Adversarial Directed Graph Embedding

Shijie Zhu[1,2], Hao Peng[1,2], Jianxin Li[1,2], Senzhang Wang[3], Lifang He[4]

[1]Beijing Advanced Innovation Center for Big Data and Brain Computing, Beihang University, Beijing 100191, China.
[2]State Key Laboratory of Software Development Environment, Beihang University, Beijing 100191, China.
[3]College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 211106, China.
[4]Department of Computer Science and Engineering, Lehigh University, Bethlehem, PA, USA.
{zhusj,penghao,lijx}@act.buaa.edu.cn, szwang@nuaa.edu.cn, lih319@lehigh.edu

Abstract

Node representation learning for directed graphs is critically important to facilitate many graph mining tasks. To capture the directed edges between nodes, existing methods mostly learn two embedding vectors for each node, source vector and target vector. However, these methods learn the source and target vectors separately. For the node with very low indegree or outdegree, the corresponding target vector or source vector cannot be effectively learned. In this paper, we propose a novel Directed Graph embedding framework based on Generative Adversarial Network, called DGGAN. The main idea is to use adversarial mechanisms to deploy a discriminator and two generators that jointly learn each node's source and target vectors. For a given node, the two generators are trained to generate its fake target and source neighbor nodes from the same underlying distribution, and the discriminator aims to distinguish whether a neighbor node is real or fake. The two generators are formulated into a unified framework and could mutually reinforce each other to learn more robust source and target vectors. Extensive experiments show that DGGAN consistently and significantly outperforms existing state-of-the-art methods across multiple graph mining tasks on directed graphs.

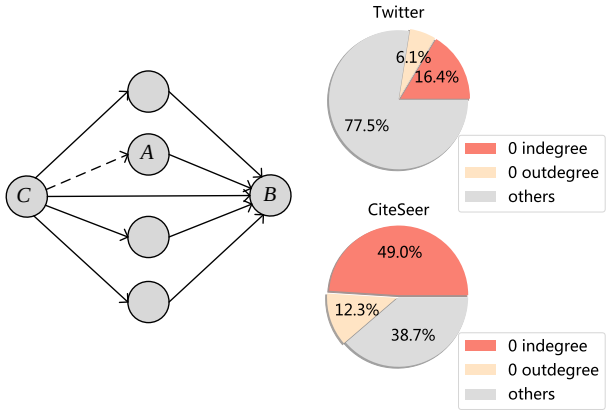


Figure 1: The left figure is a toy example of directed graph. Considering the edge from A to B, B is the target neighbor of A and A is the source neighbor of B. The right two figures are statistics from the social network of Twitter (Cha et al. 2010) and the citation network of CiteSeer (Bollacker, Lawrence, and Giles 1998), respectively.

Introduction

Graph embedding aims to learn a low-dimensional vector representation of each node in a graph, and has gained increasing research attention recently due to its wide and practical applications, such as link prediction (Liben-Nowell and Kleinberg 2007), graph reconstruction (Tsitsulin et al. 2018), node recommendation (Ying et al. 2018), and node classification (Bhagat, Cormode, and Muthukrishnan 2011).

Most existing methods, such as DeepWalk (Perozzi, Al-Rfou, and Skiena 2014), node2vec (Grover and Leskovec 2016), LINE (Tang et al. 2015), and GraphGAN (Wang et al. 2018), are designed to handle undirected graphs, but ignore the directions of the edges. While the directionality often contains important asymmetric semantic information in directed graphs such as social networks, citation networks and webpage networks. To preserve the directionality of the edges, some recent works try to use two node embedding spaces to represent the source role and the target role of the nodes, one corresponding to the outgoing direction and one for the incoming direction.

We argue that there are two major limitations for existing directed graph embedding methods: (1) Methods like HOPE (Ou et al. 2016) rely on strict proximity measures like Katz (Katz 1953) and low rank assumption of the graph. Thus, they are difficult to be generalized to different types of graphs and tasks (Khosla et al. 2019). Moreover, HOPE is not scalable to large graphs as it requires the entire graph matrix as input and then adopts matrix factorization (Tsitsulin et al. 2018). (2) Existing shallow methods focus on preserving the structure proximities but ignore the underlying distribution of the nodes. Methods like APP (Zhou et al. 2017) adopt directed random walk sampling technique which follows the outgoing direction to sample node pairs. These methods utilize negative sampling technique to randomly select existing nodes from the graph as negative samples. However, for nodes with only outgoing edges or only incoming edges, the target or source vectors cannot be effectively trained. Figure 1 presents a toy example. Although both of the nodes A and C have no incoming

Copyright © 2021, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

edges, it is more likely to exist a edge from C to A than the other way round. However, the proximities of the node pairs (A, C) and (C, A) predicted by APP are both zero, since the two node pairs are regarded as negative samples. The several proximity measurements introduced by HOPE like Katz (Katz 1953) all predict both proximities to be zero, as well. As shown in Figure 1, the nodes with 0 indegree or 0 outdegree (e.g., A and B) account for a large proportion of the graph. The directed graph embedding methods mentioned above treat the source and target roles of each node separately, which causes these methods not robust. However, a node's source role and target role are two types of properties of the node and are likely to be implicitly related. For instance, on social network like Twitter, fans who follow a star may be followed by other fans with common interests.

