moreover, it also beats GCN trained with VAT, indicating the necessity of performing AT with graph structure considered

In future, we will explore we are interested to explore the effectiveness of GAD on more graph neural network models [3], [4], [11]. Moreover, we are interested to investigate the effect of GAD on other graph-based learning tasks such as link prediction and community detection. As focusing on graph-based learning with only one graph in this paper, one potential future work is to investigate the effectiveness of graph adversarial training for graph-based learning methods simultaneously handling multiple graphs. In addition, we are interested in testing the performance of graph adversarial training on graphs with specifical structures, for instance, hyper-graphs and heterogeneous information graphs. Moreover, we would like to incorporate techniques like robust optimization [52] and adversarial dropout [53] into the proposed method to further enhance its ability of stabilizing graph neural network models.

Acknowledgments

This research is part of NExT++ reserach, supported by the National Research Foundation Singapore under its AI Singapore Programme (AISG-100E-2018-012), Prime Minister's Office under its IRC@SG Funding Initiative, and National Natural Science Foundation of China (61972372). We thank the anonymous reviewers and the associated editor for their reviewing efforts.

References

[1] D. Wang, P. Cui, and W. Zhu, "Structural deep network embedding," in *SIGKDD*. ACM, 2016, pp. 1225–1234.

[2] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *SIGKDD*. ACM, 2016, pp. 855–864.

[3] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in Neural Information Processing Systems*, 2017, pp. 1024–1034.

[4] Z. Ying, J. You, C. Morris, X. Ren, W. Hamilton, and J. Leskovec, "Hierarchical graph representation learning with differentiable pooling," in *Advances in Neural Information Processing Systems*, 2018, pp. 4800–4810.

[5] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2014, pp. 701–710.

[6] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "Line: Large-scale information network embedding," in *Proceedings of the 24th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 2015, pp. 1067–1077.

[7] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *ICLR*, 2017.

[8] X. Zhu, Z. Ghahramani, and J. D. Lafferty, "Semi-supervised learning using gaussian fields and harmonic functions," in *Proceedings of the 20th International conference on Machine learning (ICML-03)*, 2003, pp. 912–919.

[9] D. Zhou, O. Bousquet, T. N. Lal, J. Weston, and B. Schölkopf, "Learning with local and global consistency," in *Advances in neural information processing systems*, 2004, pp. 321–328.

[10] J. Ni, S. Chang, X. Liu, W. Cheng, H. Chen, D. Xu, and X. Zhang, "Co-regularized deep multi-network embedding," in *Proceedings of the 2018 World Wide Web Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 2018, pp. 469–478.

[11] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *ICLR*, vol. 1, no. 2, 2018.

[12] Z. Yang, W. Cohen, and R. Salakhudinov, "Revisiting semi-supervised learning with graph embeddings," in *International Conference on Machine Learning*, 2016, pp. 40–48.

[13] D. Zügner, A. Akbarnejad, and S. Günnemann, "Adversarial attacks on neural networks for graph data," in *SIGKDD*. ACM, 2018, pp. 2847–2856.

[14] D. Zhu, Z. Zhang, P. Cui, and W. Zhu, "Robust graph convolutional networks against adversarial attacks," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 2019, pp. 1399–1407.

[15] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *ICLR*, 2015.

[16] A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial machine learning at scale," *ICLR*, 2017.

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

[18] F. Feng, H. Chen, X. He, J. Ding, M. Sun, and T.-S. Chua, "Enhancing stock movement prediction with adversarial training," in *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, 2019, pp. 5843–5849.

[19] Y. Liu, Z. Li, C. Zhou, Y. Jiang, J. Sun, M. Wang, and X. He, "Generative adversarial active learning for unsupervised outlier detection," *IEEE Transactions on Knowledge and Data Engineering*, 2019.

[20] P. Wang, Y. Fu, H. Xiong, and X. Li, "Adversarial substructured representation learning for mobile user profiling," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 2019, pp. 130–138.

