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* (GAT), which takes the impact from connected examples into account when learning to construct and resist perturbations. We give a general formulation of GAT, which can be seen as a dynamic regularization scheme based on the graph structure. To demonstrate the utility of GAT, 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 GAT which outperforms normal training on GCN by 4.51% in node classification accuracy. Codes will be released upon acceptance.

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

1 Introduction

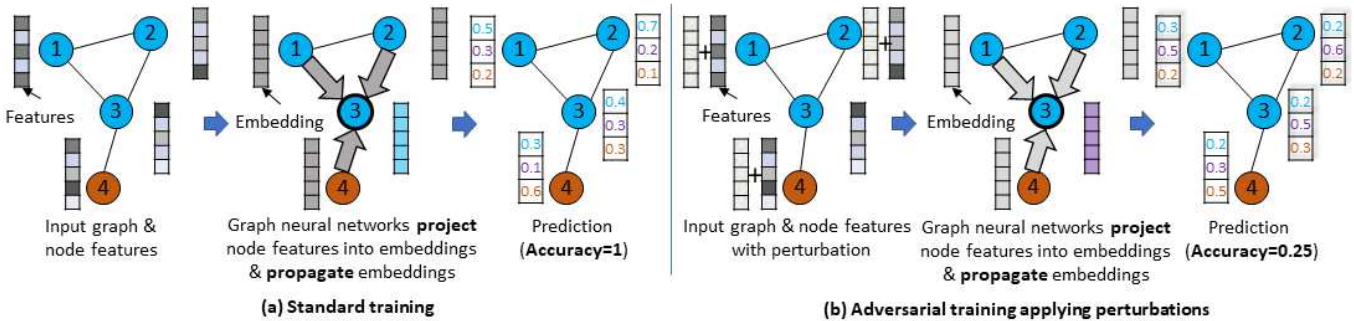
Graph-based learning makes predictions by accounting for both input features of examples and the relations between examples. It is remarkably effective for a wide range of applications, such as predicting the profiles and interests of social network users [1], [2], predicting the role of a protein in 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], 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 [14], 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 [14]. It has been empirically shown to be able to stabilize neural networks, and enhance their robustness against perturbations in standard classification tasks [15], [16]. 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 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.


Features
1
2
3
4
Input graph & node features
Embedding
Graph neural networks **project** node features into embeddings & **propagate** embeddings
0.5
0.3
0.2
0.7
0.2
0.1
0.4
0.3
0.3
0.3
0.1
0.6
Prediction (**Accuracy=1**)
Input graph & node features with perturbation
0.3
0.5
0.2
0.2
0.6
0.2
0.2
0.5
0.3
0.2
0.3
0.5
Prediction (**Accuracy=0.25**)

(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.

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* (GAT), which learns to construct and resist perturbations by taking the graph structure into account.

The key idea of GAT 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, GAT 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, GAT 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 GAT, 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 GAT — compared to normal training on GCN, GAT 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 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 GAT on GCN, conducting experiments on three datasets which show that our method achieves state-of-the-art performance for node classification. Codes will be available to facilitate the community.

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], [17]–[19].

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], [20]–[23].

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], [24], 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 [25]. 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 [14], [16], [26]–[31]. 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 [14], supervised loss [14], [26], [27] and its surrogates [29]–[31] 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 [24] and [13] are the only published work. However, methods in [24] 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 [32], [33] or smooth the leaned embeddings to follow a prior distribution [34]–[37]. 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.

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.

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 $(N - M, N]$, respectively. The target of node classification is to learn a prediction function (classifier) $\hat{\boldsymbol{y}}_{\boldsymbol{i}} = f(\boldsymbol{x}_i, G||\times)$, to forecast the label of the node.

3.3 Graph-based Learning

Graph-based learning methods have been shown remarkably effective on solving the node classification task [8], [9], [17], [18]. Generally, most of the 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], [17], [18]. 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 *GATV*, an extension of GAT, which incorporates the virtual adversarial regularization [28]. We then present two solutions for the node classification task, *GCN-GAT* and *GCN-GATV*, which employ GAT and GATV 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* (AT) 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 [14], [15], [28] and text classification [16]. Generally, applying AT would regulate the model parameters to smooth the output distribution. Specifically, for each clean example in the dataset, adversarial training 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.