In this paper, we propose DGGAN, a novel Directed Graph embedding framework based on Generative Adversarial Network (GAN) (Goodfellow et al. 2014). Specifically, we train one discriminator and two generators which jointly generate the target and source neighborhoods for each node from the same underlying continuous distribution. Compared with existing methods, DGGAN generates fake nodes directly from a continuous distribution and is not sensitive to different graph structures. Furthermore, the two generators are formulated into a unified framework and could naturally benefit from each other for better generations. Under such framework, DGGAN could learn an effective target vector for the node A in Figure 1, and will predict a high proximity for the node pair (C, A). The discriminator is trained to distinguish whether the generated neighborhood is real or fake. Competition between the generators and discriminator drives both of them to improve their capability until the generators are indistinguishable from the true connectivity distribution.

The key contributions of this paper are as follows:

- To the best of our knowledge, DGGAN is the first GAN-based method for directed graph embedding that could jointly learn the source vector and target vector for each node.
- The two generators deployed in DGGAN are also able to generate effective negative samples for nodes with low or zero out- or in- degree, which makes the model learn more robust node embeddings across various graphs.
- Through extensive experiments on four real-world network datasets, we present that the proposed DGGAN method consistently and significantly outperforms various state-of-the-art methods on link prediction, node classification and graph reconstruction tasks.

Related Work

Undirected Graph Embedding

Graph embedding methods can be classified into three categories: matrix factorization-based models, random walk-based models and deep learning-based models. The matrix factorization-based models, such as GraRep (Cao, Lu, and Xu 2015) and M-NMF (Wang et al. 2017) first preprocess adjacency matrix which preserves the graph structure, and then decompose the prepocessed matrix to obtain graph embeddings. It has been shown that many recent emergence random walk-based models such as DeepWalk (Perozzi, Al-Rfou, and Skiena 2014), LINE (Tang et al. 2015), PTE (Tang, Qu, and Mei 2015) and node2vec (Grover and Leskovec 2016) can be unified into the matrix factorization framework with closed forms (Qiu et al. 2018). The deep learning-based models like SDNE (Wang, Cui, and Zhu 2016) and DNGR (Cao, Lu, and Xu 2016) learn graph embeddings by deep autoencoder model.

Adversarial Graph Embedding Recently, Generative Adversarial Network (GAN) (Goodfellow et al. 2014) has received increasing attention due to its impressing performance on the unsupervised task. GAN can be viewed as a minimax game between generator G and discriminator D. Formally, the objective function is defined as follows:

$$\min_{\theta^G}\max_{\theta^D} \quad \mathbb{E}_{x\sim p_{\text{data}}(x)}\left[\log D(x;\theta^D)\right] + \mathbb{E}_{z\sim p_z(z)}\left[\log(1-D(G(z;\theta^G);\theta^D))\right], \quad (1)$$

where θ^G and θ^D denote the parameters of G and D, respectively. The generator G tries to generate close-to-real fake samples with the noise z from a predefined distribution $p_z(z)$. While the discriminator D aims to distinguish the real ones from the distribution $p_{\text{data}}(x)$ and the fake samples. Several methods have been proposed to apply GAN for graph embedding to improve models robustness and generalization. GraphGAN (Wang et al. 2018) generates the sampling distribution to sample negative nodes. ANE (Dai et al. 2018) imposes a prior distribution on graph embeddings through adversarial learning. NetRA (Yu et al. 2018) and ARGA (Pan et al. 2020) adopt adversarially regularized autoencoders to learn smoothly embeddings. DWNS (Dai et al. 2019) applies adversarial training by defining adversarial perturbations in embeddings space.

Directed Graph Embedding

The methods mentioned above mainly focus on undirected graphs and thus cannot capture the directions of edges. There are some works for directed graph embedding, which commonly learn source embedding and target embedding for each node. HOPE (Ou et al. 2016) derives the node-similarity matrix by approximating high-order proximity measures like Katz measure (Katz 1953) and Rooted PageRank (Song et al. 2009), and then decomposes the node-similarity matrix to obtain node embeddings. APP (Zhou et al. 2017) is a directed random walk-based method to implicitly preserve Rooted PageRank proximity. NERD (Khosla et al. 2019) uses an alternating random walk strategy to sample node neighborhoods from a directed graph. ATP (Sun et al. 2019) incorporates graph hierarchy and reachability to construct the asymmetric matrix. However, these methods are all shallow methods, failing to capture the highly non-linear property in graphs and learn robust node embeddings.

