Neighbor-Anchoring Adversarial Graph Neural Networks

Zemin Liu, Yuan Fang, Yong Liu, Vincent W. Zheng

Abstract—Graph neural networks (GNNs) have witnessed widespread adoption due to their ability to learn superior representations for graph data. While GNNs exhibit strong discriminative power, they often fall short of learning the underlying node distribution for increased robustness. To deal with this, inspired by generative adversarial networks (GANs), we investigate the problem of adversarial learning on graph neural networks, and propose a novel framework named NAGNN (*i.e.*, Neighbor-anchoring Adversarial Graph Neural Networks) for graph representation learning, which trains not only a discriminator but also a generator that compete with each other. In particular, we propose a novel neighbor-anchoring strategy, where the generator produces samples with explicit features and neighborhood structures anchored on a reference real node, so that the discriminator can perform neighborhood aggregation on the fake samples to learn superior representation. The advantage of our neighbor-anchoring strategy can be demonstrated both theoretically and empirically. Furthermore, as a by-product, our generator can synthesize realistic-looking features, enabling potential applications such as automatic content summarization. Finally, we conduct extensive experiments on four public benchmark datasets, and achieve promising results under both quantitative and qualitative evaluations.

Index Terms—Neighbor-anchoring, generative adversarial network, graph neural network.

1 Introduction

Graph structures are widespread in real-world scenarios, ranging from social networks to biological networks. Many advanced analytics on graphs, such as personalized recommendation on social networks [1] and disease gene identification on biological networks [2], can be cast as instances of node classification and link prediction. Thus, addressing these problems heavily rely on effective representations for graphs. While traditional approaches often involve significant manual feature engineering, graph neural networks [3], [4] have emerged as a promising family of representation learning models for graphs.

Existing work. Graph neural networks (GNNs) aim to map the nodes of a graph into a low-dimensional space whilst preserving their structural information, to ultimately support graph-based applications. Recent state-of-the-art approaches, such as graph convolutional networks [3], graph attention networks [4] and GraphSAGE [5], employ a similar key process of multi-layer *neighborhood aggregation*. In one layer, each node receives and aggregates messages (*i.e.*, node features or embeddings) from their neighboring nodes. Additional layers may be utilized so that the neighbors recursively receive messages from their neighbors. Such neighborhood aggregation effectively smooths the node messages along the graph structures [6], leading to powerful graph representations. In contrast, another line of graph representation learning methods known as network embedding, such as DeepWalk [7] and node2vec [8], generally predict context nodes drawn from random walk paths, to effectively preserve structural information on graphs. However, they lack the key neighborhood aggregation operator that unifies both node messages and graph structures in GNNs.

In the context of node classification, while graph neural networks can be powerful discriminative models by learning the node conditional class distributions, generative approaches can also be useful by learning the underlying node distributions conditioned on classes. To exploit the power of both discriminative and generative models, generative adversarial networks (GANs) [9], [10] have become popular in recent years. Specifically, GANs employ the adversarial principle involving both a discriminator and a generator. The two players contest with each other in a minimax game—the discriminator's objective is to distinguish "real" samples from the true distribution and "fake" samples from the generator, whereas the generator aims to fool the discriminator by producing fake samples that mimic the real ones. In particular, the generator produces complement samples in the low density regions [11], which can ultimately lead to more robust decision boundaries.

Present work. Inspired by GANs, we investigate the adversarial training of graph neural networks. Although there are many GAN-based approaches for graphs [12], [13], [14], [15], they seldom explore GANs and GNNs jointly in an end-to-end manner. In our approach, as shown in Fig. 1, GANs are utilized to generate fake nodes in the sparse region of the graph where there is a low-density of edges [15]; more specifically, the generated nodes tend to lie in the periphery of the dense regions under our proposed strategy, as we shall see. As such, by discriminating the real and fake nodes, the discriminator can learn a better decision boundary between classes. In other words, the fake

- *Z. Liu and Y. Fang are with the School of Computing and Information Systems, Singapore Management University, Singapore. E-mail: {zmliu,yfang}@smu.edu.sg*
- *Y. Liu is with Joint NTU-UBC Research Centre of Excellence in Active Living for the Elderly (LILY), Nanyang Technological University, Singapore. E-mail: stephenliu@ntu.edu.sg*
- *V. Zheng is with WeBank, China. E-mail: vincentz@webank.com*

Manuscript received xxx; revised xxx.
(Corresponding author: Yuan Fang.)

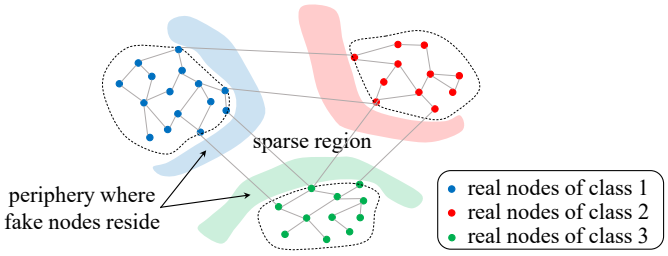


Fig. 1: Motivation of GANs on graph. Ideally, our generator would produce samples not only in the sparse regions with a low-density of links, but also in the periphery of the dense regions. The fake samples are complementary to enhance the robustness of node representations, enabling the discriminator to learn better decision boundaries.

nodes are complementary in nature to augment the real nodes, which enhance the robustness of node representations. However, training GNNs in an adversarial manner is not straightforward. In particular, two major challenges still remain with the design of the generator.

First, *what is the definition of a sample on a graph?* In non-graph data, an instance is simply its features, such as the pixels of an image. In contrast, on a graph, a node is characterized by not only its features, but also its structures, *i.e.*, its neighbors on the graph. Thus, the generator must account for both parts, which are necessary for the key operation of neighborhood aggregation in GNNs. However, in existing adversarial approaches on graphs such as GraphGAN [12] and Graph$\mathcal{S}$GAN [11], [15], a sample is only defined as its latent embedding, lacking explicit features and structures to perform neighborhood aggregation.

Second, *how do we produce good samples?* Previous approaches typically produce samples from a prior noise distribution freely [13], [15], [16], [17], without explicit constraints between the real and fake samples. In other words, they entirely depend on the model to learn any relationship between the real and fake samples, which may be hard to be reliably learned. On the contrary, if each fake sample is generated w.r.t. a reference real node, the generator and discriminator could become more conscious of each other's output. Specifically, given a pair of fake sample and its reference node, the discriminator can better detect their finer differences. This would in turn guide the generator to produce a more realistic sample for a given reference node, such that the fake samples not only "fill" in the sparse regions between classes [11], [15], but also concentrate on the periphery of the dense regions of each class, as shown in Fig. 1. As a result, the discriminator is able to learn better decision boundaries for the classes.

To address the above challenges, we propose a novel *neighbor-anchoring* strategy. In this strategy, the generator produces samples with explicit features and structures, so that the discriminator can perform neighborhood aggregation on them. Furthermore, given a reference real node, the generator produce a sample anchored on the same set of neighbors as the reference node, *i.e.*, they have the same set of neighbors. At the same time, the generator utilizes a feature synthesizer, aiming to produce features that mimic the reference node (without knowing the actual features of the reference node). By sharing the same neighbors and thus isolating their effects on a fake sample and its reference node, the model can focus on discriminating and generating their features to achieve mutual improvement, and encourage the fake sample to reside closer to the real node at the periphery of dense regions. The advantage of our neighbor-anchoring strategy can be demonstrated both theoretically and empirically. Furthermore, as a by-product, our generator can synthesize realistic-looking features for any node on the graph which are interpretable—both real and synthetic features reside in the same space—enabling potential applications such as automatic content summarization. For instance, on a bibliographic network, given a paper node and its references (*i.e.*, neighbors on the graph), we can automatically generate a bag-of-words feature vector to summarize the paper.

Contributions. In summary, we propose a new framework of Neighbor-anchoring Adversarial Graph Neural Networks (**NAGNN**), to learn more robust end-to-end node representations on a graph. We make the following contributions.

- We propose a novel neighbor-anchoring strategy for generating adversarial node samples on a graph.
- We realize the neighbor-anchoring principle with a model NAGNN, which performs end-to-end adversarial training of graph neural networks, and further present a formal analysis on its error.
- We conduct extensive experiments on four public datasets, and in terms of accuracy on average outperform GCN and GAT by 3.3% and 1.7%, respectively. Furthermore, our generator also serves as a feature synthesizer, for which we conduct a qualitative study.

2 Related Work

Graph representation learning. Network embedding [7], [8], [18], [19], [20], [21] has been extensively studied given its strong performance on downstream tasks such as node classification and link prediction. They aim to learn low-dimensional and structure-preserving embeddings for nodes on a graph, typically in an unsupervised manner. Most of these approaches exploit graph structures through random walk sampling [7], [8] and various orders of proximity [18].

More recently, graph neural networks (GNNs) [22] have emerged as a promising direction for end-to-end graph representation learning in a semi-supervised setting. Hinged on the core idea of recursive neighborhood aggregation, various approaches have been proposed. For instance, GraphSAGE [5] designs several aggregation functions to pool neighborhood information. Others try to differentiate the importance of different neighbors through neural attention mechanisms [4], [23] and adaptive receptive paths [24]. Different forms of graph convolution have also been considered, such as accounting for higher-order or hierarchical graph structures [25], [26], performing disentangled convolution according to latent factors [27], and conducting node-wise localization [28]. Additionally, structure-aware approaches [29], [30] focus on distinguishing different graph structures for more expressive aggregation.