Generally, the formulation of graph adversarial training 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 [38]) 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 adversarial training, each iteration of GAT can also be viewed as playing a *minimax game*:

- **Maximization**: GAT 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**: GAT 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.

While the traditional graph-based regularizations (*e.g.,* the graph Laplacian term) also encourage the smoothness of predictions over the graph structure, GAT is believed to be a more advanced regulation for two reasons: 1) the regularization performed by GAT 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) GAT 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 [14] for standard adversarial training, we also design a linear approximation method to calculate the graph adversarial perturbations in GAT, 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 Adversarial Training

Considering that node classification is a task of semi-supervised learning by nature, we further devise an extended version of GAT (GATV), 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 [28], we further add a virtual adversarial regularizer into the training objective function and construct virtual adversarial examples to attack the local smoothness of predictions. The formulation of GATV is:

$$\textbf{min: } \Gamma_{GATV} = \Gamma + \underbrace{\alpha \sum_{i=1}^{N} D(f(\boldsymbol{x}_i + \boldsymbol{r}_i^v, G|\boldsymbol{\Theta}), \tilde{\boldsymbol{y}}_i)}_{\text{virtual adversarial regularizer}} +$$

$$\underbrace{\beta \sum_{i=1}^{N} \sum_{j \in \mathcal{N}_i} D(f(\boldsymbol{x}_i + \boldsymbol{r}_i^g, G|\boldsymbol{\Theta}), f(\boldsymbol{x}_j, G|\boldsymbol{\Theta}))}_{\text{graph adversarial regularizer}},$$

$$\textbf{max: } \boldsymbol{r}_i^v = \arg \max_{\boldsymbol{r}_i', \|\boldsymbol{r}_i'\| \leq \epsilon'} D(f(\boldsymbol{x}_i + \boldsymbol{r}_i', G|\hat{\boldsymbol{\Theta}}), \tilde{\boldsymbol{y}}_i), \tag{4}$$

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,

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

Note that GATV 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 [14], *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, $\boldsymbol{r}_i^v \approx \arg\max_{\boldsymbol{r}_i', \|\boldsymbol{r}_i'\| \leq \epsilon'} \frac{1}{2} \boldsymbol{r}_i'^T \boldsymbol{H} \boldsymbol{r}_i'$ 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 [28]:

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

where $\boldsymbol{d}$ is a random vector. Detailed derivation of the method is referred to [28].

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,

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

where the second term is L_2-norm to prevent overfitting. To train GCN with our proposed GAT and GATV, 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 GCN-GAT 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 and a forward-propagation (to calculate $f(\boldsymbol{x}_i + \boldsymbol{r}_i^g, G|\boldsymbol{\Theta})$), the computation overhead of GCN-GAT is acceptable. Additionally, GCN-GATV computes virtual adversarial perturbations and virtual adversarial regularizer, which can be performed with one back-propagation and one forward-propagation, respectively [28]. It indicates that the overhead of GCN-GATV is still acceptable [28]. Running time comparison in Section 5.4.2 further demonstrate the efficiency of GCN-GAT and GCN-GATV.

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 are considered:

- **Uniform:** neighbors are selected uniformly.

TABLE 1: 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

- **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 [39] on the graph and takes the normalized pagerank score as the sampling probability.

Note that advanced but complex sampling strategies (*e.g.*, the one in [23]) 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 [40]) and knowledge graph (NELL [12])[2], of which the statistics are summarized in Table 1.

- 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 availabe. During training, only 0.001 of entities under each class are labeled.