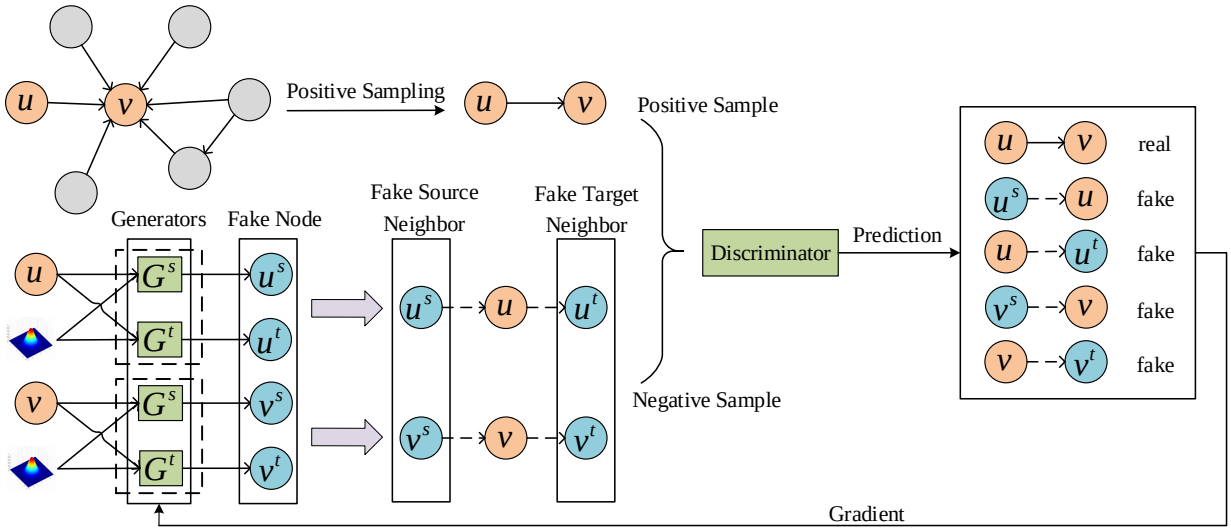


Figure 2: The architecture of DGGAN. The node pair (u, v) denotes real node pair. For node u, the two generators share a underlying distribution and jointly generate a fake source neighbor u^s and a fake target neighbor u^t. Likewise, the fake source and target neighbors can be generated for node v. Those fake node pairs aim to fool the discriminator with highest probability, while discriminator is trained to distinguish between the real node pair and fake node pair.

DGGAN model

In this section, we will first introduce the notations to be used. Then we will present an overview of DGGAN, followed by detailed descriptions of our generator and discriminator.

Notations

We define a directed graph as $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, where $\mathcal{V}$ is the node set, and $\mathcal{E}$ is the directed edge set. For nodes $u, v \in \mathcal{V}$, $(u, v) \in \mathcal{E}$ represents a directed edge from u to v. To preserve the asymmetric proximity, each node u needs to have two different roles, the source roles and target roles, represented by d dimensional vector $\mathbf{s}_u \in \mathbb{R}^{d\times 1}$ and $\mathbf{t}_u \in \mathbb{R}^{d\times 1}$, respectively.

Overall Framework of DGGAN

The objective of DGGAN is to jointly learn the source and target vectors for each node on a directed graph. Figure 2 demonstrates the proposed framework of DGGAN, which mainly consists of two components: generator and discriminator. Given a node (e.g., u), two generators are deployed to jointly generate its fake source neighborhood and target neighborhood from the same underlying continuous distribution. And one discriminator is set to distinguish whether the source neighborhood and the target neighborhood of the given node are real or fake. With the minimax game between generators and discriminator, DGGAN is able to learn more robust source embeddings and target embeddings for nodes with low indegree or outdegree, even the nodes with zero indegree or outdegree (e.g., u and v). Next, we introduce the details of generator and discriminator.

Generator and Discriminator in DGGAN

Directed, Generalized and Robust Generator The goal of our generator G is threefold: (1) It should generate close-to-real fake samples concerning specific direction. Thus, given a node $u \in \mathcal{V}$, the generator G aims to generate a fake source neighbor u^s and a fake target neighbor u^t where u^s and u^t should be as close as possible to the real nodes. (2) It should be generalized to non-existent nodes. In other words, the fake nodes u^s and u^t can be latent and not restricted to the original graph. (3) It should be able to generate efficient fake source and target neighborhoods for nodes with low or zero out- or in- degree.

To address the first aim, we design the generator G consisting of two types of generators: one source neighborhood generator G^s and one target neighborhood generator G^t. For the second and third aims, we propose to introduce a latent variable $\mathbf{z} \sim p_{\mathbf{z}}(\mathbf{z})$ shared between G^s and G^t to generate samples. Rather than directly generating samples from $p_{\mathbf{z}}(\mathbf{z})$, we integrate the multi-layer perception (MLP) into the generator for enhancing the expression of the fake samples, as deep neural networks have shown strong ability in capturing the highly non-linear property in a network (Gao, Pei, and Huang 2019; Hu, Fang, and Shi 2019). Therefore, our generator G is formulated as follows:

$$G^s(u;\theta^{G^s}) = f^s(\mathbf{z};\theta^{f^s}), \quad G^t(u;\theta^{G^t}) = f^t(\mathbf{z};\theta^{f^t}),$$
$$G(u;\theta^G) = \{G^s(u;\theta^{G^s}), G^t(u;\theta^{G^t})\}, \tag{2}$$

where f^s and f^t are implemented by MLP. θ^{f^s} and θ^{f^t} denote the parameters of f^s and f^t, respectively. $\mathbf{z}$ serves as a bridge between G^s and G^t. With the help of $\mathbf{z}$, G^s and G^t are not independent, and update each other indirectly to generate better fake source and target neighborhood. Partic-

ularly, we derive $\mathbf{z}$ from the following Gaussian distribution:

$$p_{\mathbf{z}}(\mathbf{z}) = \mathcal{N}(\mathbf{z}_u^{\mathrm{T}}, \sigma^2\mathbf{I}), \tag{3}$$

where $\mathbf{z}_u \in \mathbb{R}^{d\times 1}$ is a learnable variable and stands for the latent representation of $u \in \mathcal{V}$. The parameters of G^s and G^t are thus $\theta^{G^s} = \{\mathbf{z}_u^{\mathrm{T}} : u \in \mathcal{V}, \theta^{f^s}\}$ and $\theta^{G^t} = \{\mathbf{z}_u^{\mathrm{T}} : u \in \mathcal{V}, \theta^{f^t}\}$, respectively. Since θ^{G^s} and θ^{G^t} share parameter $\mathbf{z}_u^{\mathrm{T}}$, the parameters of the generator G can be obtained as follows:

$$\theta^G = \{\theta^{G^s}, \theta^{G^t}\} = \{\mathbf{z}_u^{\mathrm{T}} : u \in \mathcal{V}, \theta^{f^s}, \theta^{f^t}\}. \tag{4}$$

The generator G aims to fool the discriminator D by generating close-to-real fake samples. To this end, the loss function of the generator is defined as follows:

$$\mathcal{L}^G = \mathbb{E}_{u\in\mathcal{V}} \log\left(1 - D(u^s, u)\right) + \log\left(1 - D(u, u^t)\right), \tag{5}$$

where u^s and u^t denote the fake source neighbor and fake target neighbor of u, respectively. D outputs the probability that the input node pair is real, and will be introduced in the next subsection. The source vector of u^s and target vector of u^t can be obtained by G^s and G^t, i.e., $\mathbf{s}_{u^s} \sim G^s(u; \theta^{G^s}), \mathbf{t}_{u^t} \sim G^t(u; \theta^{G^t})$. The parameters θ^G of the generator can be optimized by minimizing $\mathcal{L}^G$.