GANs. Since their inception, GANs [9], [10] have achieved considerable success in many research fields including computer vision [31], [32], natural language processing [33], [34] and data mining [35], [36]. Generally, a generator is designed to learn the underlying data distribution, so as to produce fake samples that resemble the real ones. While the discriminator is tasked to differentiating the real and fake samples, resulting in improved discriminating power. This in turn drives the generator to come up with better samples. When the data is sparse or noisy, the generator becomes especially crucial to ensure the robustness of the model. GANs have also been extended to deal with supervised learning [11], [37], where each real sample belong to one of the K classes. In this setting, an extra class is considered in the discriminator to house the fake samples from the generator, resulting in a total of $K + 1$ classes.

GANs on graph data. Inspired by the strong perform of GANs, an increasing number of studies [12], [13], [14], [15], [16], [17], [38], [39] leverage the generative adversarial principal to learn more robust node representations on graphs. Most of them focus on the network embedding setting [12], [13], [14], [40]. Others consider variants of network embedding, such as heterogeneous network embedding [38] and multi-view network embedding [39]. Nevertheless, these network embedding models are typically unsupervised and often employ a direct embedding lookup for node representations. A few studies [16], [17] employ graph autoencoders to train GANs, which leverage various neighborhood aggregation such as the long short-term memory [16] and graph convolution [17]. However, like network embedding, they are also unsupervised to optimize the reconstruction error in the autoencoder. The work on Graph$\mathcal{S}$GAN [15] is the most related to ours, which adopts a GAN to learn semi-supervised node representations using a $K + 1$ class formulation. However, the discriminator only employs a multi-layer perceptron, rather than a graph neural network as characterized by the key operation of recursive neighborhood aggregation. Instead, to capture graph structures, it requires a separately trained network embedding model such as DeepWalk [7]. This also implies that Graph$\mathcal{S}$GAN is not truly end-to-end, as its optimization cannot influence network embedding through backpropagation.

On another line, there exist adversarial attack and defense models on graphs [41], [42]. However, their problem is different from ours: They aim to minimize the impact of adversarial noises added to the graph, in order to recover the performance of GNNs to a level comparable on a "clean" graph. In our problem, we aim to improve the graph representations, and do not assume that the graphs have been adversarially modified.

3 Preliminaries

In this section, we introduce the problem statement and a brief review of generative adversarial nets.

3.1 Problem Statement

In this paper, we work with *featured graphs*. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{X})$ denote a featured graph, where $\mathcal{V}$ is a set of nodes, $\mathcal{E}$ is a set of edges between the nodes, and $\mathbf{X} \in \mathbb{R}^{|\mathcal{V}|\times M}$ is a feature matrix for the nodes such that M is the number of features. Further let $\mathbf{x}_v \in \mathbb{R}^M$ denote the feature vector of node $v \in \mathcal{V}$, *i.e.*, the row of $\mathbf{X}$ corresponding to v.

On a featured graph $\mathcal{G}$, we investigate the problem of semi-supervised node classification. Specifically, each node belongs to one of the K classes $\{1, 2, \ldots, K\}$. However, we only observe the class labels of a subset of the nodes in $\mathcal{V}$. Let $\mathcal{L}$ be this subset of nodes with their observed labels, *i.e.*, $\forall(v, y) \in \mathcal{L}$, $v \in \mathcal{V}$ is a node with its observed label $1 \leq y \leq K$. Note that $|\mathcal{L}| < |\mathcal{V}|$. The problem is to classify the remaining nodes which do not have observed labels.

3.2 Generative Adversarial Networks

We employ a generative adversarial network (GAN) [9], [32] to address our classification problem. It can be regarded as a minimax game between two players, namely, a discriminator D and a generator G. Each player seeks to improve itself by competing with the other player iteratively. Specifically, the generator G aims to generate fake samples that resemble real nodes, and the discriminator aims to distinguish the real nodes from the fake samples. Thus, a better generator would force the discriminator to also become more effective, and vice versa.

In our classification setting, since each real node belongs to one of the K classes, we adopt the commonly used $K+1$ setting [15], [43]. That is, we consider $K+1$ classes given K original classes. The discriminator aims to not only classify a real node into the right class among the K original classes $\{1, 2, \ldots, K\}$, but also identify fake samples and classify them into the augmented class $K + 1$. Formally, GANs aim to optimize the following minimax game.

$$\begin{aligned}\max_{\theta_G}\min_{\theta_D} \quad & \mathbb{E}_{v,y\sim P_{\text{true}}}[-\log D(y|v;\theta_D)] \\ & + \mathbb{E}_{\hat{v}\sim P_{\theta_G}}[-\log D(K+1|\hat{v};\theta_D)],\end{aligned} \tag{1}$$

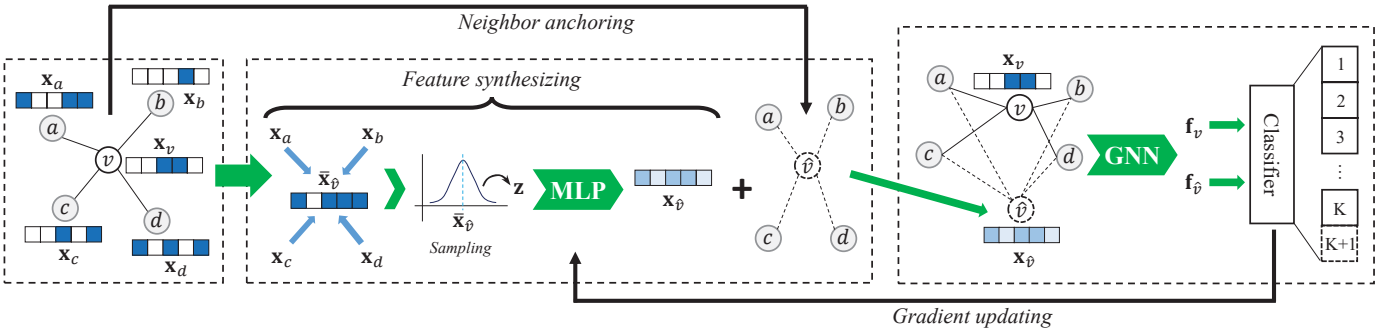
where θ_D and θ_G are the parameters of the discriminator and generator, respectively; P_{true} is the true joint distribution of nodes and labels, and P_{θ_G} is the distribution learned by the generator; $D(y|v)$ is the output of the discriminator to estimate the class distribution conditioned on a node. In this formulation, the discriminator attempts to minimize the cross entropy between the true and estimated class distributions (*i.e.*, to tell apart the original K classes as well as the fake samples), whereas the generator tries to maximize the cross entropy (*i.e.*, to fool the discriminator).

4 Proposed Model: NAGNN

In this section, we introduce the proposed model NAGNN. We start with an overview of NAGNN, followed by the two main components, namely, the generator G and the discriminator D. Finally, we present the overall algorithm.

4.1 Overview

The overall framework of NAGNN is illustrated in Fig. 2. As with typical graph-based models, for a given node, we are concerned with its neighbors on the graph. As shown in Fig. 2(a), consider node v's neighbors $\mathcal{N}_v = \{a, b, c, d\}$. Using this neighborhood as an example, Fig. 2(b) and (c)



(a) Neighborhood (b) Generator (c) Discriminator

Fig. 2: Overall framework of NAGNN. (a) An existing node v and its neighboring nodes in the graph. (b) The generator, which produces a fake sample $\hat{v}$ anchored on the neighbors of the real node v. (c) The discriminator, which utilizes a GNN to classify real nodes into the first K classes, and fake samples into class $K+1$.

illustrate the two main components, namely, the generator and the discriminator, respectively.

The role of the generator is to produce fake samples. Each fake sample $\hat{v}$ consists of a feature vector $\mathbf{x}_{\hat{v}}$, as well as a local structure or a set of neighbors $\mathcal{N}_{\hat{v}}$. The former is synthesized by a neural network, while the latter is based on a novel neighbor-anchoring strategy. On the other hand, the discriminator employs a GNN (*e.g.*, GCN or GAT) to learn node representations, so as to differentiate the fake sample $\hat{v}$ from the real nodes v based on their final representations $\mathbf{f}_{\hat{v}}$ and $\mathbf{f}_v$, respectively. Specifically, it aims to classify v into one of the original classes $\{1, 2, \ldots, K\}$ and $\hat{v}$ into the augmented class $K+1$.

4.2 Discriminator

Graph convolution. As shown in Fig. 2(c), our discriminator employs a GNN to learn node representations and predict their classes. The choice of GNN is flexible, since different GNNs mainly differ in the way of neighborhood aggregation. Here we use graph convolutional networks (GCN) [3], [5] to exemplify our discriminator, although it can be easily extended to others such as graph attention network (GAT) [4]. We will experiment with both GCN and GAT in our empirical study. A GCN consists of multiple layers of graph convolution, where in the layer $l \geq 1$, a neighborhood aggregation is performed on each node v to derive its intermediate representation $\mathbf{f}_v^{(l)} \in \mathbb{R}^{N^{(l)}}$:

$$\mathbf{f}_v^{(l)} = \text{ReLU}\left(\frac{1}{|\tilde{\mathcal{N}}_v|}\sum_{v' \in \tilde{\mathcal{N}}_v} \mathbf{W}^{(l)}\mathbf{f}_v^{(l-1)}\right), \tag{2}$$

where $N^{(l)}$ is the embedding size of the layer l representation, $\tilde{\mathcal{N}}_v = \mathcal{N}_v \cup \{v\}$ denotes v's neighbors with v itself included, and $\mathbf{W}^{(l)} \in \mathbb{R}^{N^{(l)} \times N^{(l-1)}}$ is a learnable embedding matrix in layer l. In particular, the initial representation is simply the input features, *i.e.*, $\mathbf{f}_v^{(0)} \triangleq \mathbf{x}_v$, and $N^{(0)} = M$, the number of input features. Moreover, we denote the final N-dimensional representation of an L-layer GCN as $\mathbf{f}_v \triangleq \mathbf{f}_v^{(L)}$, where $N = N^{(L)}$.