5.1.2 Baselines

We compare the following baselines:

- **LP** [8]: Label propagation ignores node features and only propagates labels to unlabeled nodes with a graph Laplacian term.
- **DeepWalk** [5]: DeepWalk is a skip-gram based method to learn graph embeddings, which uses the embedding of a node to predict node contexts that are generated by performing random walk on the graph.
- **SemiEmb** [41]: SemiEmb learns embeddings for nodes from node features and leverages Laplacian regularization to encourage connected nodes have close embeddings.
- **Planetoid** [12]: Planetoid also learns node embeddings from input features but accounts for the graph structure in the fashion of DeepWalk, *i.e.*, predicting node context.
- **GCN** [7]: GCN stacks two graph convolution layers to project node features into labels and propagates node representations and predictions over the graph structure to smooth the output.
- **GraphSGAN** [33]: GraphSGAN is a semi-supervised generative adversarial network which encodes the density signal of the graph during generation of fake nodes.

Since **LP**, **DeepWalk**, **SemiEmb**, and **Planetoid** are all 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 GCN-GAT and GCN-GATV, which train GCN with different versions of graph adversarial training, respectively, with Tensorflow (the implementations are available via https://anonymous.com). In total, GCN-GAT has six hyperparameters: D^1 size of hidden layer (GCN), λ weight for L_2-norm (GCN), dropout ratio (GCN), ϵ the scale of graph adversarial perturbations (GAT), β weight for graph adversarial regularizer (GAT), and K the number of sampled neighbors (GAT). For fair comparison and simplification, we set D^1, λ as the optimal values of standard GCN. But we set dropout ratio as zero in GCN-GAT 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, on the validation set.

Additionally, GCN-GATV has three more hyperparameters: ϵ' the scale of virtual adversarial perturbations, α weight for virtual adversarial regularizer as well as P and ξ in the approximation of virtual adversarial perturbations. Again, for simplification, we first set the other parameters with the optimal value of GCN-GAT, and empirically set $P = 1$ since previous work demonstrated that increasing P would not bring substantial improvements [28]. We then 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.

The selected values for hyperparameters of both GCN-GAT and GCN-GATV would be released together with the implementation. Moreover, similar as standard GCN, we train the models via Adam [42] with a learning rate of 0.01 and early stopping with a window size of 10, i.e. training stops if the validation loss does not decrease for 10 consecutive epochs.

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 GCN-

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

TABLE 2: 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**	—
	GCN-GATV	**73.7**	82.6	**64.7**

GATV with state-of-the-art node classification methods. Table 2 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 exactly followed its settings. We employ the public implementation[3] of GCN with same settings as the origin paper to report its performance on Citeseer and Cora. For the performance of GCN on NELL, we tune its hyperparameters with grid search since the setting released in the GCN paper [7] achieves performance (lower than 50.0) much worse than expected[4]. In [33] GraphSGAN is also evaluated on the Citeseer and Cora datasets with a similar setting, we hence directly copy the reported performance.

From the results, we have the following observations:

- GCN-GATV 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 GCN-GATV 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 the standard GCN. Besides, the results justify that GCN-GATV is effective in solving the node classification task.
- GCN-GATV achieves comparable performance as GraphSGAN, which is the state-of-the-art of the node classification task, demonstrating the efficacy of the proposed method. Moreover, our method is believed to be a more feasible solution for two reasons: 1) GraphSGAN is in the fashion of generative adversarial networks, which explicitly play a mini-max game between a discriminator and a generator (two different networks), inevitably doubling the computation of model training and the labor of parameter tuning. 2) For different applications, one may need to build GraphSGAN from scratch, whereas our GCN-GATV is a generic solution can be seamlessly applied to enhance the existing models of the applications.
- GCN-GATV 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 (GCN-GATV) and explicitly (GraphSGAN) in the training phase. Moreover, the results are consistent with findings in previous work [14], [28], [35], [43].
- Among the baselines, 1) methods jointly account for the graph structure and node features (in the category of *+Node Features*) outperforms 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, 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. It suggests that neural networks would be beneficial once node features are incorporated. 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 then study how the graph adversarial training performs on nodes with different densities of connections so as to understand where this regularization technique is suitable for. 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 $[1, 2]$, $[3, 5]$, $[6, N]$, respectively. Figure 2 illustrates the distribution of nodes in the three datasets over the groups. As can be seen, in all the three datasets, a great number of nodes are sparsely connected (*i.e.,* with degrees smaller than three), and only about ten percent of the nodes are densely connected with degrees bigger than five. Note that we omit the distribution of testing nodes since they are randomly sampled from the whole node set and roughly follow the same distributions. By separately counting the accuracy of standard GCN and GCN-GATV over nodes in different groups, we obtain the group-oriented performance on the three datasets, which is depicted in Figure 3. From the results, we observe that:

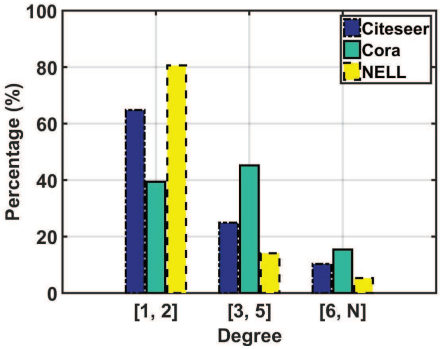
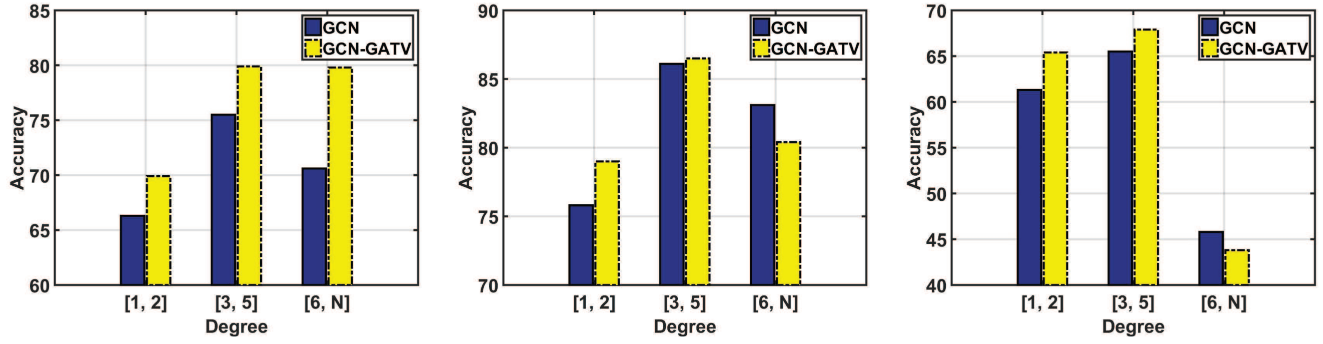


Fig. 2: Percentage of nodes with degrees in different groups in the three datasets.

- In all the three datasets, both of GCN and GCN-GATV 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 the group are sparsely connected and lacks enough signals propagated from neighbors, which are helpful for the classification [7], [8], [44]. In addition, we postulate the reason of the worse performance over nodes with degrees in $[6, N]$ as

[3] https://github.com/tkipf/gcn.

[4] According to the record on GitHub (https://github.com/tkipf/gcn/issues/14), we are not the first one struggling for reproducing the performance. And the author of the GCN paper suggests us to tune the hyperparameters by ourselves.



(a) Citeseer

(b) Cora

(c) NELL

Fig. 3: Performance comparison between GCN and GCN-GATV on nodes with different degrees in the Citeseer (a), Cora (b), and NELL (c) datasets.

TABLE 3: Effect of graph adversarial regularization and virtual 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
	GCN-GAT	73.4	82.5	62.3
	GCN-GATV	**73.7**	**82.6**	**64.7**

such nodes are harder to be classified. This is because such nodes typically represent more general entities, for instance, an entity having connections to other entities with different types of relations might be a more general concept, and hard to be accurately classified into a specific category.