Directed Discriminator The discriminator D tries to distinguish the positive samples from the input graph $\mathcal{G}$ and the negative samples produced by the generator G. Thus, D could enforce G to more accurately fit the real graph distribution $p_{\mathcal{G}}$. Note that for a given node pair (u, v), D essentially outputs a probability that the sample v is connected to u in the outgoing direction. For this purpose, we define the D as the sigmoid function of the inner product of the input node pair (u, v):

$$D(u, v; \theta^D) = \frac{1}{1 + \exp(-\mathbf{s}_u{}^{\mathrm{T}} \cdot \mathbf{t}_v)}, \tag{6}$$

where $\theta^D = \{\mathbf{s}_u, \mathbf{t}_u : u \in \mathcal{V}\}$ is the parameter for D, i.e., the union of source role embeddings and target role embeddings of all real node on the observed $\mathcal{G}$. Specifically, the input node pair can be divided into the following two cases.

Positive Sample There indeed exists a directed edge from u to v on the $\mathcal{G}$, i.e., $(u, v) \in \mathcal{E}$, such as (u, v) shown in Figure 2. Such node pair (u, v) is considered positive and can be modeled by the following loss:

$$\mathcal{L}^D_{\mathrm{pos}} = \mathbb{E}_{(u,v)\sim p_{\mathcal{G}}} - \log D(u, v). \tag{7}$$

Negative Sample For a given node $u \in \mathcal{V}$, u^s and u^t denote its fake source neighbor and fake target neighbor generated by G^s and G^t, respectively, i.e., $\mathbf{s}_{u^s} \sim G^s(u; \theta^{G^s}), \mathbf{t}_{u^t} \sim G^t(u; \theta^{G^t})$, such as (u^s, u) and (u, u^t) shown in Figure 2. Such node pairs (u^s, u) and (u, u^t) are considered negative and can be modeled by the following loss:

$$\mathcal{L}^D_{\mathrm{neg}} = \mathbb{E}_{u\in\mathcal{V}} - \log\left(1 - D(u^s, u)\right) - \log\left(1 - D(u, u^t)\right). \tag{8}$$

Algorithm 1 DGGAN framework

Require: directed graph $\mathcal{G}$, number of maximum training epochs n^{epoch}, numbers of generator and discriminator training iterations per epoch n^G, n^D, number of samples n^s
Ensure: θ^G, θ^D

```
Initialize θ^G and θ^D for G and D, respectively
for epoch = 0; epoch < n^epoch do
    for n = 0; n < n^D do
        Generate n^s fake source neighbors u^s, v^s and fake target neighbors u^t, v^t for each node pair (u, v) ∈ E
        Update θ^D according to Eq.(9)
    end for
    for n = 0; n < n^G do
        Generate n^s fake source neighbors u^s and fake target neighbors u^t for each node u ∈ V
        Update θ^G according to Eq.(5)
    end for
end for
```

Note that the fake node embedding $\mathbf{s}_{u^s}$ and $\mathbf{t}_{u^t}$ are not included in θ^D and the discriminator D simply treats them as non-learnable input.

We integrate above two parts to train the discriminator:

$$\mathcal{L}^D = \mathcal{L}^D_{\mathrm{pos}} + \mathcal{L}^D_{\mathrm{neg}}. \tag{9}$$

The parameters θ^D of the discriminator can be optimized by minimizing $\mathcal{L}^D$.

Training of DGGAN

Training Algorithm In each training epoch, we alternate the training between the discriminator D and generator G with mini-batch gradient descent. Specifically, we first fix θ^G and the two generators jointly generate fake neighborhoods for each node pair on the graph to optimize θ^D. Then we fix θ^D and optimize θ^G to generate close-to-real fake neighborhoods for each node under the guidance of the discriminator D. The discriminator and generator play against each other until DGGAN converges. The overall training algorithm for DGGAN is summarized in Algorithm 1.

Complexity Analysis The time complexity for the discriminator D in each iteration is $O(n^s \cdot |\mathcal{E}| \cdot d^2)$, where $|\mathcal{E}|$ is the number of edges and d is the embedding dimension of the node. The time complexity for the generator G in each iteration is $O(n^s \cdot |\mathcal{V}| \cdot d^2)$, where $|\mathcal{V}|$ is the number of nodes. Therefore, the overall time complexity of DGGAN per epoch is $O\left(n^s \cdot (n^D \cdot |\mathcal{E}| + n^G \cdot |\mathcal{V}|) \cdot d^2\right)$. Since n^s, n^G, n^D and d are small constants, the time complexity is linear to the number of edges $|\mathcal{E}|$. The space complexity of DGGAN is $O(2 \cdot d \cdot |\mathcal{V}| + |\mathcal{E}|)$. We can see that, DGGAN is both time and space efficent and is scalable for large scale graphs.

Experiments

In this section, we conduct extensive experiments on several datasets to investigate the performance of DGGAN.