Loss function. Typically, the final representation leverages a softmax function for multi-class output [3], [4]. Thus, the discriminator is also parameterized by a classification matrix $\mathbf{W} \in \mathbb{R}^{(K+1)\times N}$ under the $K+1$ class setting discussed in Section 3, such that

$$D(y|v;\theta_D) = \frac{\exp\left(\mathbf{W}_y\mathbf{f}_v\right)}{\sum_{y'=1}^{K+1}\exp\left(\mathbf{W}_{y'}\mathbf{f}_v\right)}, \tag{3}$$

where $\mathbf{W}_y \in \mathbb{R}^N$ denotes the y-th row of $\mathbf{W}$, *i.e.*, the weight vector of class y. Specifically, $D(y|v;\theta_D)$ estimates the probability of class y given node v. Note that θ_D, the parameters of the discriminator, include the embedding matrix $\mathbf{W}^{(*)}$ in each layer of graph convolution as well as the classification matrix $\mathbf{W}$.

The goal of the discriminator is to classify the real nodes into the their corresponding classes $\{1, 2, \ldots, K\}$, and simultaneously put the fake samples into the augmented class $K+1$. Subsequently, the loss function for the discriminator can be formulated as follows:

$$\begin{aligned} &-\frac{1}{|\mathcal{L}|}\sum_{(v,y)\in\mathcal{L}}\log D(y|v;\theta_D) \\ &-\alpha\cdot\frac{1}{|\hat{\mathcal{V}}|}\sum_{\hat{v}\in\hat{\mathcal{V}}}\log D(K+1|\hat{v};\theta_D) + \lambda_D\left\|\theta_D\right\|_2^2. \end{aligned} \tag{4}$$

Here $(v, y) \in \mathcal{L}$ is a real node $v \in \mathcal{V}$ with its observed label y, whereas $\hat{\mathcal{V}}$ is the set of fake samples produced by the generator. Moreover, $\alpha > 0$ controls the importance of fake samples, and $\lambda_D > 0$ is the regularization parameter for the discriminator.

4.3 Generator

Typically, the generator draws samples from a preset noise distribution, which are further transformed by a neural network to generate realistic samples that mimic real samples. Its parameters receive gradient updates from the discriminator as it tries to fool the discriminator.

Neighbor anchoring. In our context, a fake sample $\hat{v}$ consists of dual parts: its feature vector $\mathbf{x}_{\hat{v}}$, and its structure or the set of neighbors $\mathcal{N}_{\hat{v}}$. While both parts can be entirely generated, we propose a novel form of neighbor-anchoring strategy. Specially, the generator produces a (fake) sample $\hat{v}$ w.r.t. a reference real node v on the graph. On the one hand, the sample $\hat{v}$'s feature vector $\mathbf{x}_{\hat{v}}$ is synthesized by a neural

network such as a multi-layer perceptron (MLP) as shown in Fig. 2(b). On the other hand, $\hat{v}$'s neighborhood is anchored on v's neighborhood, *i.e.*, $\mathcal{N}_{\hat{v}} = \mathcal{N}_v$. In other words, the sample $\hat{v}$ simply borrows its neighbors from the reference node v, as shown in Fig. 2(b), such that $\mathcal{N}_{\hat{v}} = \{a, b, c, d\}$ as well.

The advantages of neighbor anchoring are twofold. First, the generator and discriminator become more conscious of each other's output. By controlling v and $\hat{v}$ to share the same neighbors, the discriminator can perceive finer differences between the real and synthesized features, $\mathbf{x}_v$ and $\mathbf{x}_{\hat{v}}$. Meanwhile, by isolating the effect of neighbors, the generator can better leverage the feedback from the discriminator to synthesize more realistic features. Second, given the shared neighbors and similar features $\mathbf{x}_v$ and $\mathbf{x}_{\hat{v}}$ under a sufficiently powerful generator, a GNN tends to produce similar final representations $\mathbf{f}_v$ and $\mathbf{f}_{\hat{v}}$. This encourages the discriminator to learn a more refined boundary around the class of v due to the presence of a nearby fake sample $\hat{v}$, as illustrated in Fig. 1. This is largely consistent with previous theoretical findings [11], [15] in that an effective generator should not be a perfect generator that samples from the true distribution, but a complement generator that "fills" in the low-density regions between classes. In particular, the neighbor-anchoring strategy further requires that the fake samples reside near the periphery of dense regions, driving the discriminator to search for better decision boundaries. A formal analysis will be presented in Section 4.5.

Remark. Several recent studies investigate mutual information maximization [44], [45], [46] for graph representation learning in an unsupervised manner. Although they also exploit the relationship between each node and the context to improve graph representation learning, they have fundamental differences from our method. First, they maximize the mutual information between each node and its context (*e.g.*, the neighborhood), as an additional constraint such that the GNN-based encoder can train contextually relevant node representations. In contrast, our proposed NAGNN capitalizes on the neighboring nodes as the input of the generator to produce fake samples anchored on them. The anchoring strategy is more desirable than generating fake nodes without constraints, as the generator and discriminator could become more conscious of each other's output by focusing on the finer differences between a fake node and its reference node with an identical neighborhood. Second, they usually regard the graph-level representation as the context. For efficiency, the aggregation of several subsampled patch-level representations (*e.g.*, sampling a few nodes from the 1-hop and 2-hop neighbors, respectively) can be an alternative context on large graphs. In contrast, our NAGNN strictly resorts to the ego's neighborhood as a reference to achieve anchoring.

Feature synthesizing. With the neighbor-anchoring strategy, our generator parameterized by θ_G can be formulated as a function $G(v, \mathbf{z}; \theta_G)$, which outputs a fake sample $\hat{v}$ based on a reference node v and some noise vector $\mathbf{z}$. While the reference node v lends $\hat{v}$ its set of neighbors, the noise $\mathbf{z}$ is drawn from a predetermined multivariate distribution Z for synthesizing the feature vector $\mathbf{x}_{\hat{v}}$. We adopt a multivariate Gaussian distribution as Z:

$$Z \triangleq \text{Gaussian}(\bar{\mathbf{x}}_{\hat{v}}, \sigma^2\mathbf{I}), \tag{5}$$

where $\bar{\mathbf{x}}_{\hat{v}}$ represents the mean vector, and $\sigma^2\mathbf{I}$ is the covariance matrix for some choice of real-valued σ. The mean vector can simply be a zero vector, but we can also regarded it as a initial estimator of the synthesized feature vector $\mathbf{x}_{\hat{v}}$. In order to synthesize more realistic features, we derive the initial estimator from the features of $\hat{v}$'s neighbors (which are also the neighbors of the reference node v due to neighbor anchoring), based on the assumption that the features of a node is related to the features of its neighbours. Here we compute the mean feature vector, *i.e.*,

$$\bar{\mathbf{x}}_{\hat{v}} = \frac{1}{|\mathcal{N}_{\hat{v}}|} \sum_{v' \in \mathcal{N}_{\hat{v}}} \mathbf{x}_{v'}, \tag{6}$$

although other forms of aggregation such as max- or sum-pooling can also be adopted, which achieve similar empirical performance as mean-pooling. Subsequently, given some noise $\mathbf{z} \sim Z$, we employ an MLP to synthesize the feature vector of the sample $\hat{v}$, *i.e.*, $\mathbf{x}_{\hat{v}} = \text{MLP}(\mathbf{z})$, and the generator's parameters θ_G are in fact this MLP's parameters. To generate features in the same space as the real features, the input and output layer must have the same dimensions as the real features. The dimensions of the hidden layer is flexible (usually a small constant such as 64 or 128). Thus, the total number of parameters of the MLP is linear in the feature dimensions.

Loss function. In our neighbor-anchoring generator, while each sample $\hat{v}$ is generated w.r.t. a real node v, we further constrain the reference node v to be a labeled node from $\mathcal{L}$, to make it more challenging for the discriminator to correctly classify real labeled node. Moreover, while the generator is deemed successful when a fake sample $\hat{v}$ is classified by the discriminator into *any* of the K real classes, we further require the generator to fool the discriminator to the extent that $\hat{v}$ is classified into the *same* class of the reference node v. Thus, the loss of the generator can be defined as follows, which tries to make the discriminator believe that the sample $\hat{v}$ also belongs to the class of the reference node v.

$$-\frac{1}{|\mathcal{L}|} \sum_{(v,y) \in \mathcal{L}, \mathbf{z} \sim Z} \log D(y | G(v, \mathbf{z}; \theta_G); \theta_D) + \lambda_G \left\| \theta_G \right\|_2^2. \tag{7}$$

4.4 Overall Algorithm

We resort to an iterative strategy to train NAGNN, alternating between the discriminator and generator until the model converges. More precisely, in each iteration, we first fix θ_G to optimize θ_D to improve the discriminator using fake samples from the current generator, after which we fix θ_D to optimize θ_G so that the generator can produce better samples as judged by the current discriminator. The pseudocode of our training process is sketched in Algorithm 1.