[21] J. Tang, X. Du, X. He, F. Yuan, Q. Tian, and T.-S. Chua, "Adversarial training towards robust multimedia recommender system," *IEEE Transactions on Knowledge and Data Engineering*, 2019.

[22] M. Belkin, P. Niyogi, and V. Sindhwani, "Manifold regularization: A geometric framework for learning from labeled and unlabeled examples," *Journal of machine learning research*, pp. 2399–2434, 2006.

[23] P. P. Talukdar and K. Crammer, "New regularized algorithms for transductive learning," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 2009, pp. 442–457.

[24] F. Feng, X. He, Y. Liu, L. Nie, and T.-S. Chua, "Learning on partial-order hypergraphs," in *Proceedings of the 2018 World Wide Web Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 2018, pp. 1523–1532.

[25] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and locally connected networks on graphs," *ICLR*, 2014.

[26] 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 *Advances in neural information processing systems*, 2015, pp. 2224–2232.

[27] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Advances in Neural Information Processing Systems*, 2016, pp. 3844–3852.

[28] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, "Graph convolutional neural networks for web-scale recommender systems," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery &#38; Data Mining*. ACM, 2018, pp. 974–983.

[29] Z. Zhang, P. Cui, and W. Zhu, "Deep learning on graphs: A survey," *arXiv preprint arXiv:1812.04202*, 2018.

[30] X. Wang, X. He, M. Wang, F. Feng, and T.-S. Chua, "Neural graph collaborative filtering," in *Proceedings of the 42Nd International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, 2019, pp. 165–174.

[31] H. Dai, H. Li, T. Tian, X. Huang, L. Wang, J. Zhu, and L. Song, "Adversarial attack on graph structured data," in *ICML*, vol. 80. PMLR, 2018, pp. 1115–1124.

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

[33] S.-M. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard, "Universal adversarial perturbations," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[34] Y. Wu, D. Bamman, and S. Russell, "Adversarial training for relation extraction," in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, 2017, pp. 1778–1783.

[35] T. Miyato, S.-i. Maeda, S. Ishii, and M. Koyama, "Virtual adversarial training: a regularization method for supervised and

semi-supervised learning," *IEEE transactions on pattern analysis and machine intelligence*, 2018.

[36] F. Liao, M. Liang, Y. Dong, and T. Pang, "Defense against adversarial attacks using high-level representation guided denoiser," *CVPR*, 2018.

[37] F. Tramèr, A. Kurakin, N. Papernot, I. Goodfellow, D. Boneh, and P. McDaniel, "Ensemble adversarial training: Attacks and defenses," *ICLR*, 2018.

[38] A. Raghunathan, J. Steinhardt, and P. Liang, "Certified defenses against adversarial examples," *ICLR*, 2019.

[39] H. Wang, J. Wang, J. Wang, M. Zhao, W. Zhang, F. Zhang, X. Xie, and M. Guo, "Graphgan: Graph representation learning with generative adversarial nets," *AAAI*, 2017.

[40] M. Ding, J. Tang, and J. Zhang, "Semi-supervised learning on graphs with generative adversarial nets," in *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*. ACM, 2018, pp. 913–922.

[41] L. Sang, M. Xu, S. Qian, and X. Wu, "Aaane: Attention-based adversarial autoencoder for multi-scale network embedding," in *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, 2019, pp. 3–14.

[42] W. Yu, C. Zheng, W. Cheng, C. C. Aggarwal, D. Song, B. Zong, H. Chen, and W. Wang, "Learning deep network representations with adversarially regularized autoencoders," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 2018, pp. 2663–2671.

[43] S. Pan, R. Hu, G. Long, J. Jiang, L. Yao, and C. Zhang, "Adversarially regularized graph autoencoder for graph embedding." in *IJCAI*, 2018, pp. 2609–2615.

[44] Q. Dai, Q. Li, J. Tang, and D. Wang, "Adversarial network embedding," *AAAI*, 2018.

[45] J. M. Joyce, "Kullback-leibler divergence," *International encyclopedia of statistical science*, pp. 720–722, 2011.