Table 1: Area Under Curve (AUC) scores of link prediction on directed graphs with different fractions of positive edges except bi-directional edges been reversed to create negative edges in the test set (scores are with %). Best scores are shown in bold and the second-best scores are underlined.

method	Cora			Twitter			Epinions			Google		
	0%	50%	100%	0%	50%	100%	0%	50%	100%	0%	50%	100%
DeepWalk	84.9±1.39	68.1±0.43	52.9±0.12	50.4±0.67	50.3±0.21	50.3±0.01	76.6±1.21	67.9±0.54	66.6±0.12	83.6±2.61	72.1±0.65	66.5±0.32
LINE-1	84.7±0.63	68.0±0.25	52.5±0.06	53.1±0.45	51.5±0.13	50.0±0.01	78.8±0.52	69.8±0.26	68.5±0.05	89.7±0.82	72.7±0.45	65.1±0.21
node2vec	**85.3±1.07**	65.5±0.35	52.1±0.09	50.6±0.75	50.5±0.33	50.3±0.01	89.7±0.31	74.6±0.12	72.6±0.02	84.3±1.13	70.5±0.53	64.3±0.26
GraphGAN	51.6±0.67	51.3±0.31	51.2±0.12	-	-	-	-	-	-	71.3±2.37	61.1±1.59	56.2±1.13
ANE	72.8±0.53	61.4±0.28	51.5±0.07	49.7±0.53	49.8±0.29	50.0±0.02	85.5±2.15	69.2±0.74	66.9±0.24	76.1±1.86	63.7±0.83	57.8±0.53
LINE-2	69.3±0.47	72.1±0.23	74.3±0.05	95.6±0.37	95.7±0.13	95.8±0.01	58.1±0.67	67.1±0.52	68.4±0.41	77.4±0.24	85.2±0.17	89.0±0.13
HOPE	77.6±1.53	74.2±0.65	71.5±0.42	98.0±0.63	97.9±0.42	97.8±0.03	79.6±1.13	71.7±0.57	70.5±0.23	87.5±0.46	85.6±0.32	84.6±0.38
APP	76.6±0.83	76.4±0.41	76.2±0.11	71.6±0.57	70.1±0.36	68.7±0.01	70.5±0.47	71.3±0.23	71.4±0.09	92.1±0.21	86.4±0.15	81.0±0.13
DGGAN*	83.0±0.91	83.3±0.53	83.5±0.25	99.4±0.27	99.3±0.12	99.2±0.01	92.7±0.85	80.0±0.36	78.2±0.21	91.6±0.63	89.2±0.47	87.7±0.26
DGGAN	85.1±0.63	**86.7±0.31**	**88.3±0.11**	**99.7±0.15**	**99.7±0.09**	**99.7±0.01**	**96.1±0.51**	**86.4±0.25**	**85.1±0.11**	**92.3±0.52**	**93.4±0.36**	**94.4±0.23**

Table 2: Statistics of datasets.

dataset	#nodes	#edges	Avg. degree	#labels
Cora	23,166	91,500	7.90	10
CoCit	44,034	195,361	8.86	15
Twitter	465,017	834,797	3.59	-
Epinions	75,879	508,837	13.41	-
Google	15,763	171,206	21.72	-

Dataset Description

We use four different types of directed graphs, including citation network, social network, trust network and hyperlink network to evaluate the performance of the model. The details of the data are described as follows: **Cora** (Šubelj and Bajec 2013) and **CoCit** (Tsitsulin et al. 2018) are citation networks of academic papers. Nodes represent papers and directed edges represent the citation relationships between papers. Labels represent conferences in which papers were published. **Twitter** (Choudhury et al. 2010) is a social network. Nodes represent users and directed edges represent following relationships between users. **Epinions** (Richardson, Agrawal, and Domingos 2003) is a trust network from the online social network Epinions. Nodes represent users and directed edges represent trust between users. **Google** (Palla et al. 2007) is a hyperlink network from pages within Google's sites. Nodes represent pages and directed edges represent hyperlink between pages. The statistics of these networks are summarized into Table 2.

Experiment Settings

To verify the performance of DGGAN[1], we compare it with several state-of-the-art methods.

- **Traditional undirected graph embedding methods**: DeepWalk (Perozzi, Al-Rfou, and Skiena 2014) uses local information obtained from truncated random walks to learn node embeddings. LINE (Tang et al. 2015) learns large-scale information network embedding using first-order and second-order proximities. node2vec (Grover and Leskovec 2016) is a variant of DeepWalk and utilizes a biased random walk algorithm to more efficiently explore the neighborhood architecture.
- **GAN-based undirected graph embedding methods**: GraphGAN (Wang et al. 2018) generates the sampling distribution to sample negative nodes from the graph. ANE (Dai et al. 2018) proposes to train a discriminator to push the embedding distribution to match the fixed prior.
- **Directed graph embedding methods**: HOPE (Ou et al. 2016) preserves the asymmetric role information of the nodes by approximating high-order proximity measures. APP (Zhou et al. 2017) proposes a random walk based method to encode Rooted PageRank proximity.
- DGGAN* is a simplified version of DGGAN which uses only one generator G^t to generate target neighborhoods of each node. We omit another simplified version which uses only one generator G^s as we do not observe a significant performance difference compared with DGGAN*.

For DeepWalk, node2vec and APP, the number of walks, the walk length and the window size are set to 10, 80 and 10, respectively, for fair comparision. node2vec is optimized with grid search over its return and in-out parameters $(p, q) \in \{0.25, 0.50, 1, 2, 4\}$ on each dataset and task. For LINE, we utilize both the first-order and the second-order proximities. For the second-order proximities, node embeddings are considered as source embeddings, and context embeddings are used as target embeddings. In addition, the number of negative samples is empirically set to 5. For GraphGAN, ANE and HOPE, we follow the parameters settings in the original papers. Note that we do not report the results of GraphGAN on Twitter and Epinions datasets, since it cannot run on these two large datasets. For DGGAN* and DGGAN, we choose parameters by cross validation and we fix the numbers of generator and discriminator training iterations per epoch $n^G = 5, n^D = 15$ across all datasets and tasks. Throughout our experiments, the dimension of node embeddings is set to 128.