In the following, we conduct a complexity analysis on the algorithm. The discriminator inherits its complexity from the GNN (*e.g.*, GCN). In each iteration of our adversarial training procedure, to perform L-layer recursive neighborhood aggregation for a given node, a total of $O(d^L)$ nodes must be processed, where d is the maximal node

Algorithm 1 Model training for NAGNN

Input: graph $\mathcal{G}$, labeled set $\mathcal{L}$, number of epochs n_D for the discriminator and n_G for the generator in each iteration, number of fake samples m_D for the discriminator and m_G for the generator.
Output: θ_D, θ_G.

```
initialize parameters θ^D, θ^G;
while not converged do
    for i = 1 to n_D do                                  ▷ train discriminator
        V̂ ← generate m_D fake nodes for each labeled node
        update θ_D with L and V̂ according to Equation (4)
    end for
    for i = 1 to n_G do                                  ▷ train generator
        generate m_G fake samples w.r.t. each labeled node
        evaluate the fake samples using the discriminator
        update θ_G according to Equation (7)
    end for
end while
return θ_D, θ_G.
```

degree in the graph. Thus, with m_D samples generated w.r.t. each reference node, the discriminator incurs a time complexity of up to $O(n_D \cdot m_D \cdot |\mathcal{V}| \cdot d^L)$ for n_D epochs. Next, we analyze the generator. The feature synthesizer can precompute the initial estimator for all reference nodes in $O(d \cdot |\mathcal{V}|)$ time. During training time, the generator produces one fake sample w.r.t. each reference node, and evaluate them using the discriminator, thus taking $O(n_G \cdot m_G \cdot |\mathcal{V}| \cdot d^L)$ time for n_G epochs. Thus, the overall time complexity for NAGNN is $O((n_G \cdot m_G + n_D \cdot m_D) \cdot |\mathcal{V}| \cdot d^L)$. Note that n_G, m_G, n_D, m_D are all small constants. While the term d^L may be prohibitive for large d and L, in practice a small number of layers L such as two or three is sufficient. Furthermore, neighborhood sampling [5], [47], [48] can also be performed to reduce the effective maximum degree d to a small constant, although these methods are beyond the scope of our discussion. Overall, our adversarial approach belongs to the same complexity class as vanilla GNNs.

4.5 Theoretical Analysis

We provide a theoretical analysis to substantiate the intuition behind our neighbor-anchoring generator.

Definitions and assumptions. The discriminator $D(y|v;\theta_D)$ is defined as the softmax output in Eq. (3), to estimate the class distribution given a node v under the $K+1$ formulation. The softmax output is parameterized by a classification matrix $\mathbf{W} \in \mathbb{R}^{(K+1)\times N}$. For brevity, we call $\mathbf{W}$ a classifier, which makes the following prediction on a node v based on its final representation $\mathbf{f}_v$:

$$h(v;\mathbf{W}) = \arg \max_{1 \le y \le K+1} (\mathbf{W}_y \mathbf{f}_v)\,. \tag{8}$$

Suppose the discriminator has high enough capacity such that the $K+1$ classes become pairwise linearly separable in the space of final representations. Subsequently, we can define the convergence conditions for the discriminator as the basis of our error analysis.

Definition 1 (Converged Classifier)**.** *A classifier* $\mathbf{W}$ *is said to be converged if both the following conditions hold:*

- $\forall (v,y) \in \mathcal{L}$, $\mathbf{W}_y \mathbf{f}_v > \max_{y' \neq y} \mathbf{W}_{y'} \mathbf{f}_v$;
- $\forall \hat{v} \in \hat{\mathcal{V}}$, $\max_{y \le K} \mathbf{W}_y \mathbf{f}_{\hat{v}} < \mathbf{W}_{K+1} \mathbf{f}_{\hat{v}}$. □

In Definition 1, the first condition requires that all labeled nodes are classified correctly, and the second condition implies that all fake samples are classified into the class $K+1$.

For the generator, to quantitatively differentiate the neighbor-anchoring generator from the non-anchoring counterpart, we introduce the following definition of ϵ-samples.

Definition 2 (ϵ-Sample)**.** *For some finite positive constant* ϵ*, a fake sample* $\hat{v}$ *is called an* ϵ*-sample, if there exists a real labeled node* v *such that the* L_1 *distance between their final representations is bounded by* ϵ*, i.e.,* $\|\mathbf{f}_v - \mathbf{f}_{\hat{v}}\|_1 < \epsilon$. □

Definition 2 essentially requires ϵ-samples to be close to some labeled node in the final representation. In particular, the final representation of a node v is a function of v's input features $\mathbf{x}_v$, and the set of v's neighbors. In other words, $\mathbf{f}_v \triangleq \phi(\mathbf{x}_v, \mathcal{N}_v)$, where ϕ denotes some form of recursive neighbor aggregation, *i.e.*, a GNN. The function ϕ is a form of smoothing [49], in which each node v becomes similar to its neighbors $\mathcal{N}_v$, whilst also accounting for its own input features $\mathbf{x}_v$. Thus, to see if two nodes v and $\hat{v}$ have similar final representations, we can examine their neighbors $\mathcal{N}_v$ and $\mathcal{N}_{\hat{v}}$, as well as their input features $\mathbf{x}_v$ and $\mathbf{x}_{\hat{v}}$.

For our neighbor-anchoring generator, since each sample $\hat{v}$ replicates the neighbors of some labeled reference node, say v, we have $\mathcal{N}(\hat{v}) = \mathcal{N}(v)$. Furthermore, thanks to the universal approximation theorem [50], [51], the MLP-based generator is capable of recovering the features of $\hat{v}$ sufficiently well, *i.e.*, $\|\mathbf{x}_{\hat{v}} - \mathbf{x}_v\|$ can be arbitrarily small. Applying the aggregation function ϕ with its associated smoothing effect [6], [49], we have $\|\phi(\mathbf{x}_{\hat{v}}, \mathcal{N}_{\hat{v}}) - \phi(\mathbf{x}_v, \mathcal{N}_v)\| = \|\mathbf{f}_{\hat{v}} - \mathbf{f}_v\| < \epsilon$ for some finite ϵ. In other words, $\hat{v}$ is an ϵ-sample given the existence of v. This essentially imply that our generator produces the fake samples that lie close to the reference nodes, which tend to be in the periphery of the dense regions.

In contrast, without neighbor-anchoring, the neighbors of $\hat{v}$ are sampled from the underlying distribution given a noise prior, and thus there may not exist any labeled node with the same or a similar set of neighbors. That is, the difference between $\mathcal{N}(\hat{v})$ and $\mathcal{N}(v)$ for any given labeled node v can be arbitrarily large, and so does the difference between $\mathbf{x}_{\hat{v}}$ and $\mathbf{x}_v$. Therefore, $\hat{v}$ is not an ϵ-sample or ϵ can be arbitrarily large.

Error analysis. We give a bound on the generalization error [52], to show that the neighbor-anchoring generator tends to give a smaller error than a non-anchoring generator.

Consider a classifier $\mathbf{W}$. Its generalization error is the difference between its expected error $e(\mathbf{W})$ over the true distribution of data, and its empirical error $\tilde{e}(\mathbf{W})$ over the observed data. Specifically,

$$e(\mathbf{W}) = \mathbb{E}_{v,y \sim P(v,y)} \left[\mathbb{I}(y \neq h(v;\mathbf{W}))\right], \tag{9}$$

$$\tilde{e}(\mathbf{W}) = \frac{1}{|\mathcal{L}|} \sum_{(v,y) \in \mathcal{L}} \mathbb{I}(y \neq h(v;\mathbf{W})), \tag{10}$$

where $\mathbb{I}$ is a binary indicator function to model the 0-1 error.

We first derive a generalization bound in the presence of generic fake samples, without considering ϵ-samples for now.

Lemma 1 (Generalization Error)**.** *Let $\mathcal{U}$ denote the universe of converged classifiers. The generalization error of any classifier $\mathbf{W}$ is bounded, i.e., $e(\mathbf{W}) - \tilde{e}(\mathbf{W}) \leq B_{\mathcal{U}}(\mathbf{W})$ such that*

$$B_{\mathcal{U}}(\mathbf{W}) = \max_{\mathbf{U}\in\mathcal{U}} \sum_{y=1}^{K} \sum_{v\in\mathcal{V}} P(v,y)\mathbb{I}(h(v;\mathbf{W}) \neq h(v;\mathbf{U})). \quad (11)$$

Furthermore, without knowing the joint distribution $P(v,y)$, $B_{\mathcal{U}}(\mathbf{W})$ gives the tightest possible bound for the same $\mathcal{U}$. □

Proof. Under the assumption that the $K+1$ classes are pairwise linearly separable in the final representation space, there exists an oracle classifier $\mathbf{U}^* \in \mathcal{U}$ such that it gives zero expected error. Thus, given a node v, a classifier $\mathbf{W}$ makes an error on this node if $\mathbf{W}$ disagrees with U^*. Thus, the expected error is $e(\mathbf{W}) = \sum_{y=1}^{K}\sum_{v\in\mathcal{V}} P(v,y)\mathbb{I}(h(v;\mathbf{W}) \neq h(v;\mathbf{U}^*))$. Subsequently, $B_{\mathcal{U}}(\mathbf{W})$ in Equation (11) is an upper bound on $e(\mathbf{W})$. Since the empirical error $\tilde{e}(\mathbf{W})$ is bounded by zero from below, the generalization error is bounded by $B_{\mathcal{U}}(\mathbf{W})$ from above.

Furthermore, without knowing the full distribution $P(v,y)$, every classifier in $\mathcal{U}$ could be the oracle. When the oracle $\mathbf{U}^*$ happens to the maximizing classifier in Equation (11), $B_{\mathcal{U}}(\mathbf{W})$ is the exact error and thus also represents the tightest possible bound if $P(v,y)$ or $\mathbf{U}^*$ is unknown. □

One immediate consequence of Lemma 1 is that, the bound is monotonic w.r.t. the universe of converged classifiers $\mathcal{U}$, as formalized below.