[46] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web." Stanford InfoLab, Tech. Rep., 1999.

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

[48] J. Weston, F. Ratle, H. Mobahi, and R. Collobert, "Deep learning via semi-supervised embedding," in *Neural Networks: Tricks of the Trade*. Springer, 2012, pp. 639–655.

[49] X. He, Z. He, X. Du, and T.-S. Chua, "Adversarial personalized ranking for recommendation," in *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*. ACM, 2018, pp. 355–364.

[50] H. Gao, Z. Wang, and S. Ji, "Large-scale learnable graph convolutional networks," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 2018, pp. 1416–1424.

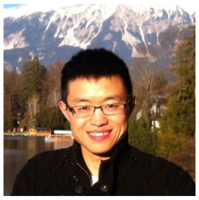
[51] P. Cui, X. Wang, J. Pei, and W. Zhu, "A survey on network embedding," *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 5, pp. 833–852, 2018.

[52] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," *arXiv preprint arXiv:1706.06083*, 2017.

[53] S. Park, J.-K. Park, S.-J. Shin, and I.-C. Moon, "Adversarial dropout for supervised and semi-supervised learning," *AAAI*, pp. 3917–3924.

Fuli Feng is a Ph.D. student in the School of Computing, National University of Singapore. He received the B.E. degree in School of Computer Science and Engineering from Baihang University, Beijing, in 2015. His research interests include information retrieval, data mining, and multi-media processing. He has over 10 publications appeared in several top conferences such as SIGIR, WWW, and MM. His work on Bayesian Personalized Ranking has received the Best Poster Award of WWW 2018. Moreover, he has been served as the PC member and external reviewer for several top conferences including SIGIR, ACL, KDD, IJCAI, AAAI, WSDM etc.

Xiangnan He is currently a research fellow with School of Computing, National University of Singapore (NUS). He received his Ph.D. in Computer Science from NUS. His research interests span recommender system, information retrieval, natural language processing and multimedia. His work on recommender system has received the Best Paper Award Honorable Mention in WWW 2018 and SIGIR 2016. Moreover, he has served as the PC member for top-tier conferences including SIGIR, WWW, MM, KDD, WSDM, CIKM, AAAI, and ACL, and the invited reviewer for prestigious journals including TKDE, TOIS, TKDD, TMM, and WWWJ.



Jie Tang is an associate professor with the Department of Computer Science and Technology, Tsinghua University. His main research interests include data mining algorithms and social network theories. He has been a visiting scholar with Cornell University, Chinese University of Hong Kong, Hong Kong University of Science and Technology, and Leuven University. He has published more then 100 research papers in major international journals and conferences including: KDD, IJCAI, AAAI, ICML, WWW, SIGIR, SIGMOD, ACL, Machine Learning Journal, TKDD, and TKDE.

Tat-Seng Chua Tat-Seng Chua is the KITHCT (short for Kwan Im Thong Hood Cho Temple) Chair Professor at the School of Computing, National University of Singapore. He was the Acting and Founding Dean of the School from 1998-2000. Dr Chuas main research interest is in multimedia information retrieval and social media analytics. In particular, his research focuses on the extraction, retrieval and question-answering (QA) of text and rich media arising from the Web and multiple social networks. He is the co-Director of NExT, a joint Center between NUS and Tsinghua University to develop technologies for live social media search. Dr Chua is the 2015 winner of the prestigious ACM SIGMM award for Outstanding Technical Contributions to Multimedia Computing, Communications and Applications. He is the Chair of steering committee of ACM International Conference on Multimedia Retrieval (ICMR) and Multimedia Modeling (MMM) conference series. Dr Chua is also the General Co-Chair of ACM Multimedia 2005, ACM CIVR (now ACM ICMR) 2005, ACM SIGIR 2008, and ACMWeb Science 2015. He serves in the editorial boards of four international journals. Dr. Chua is the co-Founder of two technology startup companies in Singapore. He holds a PhD from the University of Leeds, UK.