- In most cases (except the $[6, N]$ group of Cora and NELL), GCN-GATV outperforms the standard 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 speculated that the reason is the under-fitting of standard GCN on such nodes (note that the performance of GCN on $[6, N]$ is averagely 27.7% worse than the other two groups), where additional regularization performed by graph adversarial training worsens it. Investigating the reason of the other exception (the $[6, N]$ group of Cora) is left to future work.
- GCN-GATV significantly and consistently outperforms GCN on the group of $[1, 2]$ across all three datasets, 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 [45]. As such this result further demonstrates the potential of the proposed methods in solving more real world applications.

5.2.3 Method Ablation

Recall that we design two versions of graph adversarial training: 1) basic GAT (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 standard training, optimizing 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 $\beta = 0$.
- **GCN-GAT**: It trains GCN by the pure GAT, of which the perturbations only focus on only graph structure,*i.e.*, optimizing Equation 4 with $\alpha = 0$.
- **GCN-GATV**: It accounts for both the virtual and graph adversarial regularizations during the training of GCN.

Table 3 shows the performance of 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.
- GCN-GATV achieves the best performance in all cases. It justifies that perturbations targeting on individual nodes (virtual adversarial perturbations) and connected nodes (graph adversarial perturbations) both benefits the training of graph neural network model. Moreover, it suggests joint consideration of node features and the graph structure in adversarial training on graph data.
- Compared to GCN-VAT, GCN-GAT 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 GCN-GAT is 1.58% worse than GCN-VAT, which is reasonable. We speculate that the decrease is mainly because NELL is a bipartite graph where the connected nodes of an entity node are all relation nodes without bag-of-words descriptions and predictions as meaningful as entity nodes. 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, GAT still implicitly encourages smooth predictions of entity nodes connected by the same relation node, which could

TABLE 4: Performance comparison of GCN-GAT with different strategies to sample neighbors during adversarial example generation.

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

be the reason why GCN-GAT outperforms standard GCN on NELL.

5.2.4 Effect of Sampling Strategies

As mentioned in Section 4.4, different sampling 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 GCN-GAT performing different samplings. It should be noted that we separately tune the hyperparameters of GCN-GAT when different samplings are employed. Table 4 shows the corresponding performance, from which we can observe that the performance of different sampling strategies are comparable to each other. It indicates that the efficacy of GCN-GAT is not sensitive to sampling strategies, 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 method. Given a hyperparameter, we evaluate model performance when adjusting its value and fixing the other hyperparameters with optimal values. Considering that effect of hyperparameters relevant to virtual adversarial training, the weight for virtual adversarial regularizer (α), the magnitude of virtual adversarial perturbations (ϵ'), and the magnitude of input to calculate the perturbations (ξ), has been studied in previous work [28], we focus on the remaining ones: a) weight of graph adversarial regularizer (β), b) scale of graph adversarial perturbations (ϵ), and c) number of sampled neighbors (k), and used GCN-GAT to report the performance.

Figure 4 illustrates the performance of GCN-GAT 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 GCN-GAT changes smoothly near the optimal value of the selected hyperparameter, which indicates that GCN-GAT is not sensitive to hyperparameters. The only exception is that GCN-GAT 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.

TABLE 5: Performance of GCN-GAT as tuning all hyperparameters (*i.e.*, β, ϵ, and k) and tuning ϵ with fixed $\beta = 1.0$ and $k = 1$.

Hyperparameter	Citeseer	Cora	NELL
$\{\beta,\ \epsilon,\ k\}$	73.4	**82.5**	**62.3**
$\{\epsilon\}$	**73.6**	**82.5**	45.4

- For individual parameter, a) GCN-GAT achieves best performance with k around 0.1, which roughly balance the contribution of the supervised loss and the graph adversarial regularizer (note that the supervised loss decreases fast in the early epochs). Larger value of k (stronger regularization) will harm GCN-GAT since the model could suffer from underfitting. b) GCN-GAT performs well when ϵ is in the range of [1e-4, 1e-2], but the performance decrease significantly as increasing ϵ. This justifies the assumption that perturbations have to be in small scale so that the constructed adversarial examples have similar feature distributions as real data. c) On all the three datasets, GCN-GAT performs best when $k = 1$ or $k = 2$, which is somehow coherent with the result in Figure 3 that graph adversarial training are more effective to nodes with degree in [1, 2]. The specific reasons of this result are left for future exploration.