Corollary 1 (Bound Monotonicity)**.** *Let $\mathcal{U}'$ be a subset of $\mathcal{U}$, i.e., $\mathcal{U}' \subseteq \mathcal{U}$. For any classifier $\mathbf{W}$, $B_{\mathcal{U}'}(\mathbf{W}) \leq B_{\mathcal{U}}(\mathbf{W})$.*

Finally, we investigate the error when the fake samples are constrained such that they are ϵ-samples.

Theorem 1 (Error with ϵ-Samples)**.** *Let $\mathcal{U}_\epsilon$ be the universe of converged classifiers when the generator produces ϵ-samples. For any classifier $\mathbf{W}$, it holds that $B_{\mathcal{U}_\epsilon}(\mathbf{W}) \leq B_{\mathcal{U}_{\epsilon'}}(\mathbf{W})$ if $\epsilon \leq \epsilon'$.*□

Proof. By Definition 2, for an ϵ-sample $\hat{v}$, there exist a real labeled node v such that $\|\mathbf{f}_v - \mathbf{f}_{\hat{v}}\|_1 < \epsilon$, and thus $\mathbf{U}_y\mathbf{f}_v - \mathbf{U}_y\mathbf{f}_{\hat{v}} < \epsilon\|\mathbf{U}_y\|_1$ for any $\mathbf{U}_y \in \mathbb{R}^{1\times N}$. Equivalently, $\mathbf{U}_y\mathbf{f}_v < \mathbf{U}_y\mathbf{f}_{\hat{v}} + \epsilon\|\mathbf{U}_y\|_1$ (*).

Next, suppose v is from class y, *i.e.*, $(v,y) \in \mathcal{L}$. By condition (ii) in Definition 1, a converged classifier $\mathbf{U}$ requires $\max_{y\leq K} \mathbf{U}_y\mathbf{f}_{\hat{v}} < \mathbf{U}_{K+1}\mathbf{f}_{\hat{v}}$, which implies $\mathbf{U}_y\mathbf{f}_{\hat{v}} < \mathbf{U}_{K+1}\mathbf{f}_{\hat{v}}$. Since having $K+1$ linear models for $K+1$ classes is a form of over-parameterization, $\mathbf{U}_{K+1}$ can be fixed as a zero vector [11], [43]. Thus, we have $\mathbf{U}_y\mathbf{f}_{\hat{v}} < 0$. Together with (*), we obtain $\mathbf{U}_y\mathbf{f}_v < \epsilon\|\mathbf{U}_y\|_1$.

Finally, jointly considering condition (i), a converged classifier $\mathbf{U}$ requires $\max_{y'\neq y} \mathbf{U}_{y'}\mathbf{f}_v < \mathbf{U}_y\mathbf{f}_v < \epsilon\|\mathbf{U}_y\|_1$, when the generator produces ϵ-samples. Note that $\epsilon\|\mathbf{U}_y\|_1 < \epsilon'\|\mathbf{U}_y\|_1$ if $\epsilon < \epsilon'$. That means, ϵ'-samples imply a less strict requirement on the converged classifiers, *i.e.*, $\mathcal{U}_\epsilon \subseteq \mathcal{U}_{\epsilon'}$. Thus, by Corollary 1, we have $B_{\mathcal{U}_\epsilon}(\mathbf{W}) \leq B_{\mathcal{U}_{\epsilon'}}(\mathbf{W})$ for any classifier $\mathbf{W}$, which concludes the proof. □

As discussed earlier, the neighbor-anchoring generator is assumed to produce ϵ-samples for some finite ϵ. On the other hand, without neighbor-anchoring, the fake samples are unconstrained, which can be regarded as ϵ'-samples for some arbitrarily large ϵ' such that $\epsilon' \geq \epsilon$. By Theorem 1, the neighbor-anchoring generator entails a better error bound than its non-anchoring counterpart.

TABLE 1: Summary of datasets.

Datasets	# Nodes	# Edges	# Classes	# Features
Cora	2,708	5,429	7	1,433
Citeseer	3,327	4,732	6	3,703
Pubmed	19,717	44,338	3	500
DBLP	1,866	7,153	4	1,084

5 EXPERIMENTS

In this section, we evaluate the performance of our proposed model NAGNN on four public datasets. Specifically, we compare our model with state-of-the-art baselines, study the neighbor-anchoring strategy, and investigate parameter sensitivity. Finally, we also present a case study to demonstrate the quality of our generator.

5.1 Experimental Setup

Datasets. We conducted experiments on four public real-world graph datasets, namely, Cora, Citeseer, Pubmed [53] and DBLP [54]. Each node in the graph is a document, and the links among documents represent the citation relation. Each document has a sparse bag-of-words feature vector, and belongs to one of the several classes based on its topic.

While no further processing was done on the three datasets Cora, Citeseer and Pubmed, we performed additional filtering on the DBLP dataset to focus on four research areas: artificial intelligence (AI), database (DB), data mining (DM) and information retrieval (IR). We identified major conferences in the four areas, and only retained their papers. In particular, AI includes "IJCAI", "NeurIPS", "AAAI", "ECAI", "ACL", "COLT", "ICML" and "ECML"; DB includes "ICDE", "VLDB", "SIGMOD", "PODS", "EDBT" and "ICDT"; DM includes "PKDD", "WSDM", "SDM", "ICDM", "WWW", "KDD", "PAKDD" and "CIKM"; IR includes "SIGIR", "TREC" and "ECIR". Based on its publication venue, each paper is assigned a research area as its class label. Additionally, to reduce potential noises, we removed words occurring in fewer than five papers and papers with fewer than five words.

The four datasets are summarized in Table 1. We employed all of them for multi-class node classification. For the case study on the generator, only DBLP was used, as it is the only dataset with explicit words as features, whereas the other datasets only contain word identifiers.

Baselines. We consider baselines from the following four main categories of graph representation learning. (1) Network embedding: *DeepWalk*; (2) Unsupervised GAN-based models: *GraphGAN*, *ARGA* and *ARVGA*; (3) Semi-supervised GAN-based models: *ARGA(S)*, *ARVGA(S)* and *GraphSGAN*; (4) End-to-end graph neural networks: *GCN* and *GAT*. The details of the baselines are introduced below.

- *DeepWalk* [7]: It is a network embedding approach, which samples truncated random walks on the graph, and applies the skip-gram model on the resulting node

TABLE 2: Node classification performance (in percent) with standard deviation using 20 labeled nodes per class, averaged over 10 runs. The best results are **bolded**. In the column of "Input data", $\mathbf{A}$ denotes the adjacency matrix, $\mathbf{X}$ denotes the feature matrix, and $\mathcal{L}$ denotes the labeled nodes.

Methods	Input data	Cora			Citeseer			Pubmed			DBLP		
		Accuracy	Micro-F	Macro-F	Accuracy	Micro-F	Macro-F	Accuracy	Micro-F	Macro-F	Accuracy	Micro-F	Macro-F
DeepWalk	$\mathbf{A}$	73.8±0.3	74.9±0.1	74.0±0.1	61.6±0.2	60.5±1.0	59.8±0.5	67.4±0.3	65.2±0.1	66.1±0.1	50.4±1.0	51.8±0.8	49.1±1.1
GraphGAN	$\mathbf{A}$	58.8±0.2	57.9±0.1	57.2±0.1	60.4±1.4	58.5±0.1	58.6±0.1	73.2±0.1	75.3±0.1	73.2±0.1	52.4±2.5	51.1±3.5	52.1±4.3
ARGA	$\mathbf{A}$, $\mathbf{X}$	58.2±0.5	48.8±0.8	39.7±0.7	48.7±1.3	47.4±2.1	44.5±2.3	53.8±1.3	46.5±2.7	41.4±3.5	56.4±1.3	55.1±1.1	55.2±1.2
ARVGA	$\mathbf{A}$, $\mathbf{X}$	46.4±1.9	35.6±2.3	26.8±1.4	64.4±0.3	64.0±0.2	56.0±0.3	40.7±0.2	20.1±0.4	19.3±0.3	25.0±0.2	17.1±1.2	23.7±0.5
ARGA(S)	$\mathbf{A}$, $\mathbf{X}$, $\mathcal{L}$	72.1±0.7	68.7±0.6	56.4±0.9	61.8±1.2	59.9±1.8	57.2±1.4	62.6±1.6	55.3±2.3	50.8±2.4	59.3±1.8	57.8±1.5	58.0±1.4
ARVGA(S)	$\mathbf{A}$, $\mathbf{X}$, $\mathcal{L}$	63.7±1.5	62.5±1.7	50.6±1.4	68.9±0.5	68.1±0.8	61.4±0.6	50.7±1.4	46.0±0.8	39.3±0.9	41.7±1.1	43.9±1.3	42.3±1.4
Graph$\mathcal{S}$GAN	$\mathbf{A}$, $\mathbf{X}$, $\mathcal{L}$	79.2±0.6	79.3±0.5	78.0±0.6	67.4±0.7	65.8±0.4	61.8±0.5	68.2±0.4	68.7±0.5	67.5±0.5	58.6±0.9	57.4±0.8	56.8±0.9
GCN	$\mathbf{A}$, $\mathbf{X}$, $\mathcal{L}$	81.5±0.7	80.8±0.5	80.4±0.6	70.4±0.5	68.3±0.7	66.9±0.4	78.9±0.3	78.8±0.4	78.0±0.3	61.7±1.5	62.2±1.2	60.9±0.7
NAGCN	$\mathbf{A}$, $\mathbf{X}$, $\mathcal{L}$	**83.2**±0.6	**81.7**±0.4	**81.9**±0.5	**72.8**±0.4	**70.7**±0.5	**69.0**±0.4	**79.0**±0.3	**79.4**±0.2	**78.4**±0.3	**66.4**±0.7	**65.4**±0.9	**64.6**±0.9
GAT	$\mathbf{A}$, $\mathbf{X}$, $\mathcal{L}$	82.9±0.6	82.0±0.6	81.8±0.6	72.4±0.7	70.4±0.8	68.2±0.7	77.2±0.5	77.7±0.7	76.6±0.5	68.6±3.1	64.1±4.3	57.2±7.2
NAGAT	$\mathbf{A}$, $\mathbf{X}$, $\mathcal{L}$	**83.5**±0.4	**82.6**±0.3	**82.5**±0.2	**72.9**±0.4	**70.9**±0.5	**68.3**±0.8	**77.7**±0.4	**77.8**±0.1	**77.0**±0.3	**71.8**±1.7	**69.1**±1.4	**68.6**±1.3

sequences for node embedding. We concatenate the output embeddings with the feature vectors as the final representation, since DeepWalk does not leverage node features in training.