Link Prediction

In link prediction task, we predict missing edges given a network with a fraction of removed edges. A fraction of edges is removed randomly to serve as test split while the remaining

[1]https://github.com/RingBDStack/DGGAN

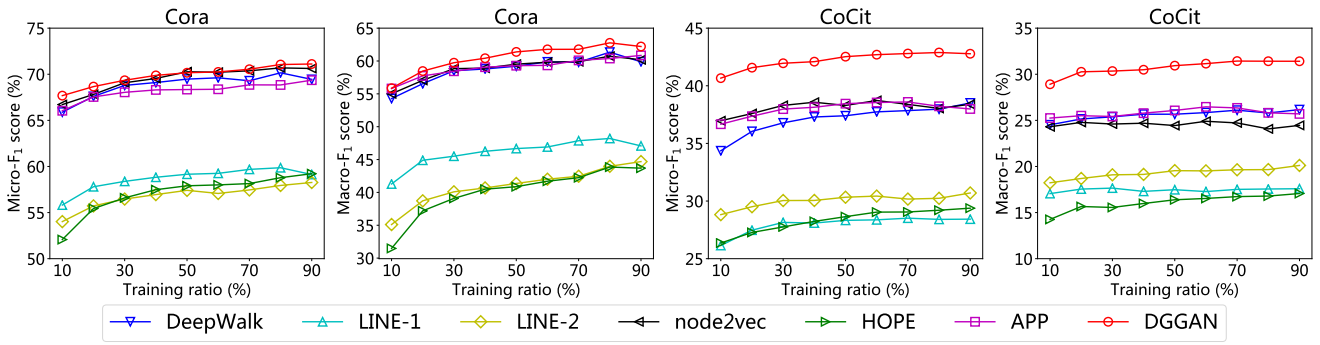


Figure 3: Performance comparison of node classification task on Cora and CoCit datasets. The x axis denotes the training ratio of labeled nodes, and the y axis denotes Micro-F_1 or Macro-F_1 score.

network are utilized for training. When removing edges randomly, we make sure that no node is isolated to avoid meaningless embedding vectors. Specifically, we remove 50% of edges for Cora, Epinions and Google datasets, and 40% of edges for Twitter dataset. Note that the test split is balanced with negative edges sampled from random node pairs that have no edges between them. Since we are interested in both the existence of the edge between two nodes and the direction of the edge, we reverse a fraction of node pairs in the positive samples to replace the original negative samples if the edges are not bi-directional. A value in $(0; 1]$ determines what fraction of positive edges from the test split are inverted at most to create negative examples. And a value of 0 corresponds to the classical undirected graph setting where all the negative edges are sampled from random node pairs.

We summarize the Area Under Curve (AUC) scores for all methods in Table 1. The reported results are the average performance of 10 times experiments. Note that some methods like DeepWalk which mainly focus on undirected graphs, also achieve good performance on Cora dataset with random negative edges in test set. But their performance decreases rapidly with the increase of reversed positive edges as they cannot model the asymmetric proximity, and their AUC scores are near 0.5 as expected. HOPE shows good performance on Twitter dataset but does not perform well on other datasets like Cora and Epinions. It suggests that HOPE is difficult to be generalized to different types of graphs as mentioned above. Note that on Epinions dataset, up to 31.5% nodes have no incoming edges and 20.5% nodes have no outgoing edges. The directed graph embedding methods like APP show poor performance on Epinions dataset. The reason is that these methods treat the source role and target role of one node separately, which renders them not robust. We can see that DGGAN* shows much better performance than HOPE and APP across datasets. This is because the negative samples of DGGAN* are generated directly from a continuous distribution and thus DGGAN* is not sensitive to different graph structures. Moreover, DGGAN outperforms DGGAN* as DGGAN utilizes two generators which mutually update each other to learn more robust source and target vectors. Compared with baselines, the performance of DGGAN does not change much with different fractions of reversed positive test edges. Overall, DGGAN shows more robustness across datasets and outperforms all methods on link prediction task.

Node Classification

To further verify whether a network embedding method can discover and preserve the proximity, we conduct multi-class classification on Cora and CoCit datasets. Specifically, we randomly sample a fraction of the labeled nodes as training data and the task is to predict the labels for the remaining nodes. We train a standard one-vs-rest L2-regularized logistic regression classifier on the training data and evaluate its performance on the test data. We report Micro-F_1 and Macro-F_1 scores as evaluation metrics. Note that for the methods using two embedding matrices, we set the dimension of node embeddings $d = 64$ and concatenate the two 64-dimensional embedding vectors into a 128-dimensional vector to represent each node.

Figure 3 summarizes the experimental results when varying the training ratio of the labeled nodes. Each result is averaged by 10 runs. The results exhibit similar trends as follows. First, DGGAN consistently outperforms all of the other methods across all training ratios on both datasets. It demonstrates that DGGAN can effectively capture the neighborhood information in a robust manner through the adversarial learning framework. Second, the undirected graph embedding methods DeepWalk and node2vec show good performance, and perform as well as the directed method APP. This suggests that the directionality might have limited impact on performance for node classification task on these two datasets. Third, we notice that although HOPE have good link prediction results, yet it performs poorly on this node classification task. The reason might be that HOPE is linked to a particular proximity measure, which makes it hard to generalize to different tasks.