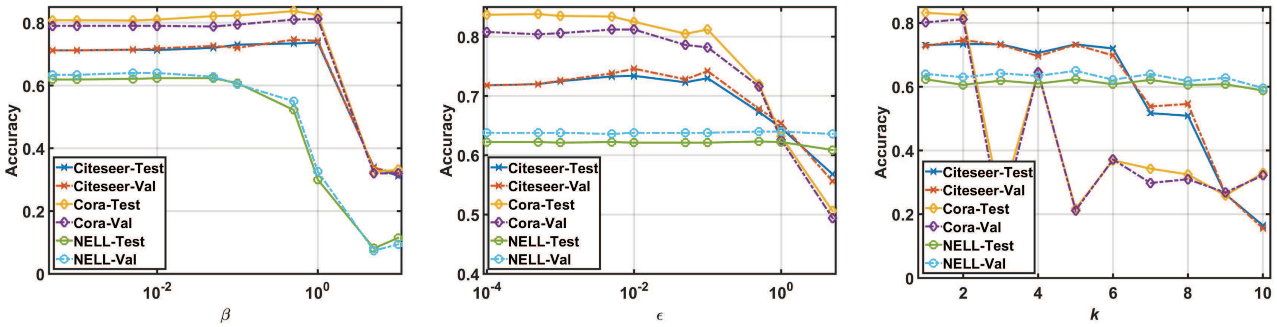
5.3.2 Tuning ϵ Only

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 [28] 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 GCN-GAT in Table 5. As can be seen, on the citation graphs, tuning ϵ alone achieves satisfactory performance, whereas the performance on NELL is not desirable. We find that the graph adversarial regularizer would get much larger value on the NELL dataset as compared to the other two citation datasets, which might caused by the larger number of classes (210 in NELL). By setting $\beta = 0.01$ and $\beta = 0.1$, which roughly balance the supervised loss and the regularizer, we obtain satisfactory performance when tune ϵ alone. Therefore, we would conclude that the hyperparameter search for only ϵ suffices for achieving satisfactory performance.

5.4 Impact of Graph Adversarial Training

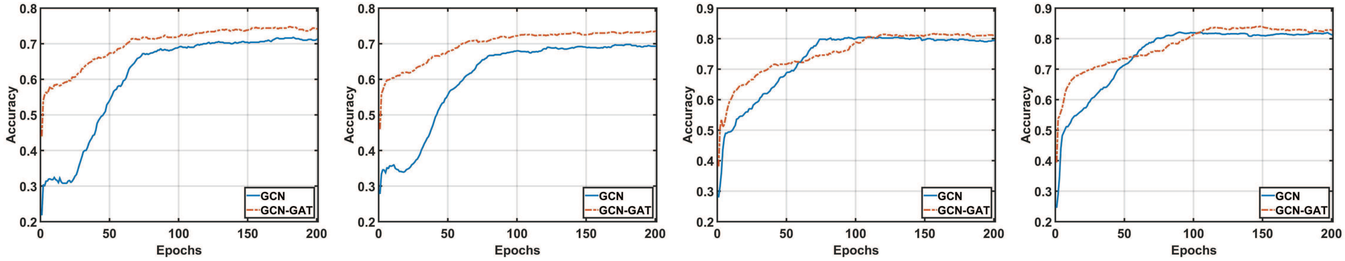
5.4.1 Training Process

By taking the basic version of graph adversarial training GCN-GAT as example, we then study the effect of GAT on the training process. Specifically, we observe the performance of GCN and GCN-GAT on the validation and testing of Citeseer and Cora, which is depicted in Figure 5. Note that we omit the performance on NELL, which shows the same trend, for saving space. As can be seen, 1) On the two datasets, the performance of both GCN and GCN-GAT becomes stable after 100 epochs, which indicates that GAT will not affect the convergence speed of GCN. 2) It is interesting to see that the performance of GCN-GAT



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

Fig. 4: Performance of GCN-GAT 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. 5: Training curves of the GCN and GCN-GAT on the validation and testing of Citeseer and Cora.