- *GraphGAN* [12]: It is an unsupervised GAN-based approach, which learns more robust node embedding in a network. Similar to DeepWalk, we concatenate their output node embeddings with the feature vectors for further node classification.
- *ARGA* [17]: It is an unsupervised GAN-based model, which employs a graph autoencoder in the adversarial setting and uses a GCN as its encoder. Thus, both graph structures and node features have been considered in this method. It also has a variant *ARVGA*, which employs a variational graph autoencoder.
- *ARGA(S)* [17]: We further extend ARGA to the semi-supervised setting, by implementing a classification layer on the learned representation, in order to fully exploit the labeled data for better performance. Similarly, we extend ARVGA into a semi-supervised variant *ARVGA(S)* by also adding a classification layer.
- *Graph$\mathcal{S}$GAN* [15]: It performs semi-supervised node classification using the adversarial principle. Its generator generate fake, latent representations for nodes. Both its discriminator and generator are MLPs, and it requires a separate network embedding step to capture graph structures.
- *GCN* [3]: It is a GNN approach that utilizes graph convolutions to aggregate information from neighbours to form the representation of each node. It is trained end-to-end in a semi-supervised manner.
- *GAT* [4]: It is a more advanced variant of GCN, which exploits a self-attention mechanism to assign different weights to neighbours to reflect their relative importance.

We used the following settings for the baselines, which are derived based on our empirical tuning in line with recommended settings reported in the literature. For DeepWalk, we sampled 10 random walks per node, with walk length 100 and window size 5, and set the embedding dimension to 128. For GraphGAN, we use DeepWalk for pre-training its generator and discriminator. We set its embedding dimension to 128, batch size as 64, L2 regularization weight as 0.00001, learning rate as 0.001, and number of steps for the generator and discriminator in one iteration both as 30. We report the classification performance based on the node embeddings from its generator, since they achieve better empirical results than the discriminator's output. For ARGA and ARVGA, we set the learning rate as 0.001, number of epochs as 200, and constructed encoders with a 32-neuron hidden layer and a 16-neuron embedding layer. Two hidden layers with 16 and 64 neurons were used for the discrminator. For Graph$\mathcal{S}$GAN, we set the training batch size as 64, the number of training epochs as 100, learning rate as 0.003, and the scale factor between labeled and unlabeled nodes as 0.5. For GCN, we adopted two layers, such that the first (hidden) layer has a dimension of 8, while the second (output) layer follows a multi-class classifier. Besides, we set learning rate as 0.005, L2 regularization as 0.0005, and a dropout probability of 0.6. For GAT, we adopted similar settings as GCN, but with 8 attention heads in the first layer, each of which computes an 8-dimensional output. An exponential linear unit (ELU) is applied following each layer.

For unsupervised methods (DeepWalk, GraphGAN, ARGA and ARVGA), we further train a logistic regression classifier using their output embeddings as node features. We also tried support vector machines and softmax classification with cross entropy, which yielded similar or inferior results as logistic regression.

Our models. For the proposed NAGNN, we experiment with different GNNs as its discriminator. Specifically, we adopted GCN and GAT from the baselines. Thus, we call the two instances of our model *NAGCN* and *NAGAT*, using GCN and GAT as their discriminator, respectively. For our discriminators, we applied the same parameters employed in the corresponding baseline GCN or GAT. We used the Adam optimizer to train the model, with learning rate 0.005 for discriminator and 0.001 for generator, L2 regularization 0.005 for both the discriminator and generator. The generator employs an MLP with one hidden layer, and uses ELU as its activation function.

We pre-trained both the discriminators and generators. Specifically, the discriminators use the model of their cor-

responding baseline, GCN or GAT, to initialize the parameters, whereas the generator is pre-trained by reducing the error between the synthesized features and the actual features on the real nodes.

In each iteration of training, we chose the number of fake samples per labeled node as $m_D = 10$ for the discriminator and $m_G = 1$ for the generator. We further set the number of each epochs to $n_D = 20$ for the discriminator and $n_G = 10$ for the generator, which report stable empirical performance. Training the discriminator more than the generator is consistent with previous findings [38], [55]. We set $\sigma = 0.01$ for the prior Gaussian distribution in the generator, and $\alpha = 0.5$ for the weight of fake samples in the discriminator. We have also analyzed the sensitivity of these two parameters in Sect. 5.4.

Model evaluation. Three datasets Cora, Citeseer and Pubmed have standard splits into training, testing and validation sets. We split the DBLP data similarly. Specifically, on each dataset, the training set includes 20 nodes per class, the test set includes 1000 nodes and the validation set includes 500 nodes. We trained each model for 10 times, and report their average performance and the standard deviation. In particular, their performance was evaluated with three metrics: accuracy, micro-F and macro-F scores. Note that, under the standard definition micro-F is equivalent to accuracy in multi-class classification. To better evaluate datasets with imbalanced class distributions, when computing the micro-F score, we removed the largest class and only micro-averaged over the other smaller classes as another metric to complement accuracy and macro-F score.

Environment. We implement the model with Python 3.6.5 and Tensorflow 1.13.0. All experiments were run on a Linux server with an Intel Xeon W-2133 CPU and a NVIDIA GeForce RTX 2080Ti GPU.

5.2 Performance Evaluation

We compare the performance of our models NAGCN and NAGAT with the baselines. First, we evaluate the performance using 20 labeled nodes per class as training data, which is the standard split. Next, we reduce the number of labeled nodes to study the robustness of our model.

20 labeled nodes per class (standard split). We report the results in Table 2, and make the following observations.

Firstly, our models NAGCN and NAGAT, which adopts GCN and GAT as their respective discriminator, consistently outperforms the baselines on all datasets. Compared to the strongest baselines GCN and GAT, in terms of accuracy, NAGCN outperforms GCN by 3.3%, and NAGAT outperforms GAT by 1.7%, averaged across four datasets. Furthermore, our models not only improve accuracy consistently, but also reduce the standard deviation of their performances in most cases. This observation implies that our model is both more effective and robust, due to the adoption of adversarial training.

Secondly, our models perform much better than Graph$\mathcal{S}$GAN in all cases, which is the next best method after end-to-end neural networks. In particular, Graph$\mathcal{S}$GAN also trains the generator and discriminator using the $K + 1$ class formulation in a semi-supervised manner. However, it is not truly end-to-end, as it requires a separate network embedding model to capture the graph structure. Furthermore, it does not employ neighbor anchoring strategy and neighborhood aggregation as in our model.

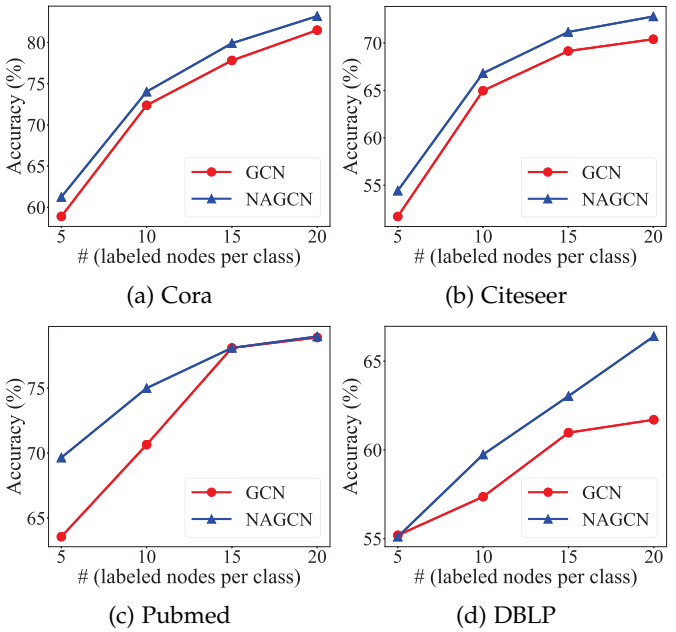


Fig. 3: Performance with fewer labeled nodes.

Thirdly, our models significantly outperform all other baselines by very large margins. As discussed earlier, DeepWalk and GraphGAN are based on unsupervised network embedding without neighborhood aggregation, and thus are limited (even if equipped with a GAN). On the other hand, ARGA and ARVGA are unsupervised methods employing adversarial regularized graph autoencoders, and ARGA(S) and ARVGA(S) are their semi-supervised extensions. While the latter perform better due to the exploitation of labeled data during embedding training, all of them employ graph convolutional layers to reduce the reconstruction error, rather than to discriminate real and fake samples for better decision boundaries.

Fewer labeled nodes. We vary the number of labeled node per class and report the comparison between NAGCN and GCN in Fig. 3. We only report the accuracy, as the micro-F and macro-F scores follow a similar trend. We observe that, in most cases, our model NAGCN's margin of improvement over GCN generally increases with fewer labeled nodes. For instance, averaging over four datasets, we improve over GCN by 3.3% with 20 labeled nodes per class, whereas the improvement grows to 3.8% with 10 labeled nodes per class. This demonstrates that our model is robust toward a small number of labeled nodes.