Graph Reconstruction

As the effective representations of a graph, node embeddings maintain the edge information and are expected to well reconstruct the original graph. We reconstruct the graph edges based on the reconstructed proximity between nodes. Since each adjacent node should be close in the embedding

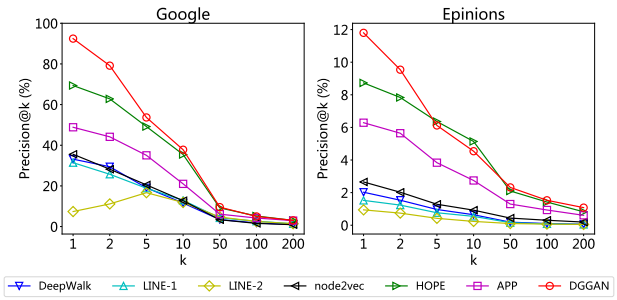
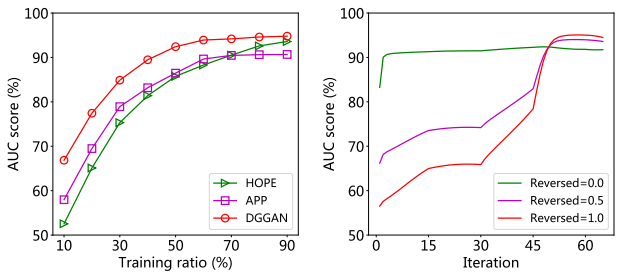


Figure 4: Precision@k of graph reconstruction task on Google and Epinions datasets.



(a) Sparsity (b) Learning curves

Figure 5: Performance w.r.t. network sparsity and learning curves of DGGAN on Google link prediction task.

space, we use inner product between node vectors to reconstruct the proximity matrix. For a given k, we obtain the k-nearest target neighbors ranked by reconstructed proximity for each method. We perform the graph reconstruction task on Google and Epinions datasets. In order to create the test set, we randomly sample 10% of the nodes on each graph.

We plot the average precision corresponding to different values of k in Figure 4. The results show that for both datasets, DGGAN outperforms baselines including HOPE and APP, especially when k is small. For Google dataset, DGGAN shows an improvement of around 33% for $k = 1$ over the second best performing method, HOPE. This shows the benefit of jointly learning the source and target vectors for each node. Some of the methods that focus on undirected graphs like node2vec exhibited good performance in link prediction. However, these methods show poor performance in graph reconstruction. This is because this task is harder than link prediction as the model needs to distinguish between small number of positive edges with a large number of negative edges. Besides, we note that all the precision curves converge to points with small values when k becomes large since most of the real target neighborhoods have been correctly predicted by these methods.

Model Analysis

In this subsection, we analyze the performance of different models under different levels of sparsity of networks and the converging performance of DGGAN. We choose Google dataset as it is much denser than the others. We first investigate how the sparsity of the networks affects the three directed graph embedding methods HOPE, APP and DGGAN. The setting of training procedure in this experiment is the same as link prediction and 50% positive edges of test set are reversed to form negative edges. We randomly select different ratios of edges from the original network to construct networks with different levels of sparsity.

Figure 5(a) shows the results with respect to the training ratio of edges on Google dataset. One can see that DGGAN consistently and significantly outperforms HOPE and APP across different training ratios. Moreover, DGGAN still achieves much better performance when the network is very sparse. While HOPE and APP extremely suffer from nodes with very low outdegree or indegree as mentioned before. It demonstrates that the novel adversarial learning framework DGGAN, which is designed to jointly learn a node's source vector and target vector, can significantly improve the robustness.

Next, we investigate performance change with respect to the training iterations of the discriminator D. Recall that we set the parameter of discriminator training iterations per epoch $n^D = 15$. Figure 5(b) shows the converging performance of DGGAN on Google dataset with different percentage of reversed positive edges of test set (results on other datasets show similar trends and are not included here). With the increase of iterations of D, the performance of Reversed=0.0 (i.e. random negative edges in test set) keeps stable first, and then slightly increases. Besides, the training curve trend of Reversed=1.0 (i.e. all positive edges except bi-directional edges are reversed to create negative edges in test set) changes every 15 iterations (i.e. one epoch). Note that the training curve trend of Reversed=1.0 rises gently during second epoch (i.e. iteration $[16, 30]$) for the generator G still been poorly trained at the moment. The trend rises steep in the following epoch for G being able to generate close-to-real fake samples.

Conclusion

In this paper, we proposed DGGAN, a novel directed graph embedding framework based on GAN. Specifically, we designed two generators which generate fake source neighborhood and target neighborhood for each node directly from same continuous distribution. With the jointly learning framework, the two generators can be mutually enhanced, which renders the proposed DGGAN generalized for various graphs and more robust to learn node embeddings. The experimental results on four real-world directed graph datasets demonstrated that DGGAN consistently and significantly outperforms various state-of-the-arts on link prediction, node classification, and graph reconstruction tasks.

Acknowledgments

Jianxin Li is the corresponding author. This work was supported by grants from the Natural Science Foundation of China (61872022, U20B2053 and 62002007) and State Key Laboratory of Software Development Environment (SKLSDE-2020ZX-12).

References

Bhagat, S.; Cormode, G.; and Muthukrishnan, S. 2011. Node classification in social networks. In *Social network data analytics*, 115–148. Springer.

Bollacker, K. D.; Lawrence, S.; and Giles, C. L. 1998. CiteSeer: An autonomous web agent for automatic retrieval and identification of interesting publications. In *AGENTS*, 116–123. ACM.

Cao, S.; Lu, W.; and Xu, Q. 2015. Grarep: Learning graph representations with global structural information. In *CIKM*, 891–900. ACM.