TABLE 6: Average training time per epoch of GCN, GCN-GAT, and GCN-GATV *w.r.t.* seconds. Slower means how many times the method is slower than GCN.

Method	Citeseer		Cora		NELL	
	Time	Slower	Time	Slower	Time	Slower
GCN	0.042	—	0.024	—	1.170	—
GCN-GAT	0.489	11.7x	0.126	5.2x	4.620	3.9x
GCN-GATV	0.800	19.1x	0.216	8.9x	5.725	4.9x

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

Method	Citeseer	Cora	NELL
GCN	-21.1%	-6.6%	-54.8%
GCN-GAT	-4.1%	-1.6%	-53.7%

increases faster than standard GCN during the initial several epochs. Considering that the supervised loss is typically much larger (about 1e5 times) than the value of graph adversarial regularizer in the initial epochs since all nodes are assigned predictions close to random leading to tiny divergence between connected nodes, the acceleration of performance increase is believed to be the effect data augmentation (additional adversarial examples) rather then the regularization.

5.4.2 Training Time

Here, we discuss the overhead of graph adversarial training via comparing the training time of GCN, GCN-GAT, and GCN-GATV, of which the average times of 50 epochs are summarized in Table 6. It should be noted that we conduct the experiment on a server equipped with two *Intel(R) Xeon(R) CPU E5-2620 V3*. We can see that adversarial training averagely decelerates the training of GCN 4.4 times on the NELL dataset, which is acceptable considering that each epoch still takes several seconds only. Besides, the additional computation on smaller datasets (*i.e.*, Citeseer and Cora) is negligible since all the methods are much faster.

TABLE 8: Average Kullback-Leibler divergence between connected node pairs calculated from predictions of GCN and GCN-GAT, of which small value indicates close predictions.

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

5.4.3 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 GCN-GAT on adversarial examples as compared to clean examples. As can be seen, by training GCN with GAT, the model becomes less sensitive to adversarial perturbations. For example, on the citation graphs, graph adversarial perturbations in the scale of 0.01 (*i.e.*, $\epsilon = 0.01$) decreases accuracy of GCN by 13.7%, while the number is only 2.7% for GCN-GAT. It justifies that the graph adversarial training technique could enhance the robustness of the GCN model.

5.4.4 Effect of GAT on Divergence of Neighbor Nodes

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

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 GAT on more graph neural network models [3], [4], [11]. Moreover, we are interested to investigate the effect of GAT 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 [46] and adversarial dropout [47] into the proposed method to further enhance its ability of stabilizing graph neural network models.

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] R. Ying, J. You, C. Morris, X. Ren, W. L. Hamilton, and J. Leskovec, "Hierarchical graph representation learning with differentiable pooling," *arXiv preprint arXiv:1806.08804*, 2018.

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

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

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

[17] 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.

[18] 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.

[19] 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.

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

[21] 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.

[22] 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.

[23] 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.

[24] 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.

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

[26] 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.

[27] 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.

[28] 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.

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

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

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

[32] 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.

[33] 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.

[34] L. Sang, M. Xu, S. Qian, and X. Wu, "Aaane: Attention-based adversarial autoencoder for multi-scale network embedding," *AAAI*, 2018.

[35] 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.

[36] 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.

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

[38] J. M. Joyce, "Kullback-leibler divergence," *Alphascript Publishing*, p. 844, 2013.

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

[40] 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.

[41] 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.

[42] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[43] 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.

[44] 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.

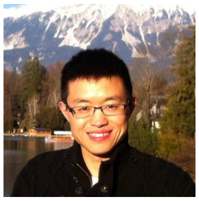
[45] P. Cui, X. Wang, J. Pei, and W. Zhu, "A survey on network embedding," *IEEE Transactions on Knowledge and Data Engineering*, 2018.

[46] 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.

[47] S. Park, J.-K. Park, S.-J. Shin, and I.-C. Moon, "Adversarial dropout for supervised and semi-supervised learning," *AAAI*, 2018.

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 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.