Training efficiency. In Fig. 4, we compare the training time of GNNs and our NAGNN counterparts. While our approach NAGNN requires more training time due to the iterative training procedure that alternates between the generator and GNN-based discriminator, our time cost is on the same order as the GNN counterparts, generally being slower by a constant factor. Overall, the time comparison shows that our approach belongs to the same compleixity

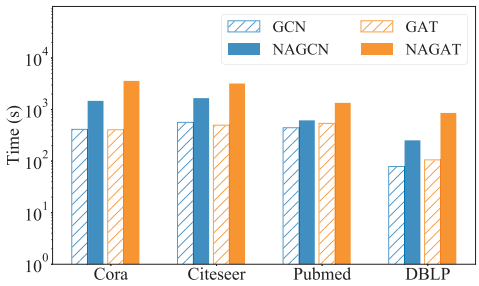


Fig. 4: Training time comparison.

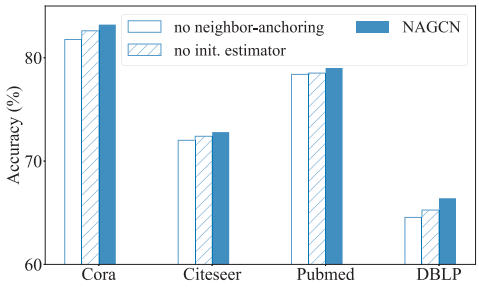


Fig. 5: Impact of our neighbor-anchoring strategy.

class as GNNs, consistent with the analysis in Section 4.4.

5.3 Model Analysis

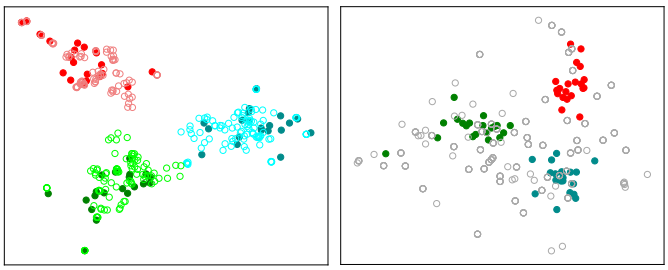
Ablation study. Next, we study the benefit of the proposed neighbor-anchoring strategy in Fig. 5, which compares NAGCN with two degenerate versions: (1) *no neighbor-anchoring*, where each fake sample is generated from a noise prior for both its features and neighbors, without anchoring its neighbors on any reference node; (2) *no initial estimator*, where the neighbor-anchoring strategy is adopted, but the feature synthesizer initializes from a zero-mean noise prior.

We observe that the model without neighbor-anchoring performs the worst, which implies that generating fake samples anchored on the neighbors of a real reference node is crucial. Moreover, without an initial estimator, the performance is also not optimal, indicating that a good initial estimator can help the generator to produce more realistic feature vectors.

Visualization. We further visualize the final representations learned by the no anchoring version and the full model in Fig. 6. As our discussion in earlier sections suggests, given neighbor-anchoring on reference nodes, in Fig. 6(a), a significant fraction of the fake samples not only residing in the low-density region, but also concentrating more on the periphery of the dense regions, enabling the discriminator to refine the decision boundary. In contrast, without neighbor-anchoring, in Fig. 6(b), the fake samples are more uniformly distributed across the entire space.

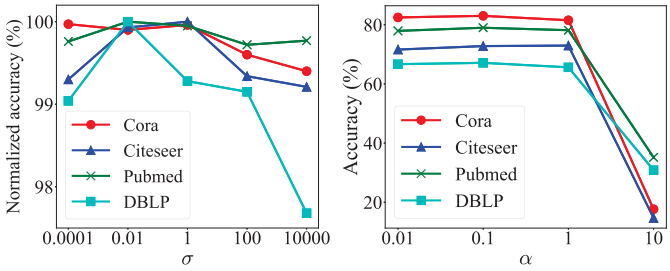
5.4 Parameters Sensitivity

We further investigate the impact of parameters on our model, using NAGCN as an example. The results on NAGAT are similar. Specifically, we study the Gaussian distribution parameter σ, and the weight of fake samples α.



(a) With neighbor-anchoring (b) No neighbor-anchoring

Fig. 6: Visualization of generated samples and real nodes. The solid dots denote real nodes, and different colors indicate different classes; the hollow circles denote fake samples. In (a), the fake samples are drawn in the same color as their reference nodes; in (b), the fake samples are not anchored on any real nodes and are all drawn in grey.



(a) Impact of σ (b) Impact of α.

Fig. 7: Impact of parameters.

Impact of σ. Fig. 7a presents the performance changes when the standard deviation of the prior σ is varied between 0.0001 and 10000. Note that, we normalize the accuracy values over the best value in each dataset so that the performance on different datasets can be plotted together. We can see that when σ is between 0.01 and 1, the optimal performance can be generally achieved. The reason is that, the non-zero values in the initial estimator of the feature synthesizer are generally in the range $[0.1, 1.0]$ with mean-pooling of the neighborhood. As a result, if σ is too large, the initial estimator is completed dominated by the noise, resulting in degraded performance. On the other hand, , this could alter the initialization feature vector too much, such that the important initialization feature information could be override. On the contrary, if σ is too small, the prior drawn from the prior has little variance and thus limit the expressiveness of the generator.

Impact of α. Fig. 7b shows the performance changes when the weight of the fake samples in the discriminator α is varied between 0.01 and 10. We can observe that, when $\alpha \in [0.01, 1.0]$, the performance is relatively stable, with $\alpha = 0.1$ being the most optimal. However, when α is too large, the performance drops quickly. This indicates that, there should be a balance between the weight of real samples and fake samples.

5.5 Case study

We conduct a case study on the DBLP dataset, to evaluate the ability of the generator to synthesize features. Note that

TABLE 3: Qualitative evaluation of the synthesized features from our generator on the DBLP dataset.

DOC	GROUND TRUTH TERMS	TERMS FROM NEIGHBOURS (BASELINE)	TERMS FROM OUR GENERATOR
1	models, language, classification, text, combining, bayes, naive	**models**, combining, positive, **bayes**, reduction, **naive**, **language**	combining, **models**, **classification**, **text**, **language**, email, **naive**
2	models, database, local, sharing, meta	**models**, **sharing**, classifiers, integrating, mining	**database**, **models**, mining, **databases**, **local**
3	problems, decision, constraint, utility, regret	**decision**, **regret**, **utility**, incremental, criterion	**regret**, **utility**, **decision**, incremental, **constraint**
4	networks, structure, local, bayesian, compiling	approximation, **local**, practical, propagation, **structure**	**networks**, **local**, **bayesian**, propagation, **structure**

DBLP is the only dataset with actual words as features, rather than only a word identifier in the other datasets.

In particular, for each node (document) in DBLP, we first obtain its actual word terms as the ground truth of comparison. Next, as a baseline, we directly aggregate feature vectors from its neighbours by taking their average, and keep the same number of terms as the ground truth by taking the top ranking features with the largest values. Finally, we employ our generator to synthesize a fake feature vector, and also only keep the same number of terms by taking the top ranking features.

Table 3 shows four documents and their terms using different methods, where each row represents one document. In the last two columns, terms in bold are also found in the ground truth. From the table we can make two observations.

Firstly, terms from the generator are generally closer to the ground truth terms than terms aggregated from its neighbours are. This is because the generator takes the feature vector aggregated from its neighbours as its initialization, which will be improved by adversarial training with the discriminator. Thus, after adversarial training, it could generate better fake samples from its initialization.

Secondly, terms from the generator is not perfect compared to the ground truth. This is in fact a crucial point of the generator, as the role of a generator is to produce complementary samples for the discriminator to differentiate [11], [15], which must not be indistinguishable from the real ones.

6 CONCLUSION

In this paper, we investigated the problem of adversarial learning with graph neural networks, and proposed a novel framework NAGNN for end-to-end graph representation learning. In particular, we proposed a novel neighbor-anchoring strategy, in which the generator produces fake samples with explicit features, based on structures anchored on the neighbors of a reference real node. Subsequently, the GNN-based discriminator can perform recursive neighborhood aggregation on the fake samples to learn powerful representations. The neighbor-anchoring strategy ensures that the fake samples lie in the periphery of the dense regions, which facilitates the learning of a more refined decision boundary. Besides, the generator could enable potential applications such as automatic content summarization by synthesizing realistic features. Finally, our experiments demonstrated promising results of NAGNN, both quantitatively and qualitatively.

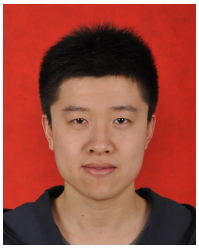
ACKNOWLEDGMENTS

This research is supported by the Agency for Science, Technology and Research (A*STAR) under its AME Programmatic Funds (Grant No. A20H6b0151).