Cao, S.; Lu, W.; and Xu, Q. 2016. Deep neural networks for learning graph representations. In *AAAI*, 1145–1152. AAAI Press.

Cha, M.; Haddadi, H.; Benevenuto, F.; and Gummadi, K. P. 2010. Measuring user influence in twitter: The million follower fallacy. In *AAAI*. AAAI Press.

Choudhury, M. D.; Lin, Y.-R.; Sundaram, H.; Candan, K. S.; Xie, L.; and Kelliher, A. 2010. How Does the Data Sampling Strategy Impact the Discovery of Information Diffusion in Social Media? In *ICWSM*, 34–41. AAAI Press.

Dai, Q.; Li, Q.; Tang, J.; and Wang, D. 2018. Adversarial network embedding. In *AAAI*, 2167–2174. AAAI Press.

Dai, Q.; Shen, X.; Zhang, L.; Li, Q.; and Wang, D. 2019. Adversarial training methods for network embedding. In *WWW*, 329–339. ACM.

Gao, H.; Pei, J.; and Huang, H. 2019. ProGAN: Network Embedding via Proximity Generative Adversarial Network. In *KDD*, 1308–1316. ACM.

Goodfellow, I.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; and Bengio, Y. 2014. Generative adversarial nets. In *NIPS*, 2672–2680.

Grover, A.; and Leskovec, J. 2016. node2vec: Scalable feature learning for networks. In *KDD*, 855–864. ACM.

Hu, B.; Fang, Y.; and Shi, C. 2019. Adversarial Learning on Heterogeneous Information Networks. In *KDD*, 120–129. ACM.

Katz, L. 1953. A new status index derived from sociometric analysis. *Psychometrika* 18(1): 39–43.

Khosla, M.; Leonhardt, J.; Nejdl, W.; and Anand, A. 2019. Node Representation Learning for Directed Graphs. In *ECML PKDD*, volume 11906 of *Lecture Notes in Computer Science*, 395–411. Springer.

Liben-Nowell, D.; and Kleinberg, J. 2007. The link-prediction problem for social networks. *JASIST* 58(7): 1019–1031.

Ou, M.; Cui, P.; Pei, J.; Zhang, Z.; and Zhu, W. 2016. Asymmetric transitivity preserving graph embedding. In *KDD*, 1105–1114. ACM.

Palla, G.; Farkas, I. J.; Pollner, P.; Derényi, I.; and Vicsek, T. 2007. Directed Network Modules. *New J. Phys.* 9(6): 186.

Pan, S.; Hu, R.; Fung, S.; Long, G.; Jiang, J.; and Zhang, C. 2020. Learning Graph Embedding With Adversarial Training Methods. *IEEE Trans. Cybern.* 50(6): 2475–2487.

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

Qiu, J.; Dong, Y.; Ma, H.; Li, J.; Wang, K.; and Tang, J. 2018. Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec. In *WSDM*, 459–467. ACM.

Richardson, M.; Agrawal, R.; and Domingos, P. 2003. Trust Management for the Semantic Web. In *The Semantic Web-ISWC 2003*, 351–368. Springer.

Song, H. H.; Cho, T. W.; Dave, V.; Zhang, Y.; and Qiu, L. 2009. Scalable proximity estimation and link prediction in online social networks. In *IMC*, 322–335. ACM.

Šubelj, L.; and Bajec, M. 2013. Model of Complex Networks based on Citation Dynamics. In *WWW*, 527–530. ACM.

Sun, J.; Bandyopadhyay, B.; Bashizade, A.; Liang, J.; Sadayappan, P.; and Parthasarathy, S. 2019. Atp: Directed graph embedding with asymmetric transitivity preservation. In *AAAI*, volume 33, 265–272. AAAI Press.

Tang, J.; Qu, M.; and Mei, Q. 2015. Pte: Predictive text embedding through large-scale heterogeneous text networks. In *KDD*, 1165–1174. ACM.

Tang, J.; Qu, M.; Wang, M.; Zhang, M.; Yan, J.; and Mei, Q. 2015. Line: Large-scale information network embedding. In *WWW*, 1067–1077. ACM.

Tsitsulin, A.; Mottin, D.; Karras, P.; and Müller, E. 2018. Verse: Versatile graph embeddings from similarity measures. In *WWW*, 539–548. ACM.

Wang, D.; Cui, P.; and Zhu, W. 2016. Structural deep network embedding. In *KDD*, 1225–1234. ACM.

Wang, H.; Wang, J.; Wang, J.; Zhao, M.; Zhang, W.; Zhang, F.; Xie, X.; and Guo, M. 2018. Graphgan: Graph representation learning with generative adversarial nets. In *AAAI*, 2508–2515. AAAI Press.

Wang, X.; Cui, P.; Wang, J.; Pei, J.; Zhu, W.; and Yang, S. 2017. Community preserving network embedding. In *AAAI*. AAAI Press.

Ying, R.; He, R.; Chen, K.; Eksombatchai, P.; Hamilton, W. L.; and Leskovec, J. 2018. Graph convolutional neural networks for web-scale recommender systems. In *KDD*, 974–983. ACM.

Yu, W.; Zheng, C.; Cheng, W.; Aggarwal, C. C.; Song, D.; Zong, B.; Chen, H.; and Wang, W. 2018. Learning deep network representations with adversarially regularized autoencoders. In *KDD*, 2663–2671. ACM.

Zhou, C.; Liu, Y.; Liu, X.; Liu, Z.; and Gao, J. 2017. Scalable graph embedding for asymmetric proximity. In *AAAI*, 2942–2948. AAAI Press.