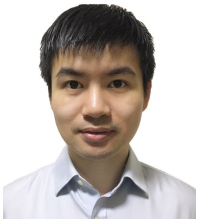
REFERENCES

[1] T. C. Zhou, H. Ma, M. R. Lyu, and I. King, "Userrec: A user recommendation framework in social tagging systems," in *AAAI*, 2010, pp. 1486–1491.
[2] S. K. Ata, Y. Fang, M. Wu, X.-L. Li, and X. Xiao, "Disease gene classification with metagraph representations," *Methods*, vol. 131, pp. 83–92, 2017.
[3] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *ICLR*, 2017.
[4] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," in *ICLR*, 2018.
[5] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *NeurIPS*, 2017, pp. 1024–1034.
[6] Q. Li, Z. Han, and X.-M. Wu, "Deeper insights into graph convolutional networks for semi-supervised learning," in *AAAI*, 2018, pp. 3538–3545.
[7] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *KDD*, 2014, pp. 701–710.
[8] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *KDD*, 2016, pp. 855–864.
[9] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *NeurIPS*, 2014, pp. 2672–2680.
[10] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," in *ICLR*, 2016.
[11] Z. Dai, Z. Yang, F. Yang, W. W. Cohen, and R. R. Salakhutdinov, "Good semi-supervised learning that requires a bad GAN," in *NeurIPS*, 2017, pp. 6510–6520.
[12] 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," in *AAAI*, 2018, pp. 2508–2515.
[13] Q. Dai, Q. Li, J. Tang, and D. Wang, "Adversarial network embedding," in *AAAI*, 2018, pp. 2167–2174.
[14] H. Gao, J. Pei, and H. Huang, "ProGAN: Network embedding via proximity generative adversarial network," in *KDD*, 2019, pp. 1308–1316.
[15] M. Ding, J. Tang, and J. Zhang, "Semi-supervised learning on graphs with generative adversarial nets," in *CIKM*, 2018, pp. 913–922.
[16] 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 *KDD*, 2018, pp. 2663–2671.
[17] 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.
[18] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "Line: Large-scale information network embedding," in *WWW*, 2015, pp. 1067–1077.
[19] D. Wang, P. Cui, and W. Zhu, "Structural deep network embedding," in *KDD*, 2016, pp. 1225–1234.

[20] S. Cao, W. Lu, and Q. Xu, "Grarep: Learning graph representations with global structural information," in *CIKM*, 2015, pp. 891–900.
[21] Z. Liu, V. W. Zheng, Z. Zhao, F. Zhu, K. C.-C. Chang, M. Wu, and J. Ying, "Semantic proximity search on heterogeneous graph by proximity embedding," in *AAAI*, 2017, pp. 154–160.
[22] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A comprehensive survey on graph neural networks," *arXiv preprint arXiv:1901.00596*, 2019.
[23] Y. Yang, X. Wang, M. Song, J. Yuan, and D. Tao, "SPAGAN: Shortest path graph attention network," in *IJCAI*, 2019, pp. 4099–4105.
[24] Z. Liu, C. Chen, L. Li, J. Zhou, X. Li, L. Song, and Y. Qi, "Geniepath: Graph neural networks with adaptive receptive paths," in *AAAI*, vol. 33, 2019, pp. 4424–4431.
[25] C. Morris, M. Ritzert, M. Fey, W. L. Hamilton, J. E. Lenssen, G. Rattan, and M. Grohe, "Weisfeiler and Leman go neural: Higher-order graph neural networks," in *AAAI*, 2019, pp. 4602–4609.
[26] Z. Ying, J. You, C. Morris, X. Ren, W. Hamilton, and J. Leskovec, "Hierarchical graph representation learning with differentiable pooling," in *NeurIPS*, 2018, pp. 4800–4810.
[27] J. Ma, P. Cui, K. Kuang, X. Wang, and W. Zhu, "Disentangled graph convolutional networks," in *ICML*, 2019, pp. 4212–4221.
[28] Z. Liu, Y. Fang, C. Liu, and S. C. Hoi, "Node-wise localization of graph neural networks," in *IJCAI*, 2021.
[29] J. You, R. Ying, and J. Leskovec, "Position-aware graph neural networks," in *ICML*, 2019, pp. 7134–7143.
[30] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" *ICLR*, 2019.
[31] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," in *CVPR*, 2017, pp. 1125–1134.
[32] S. Reed, Z. Akata, X. Yan, L. Logeswaran, B. Schiele, and H. Lee, "Generative adversarial text to image synthesis," in *ICML*, 2016, pp. 1060–1069.
[33] Y. Zhang, Z. Gan, K. Fan, Z. Chen, R. Henao, D. Shen, and L. Carin, "Adversarial feature matching for text generation," in *ICML*, 2017, pp. 4006–4015.
[34] J. Guo, S. Lu, H. Cai, W. Zhang, Y. Yu, and J. Wang, "Long text generation via adversarial training with leaked information," in *AAAI*, 2018, pp. 5141–5148.
[35] L. Yu, W. Zhang, J. Wang, and Y. Yu, "SeqGAN: Sequence generative adversarial nets with policy gradient." in *AAAI*, 2017, pp. 2852–2858.
[36] J. Wang, L. Yu, W. Zhang, Y. Gong, Y. Xu, B. Wang, P. Zhang, and D. Zhang, "IRGAN: A minimax game for unifying generative and discriminative information retrieval models," in *SIGIR*, 2017, pp. 515–524.
[37] A. Odena, "Semi-supervised learning with generative adversarial networks," *arXiv preprint arXiv:1606.01583*, 2016.
[38] B. Hu, Y. Fang, and C. Shi, "Adversarial learning on heterogeneous information networks," in *KDD*, 2019, pp. 120–129.
[39] Y. Sun, S. Wang, T.-Y. Hsieh, X. Tang, and V. Honavar12, "MEGAN: A generative adversarial network for multi-view network embedding," in *IJCAI*, 2019, pp. 3527–3533.
[40] Q. Dai, X. Shen, L. Zhang, Q. Li, and D. Wang, "Adversarial training methods for network embedding," in *TheWebConf*, 2019, pp. 329–339.
[41] D. Zhu, Z. Zhang, P. Cui, and W. Zhu, "Robust graph convolutional networks against adversarial attacks," in *KDD*, 2019, pp. 1399–1407.
[42] H. Wu, C. Wang, Y. Tyshetskiy, A. Docherty, K. Lu, and L. Zhu, "Adversarial examples for graph data: deep insights into attack and defense," in *IJCAI*, 2019, pp. 4816–4823.
[43] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training GANs," in *NeurIPS*, 2016, pp. 2234–2242.
[44] P. Velickovic, W. Fedus, W. L. Hamilton, P. Liò, Y. Bengio, and R. D. Hjelm, "Deep graph infomax." in *ICLR*, 2019.
[45] J. Cao, X. Lin, S. Guo, L. Liu, T. Liu, and B. Wang, "Bipartite graph embedding via mutual information maximization," in *WSDM*, 2021, pp. 635–643.
[46] P. Wang, Y. Fu, Y. Zhou, K. Liu, X. Li, and K. Hua, "Exploiting mutual information for substructure-aware graph representation learning," in *IJCAI*, 2020, pp. 3415–3421.
[47] J. Chen, T. Ma, and C. Xiao, "Fastgcn: fast learning with graph convolutional networks via importance sampling," *ICLR*, 2018.
[48] W. Huang, T. Zhang, Y. Rong, and J. Huang, "Adaptive sampling towards fast graph representation learning," in *NeurIPS*, 2018, pp. 4558–4567.
[49] K. Xu, C. Li, Y. Tian, T. Sonobe, K.-i. Kawarabayashi, and S. Jegelka, "Representation learning on graphs with jumping knowledge networks," in *ICML*, 2018, pp. 5453–5462.
[50] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural networks*, vol. 4, no. 2, pp. 251–257, 1991.
[51] K. Hornik, M. Stinchcombe, H. White *et al.*, "Multilayer feedforward networks are universal approximators." *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.
[52] Y. S. Abu-Mostafa, M. Magdon-Ismail, and H.-T. Lin, *Learning from Data*. AMLBook Press, 2012.
[53] 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, pp. 93–93, 2008.
[54] Y. Fang, K. C.-C. Chang, and H. W. Lauw, "Roundtriprank: Graph-based proximity with importance and specificity," in *ICDE*, 2013, pp. 613–624.
[55] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein generative adversarial networks," in *ICML*, 2017, pp. 214–223.



Zemin Liu received his Ph.D. degree in Computer Science from Zhejiang University, Hangzhou, China in 2018, and B.S. Degree in Software Engineering in Shandong University, Jinan, China in 2012. He is currently a Research Scientist in the School of Computing and Information Systems, Singapore Management University, Singapore. His research interests include graph-based machine learning.



Yuan Fang received his Ph.D. degree in Computer Science from the University of Illinois at Urbana-Champaign, United States in 2014, and Bachelor's degree in Computer Science from National University of Singapore, Singapore in 2009. He is currently an Assistant Professor at the School of Computing and Information Systems, Singapore Management University, Singapore. Previously, he was a scientist at the Institute for Infocomm Research, Agency for Science, Technology and Research (A*STAR), Singapore and a data scientist at DBS Bank, Singapore. His work has been featured in the Best Papers collection of VLDB 2013. His current research focuses on graph-based machine learning, Web and social media mining, recommendation systems and bioinformatics.

Yong Liu is a Research Scientist at Joint NTU-UBC Research Centre of Excellence in Active Living for the Elderly (LILY), Nanyang Technological University, Singapore. Before that, he was a Data Scientist at NTUC Enterprise, Singapore from November 2017 to July 2018, and a Research Scientist at Data Analytics Department, Institute for Infocomm Research (I2R), A*STAR, Singapore from November 2015 to October 2017. He received his Ph.D. from the School of Computer Science and Engineering at Nanyang Technological University in 2016 and B.S. from the Department of Electronic Science and Technology at University of Science and Technology of China in 2008. His current research focuses on recommender systems and bioinformatics.

Vincent W. Zheng received his Ph.D. degree in Computer Science from Hong Kong University of Science and Technology in 2011. He is a senior tech lead in WeBank, China. Previously, he was a senior research scientist at the Advanced Digital Sciences Center, Singapore, and a research affiliate at the University of Illinois at Urbana-Champaign. His research interests include graph mining, information extraction, ubiquitous computing and machine learning.