
# WalkGAN: Networks Representation Learning with Sequence-based Generative Adversarial Nets

| | |
|---|---|
| Journal: | *IEEE Transactions on Neural Networks and Learning Systems* |
| Manuscript ID | TNNLS-2020-P-15840 |
| Manuscript Type: | Regular Paper |
| Date Submitted by the Author: | 01-Dec-2020 |
| Complete List of Authors: | Jin, Taisong; Xiamen University, Department of Computer Science<br>Duan, Gengchen; Xiamen University<br>Yu, Zhengtao; Kunming University of Science and Technology, School of Information Engineering and Automation<br>Luo, Han; Xiamen University<br>Zeng, Xiangxiang; Hunan University,<br>Jiang, Min; Xiamen University, Cognitive Science and Technology Department |
| Keywords: | Networks representation, Generative Adversarial Nets, Random walk, Sequence |
| | |

# WalkGAN: Networks Representation Learning with Sequence-based Generative Adversarial Nets

Taisong Jin, Gengchen Duan, Zhengtao Yu, Han Luo, Xiangxiang Zeng, *Senior Member, IEEE,* Min Jiang, *Senior Member, IEEE,*

***Abstract*—Networks representation learning, also known as networks embedding, aims to learn low-dimensional representations of vertices while capturing and preserving the networks structure. For real-world networks, the edges that represent some important relationships between the vertices of the networks may be missed, resulting in the degenerated performance. The existing methods usually treat missing edges as negative samples, thus ignoring the true connections between two vertices in the networks. To effectively capture the true networks structure, we propose a novel networks representation learning method, namely WalkGAN, where both random walk scheme and Generative Adversarial Nets (GAN) are incorporated into a networks embedding framework. Specifically, WalkGAN leverages GAN to generate synthetic sequences of the vertices that sufficiently simulate random walk on the networks and further learn vertex representations from these sequences of the vertices. Thus, the unobserved links between the vertices are inferred with a high probability instead of treating them as non-existence. The experimental results on the benchmark networks datasets demonstrate that WalkGAN achieves significant performance improvements for node classification, link prediction, and visualization tasks.**

***Index Terms*—Networks representation; Generative Adversarial Nets; Random walk; Sequence.**

## I. Introduction

Networks are ubiquitous, and various real-world applications need to mine the useful information from the networks databases for information processing. An effective way of data mining is to embed the networks into a low-dimensional space, that is, to learn vector representations for each vertex by reconstructing the networks from the learned embedding space (also known as networks embedding) [1]. The informative networks features, which is crucial for a wide range of real-world tasks, can be mined through networks embedding. For example, the associations between drugs and targets can be inferred by analyzing the proximity of vector representations in the embedded space [2]. Friend recommendation and friend-circle analysis can be performed by examining the social networks [3]. Electronic websites recommend products to consumers by focusing on purchasing tendencies in recommendation systems.

Currently, learning the effective low-dimensional representations that preserve the networks structures faces big challenges. Specifically, many real-world networks are incomplete (edges that represent some relationships between nodes are missed or undiscovered), which degrades the performance of networks embedding. For instance, the construction of genome-wide networks relies on biological experiments, which are time-consuming and resource-intensive tasks. Gene networks account for only 20% of all potential existing gene interactions [4]. Meanwhile, the existing relationships and connections between genes remain unknown. More importantly, vertex features, which are learned from the incomplete networks, are not informative, limiting many real-world applications. Thus, it is crucial to discover the missing edges to derive more effective networks representations.

Most of the existing networks embedding methods ignore the missing edges in a network or treat the unknown potential relationships as negative samples. For instance, SDNE [5] determines the networks representation based on the existing edges. LINE [6] randomly selects the part of the disconnected paired vertices as negative samples, which accelerates the convergence speed of a model. DeepWalk [7] uses the local information derived from the truncated random walks to learn latent representations. Because of the high similarity between adjacent nodes in the sequence, the missing edges may result in incomplete sequences. Node2Vec [8] makes an advance by fusing the local and the global information of a network, where a biased random walk is proposed to yield the nodes that are indirectly connected in a sequence. Although these previous methods achieve promising performance in many applications, they cannot still effectively capture the true connections in the networks, leading to the limited performance improvement.

Note that Generative Adversarial Nets (GAN) [9] is a promising framework for alleviating the above problems. Specifically, a discriminator $D$ in GAN can distinguish the authenticity of data sample, and a generator $G$ attempts to confuse $D$ by generating the simulation data. Thus, the discriminator and generator can alternatively and iteratively boost the respective performance by each other. For instance, GraphGAN [10] is the first attempt that applies GAN to the network representation, which leverages a GAN framework to

This work was supported by the National Key R&D Program of China (No. 2018YFC0830105, 2018YFC0830100), the National Nature Science Foundation of China (62072386) (Corresponding author: Xiangxiang Zeng)

T. Jin is with Media Analytics and Computing Lab, Department of Computer Science, School of Informatics, Xiamen University, 361005, China.

G. Duan is with Media Analytics and Computing Lab, Department of Computer Science, School of Informatics, Xiamen University, 361005, China.

Z. Yu is with Yunnan Key Laboratory of Artificial Intelligence, Yunnan, Kunming, 650500, China and also with School of Information Engineering and Automation, Kunming University of Science and Technology, Yunnan, Kunming, 650500, China.

H. Luo is with Department of Computer Science, School of Informatics, Xiamen University, 361005, China.

X. Zeng is with College of Information Science and Engineering, Hunan University, 410082, Changsha, China.

M. Jiang is with Department of Artificial Intelligence, School of Informatics, Xiamen University, 361005, China.

generate the distribution of the neighborhoods of the vertices. NetGAN [11] leverages Wasserstein GAN (WGAN) [12] to generate the sequences of the vertices, where the generated sequences are sampled from a non-differentiable based data distribution.

Inspired by the recent advances of GAN and networks representation, we propose a novel Networks Representation Learning framework (WalkGAN) that combines random walk and GAN to eliminate the impact of missing edges. *Firstly*, the real node sequences of the vertices are derived by randomly walking on the networks. *Secondly*, both the discriminator and generator are trained by these sequences of the vertices until model convergence is reached. To confuse a discriminator, WalkGAN applies deep neural nets to generate the sequences of the vertices with the equal length by considering the nonlinear relationships. The competitive relationship between discriminator and generator in WalkGAN yields sufficiently reliable sequences of the vertices comparable with the real sequences. Besides, WalkGAN makes the disconnected vertex pairs in a network adjacent in the generated sequences, which can infer the unknown relationships that exist with a high probability. *Finally*, WalkGAN derives the low-dimensional embeddings from the generated sequences via Skipgram. The three contributions of our work are summarized as follows:

- To reveal the latent relationships in the networks, GAN is used to generate the vertex sequences that are sufficiently real. Based on the generated sequences, disconnected vertices can be generated adjacently, which is crucial to learn the effective low-dimensional embeddings for the vertices in a graph.
- Different from GraphGAN that generates the neighboring vertices by GAN, WalkGAN generates multiple sequences that are initialized from each vertex via random-walk on a network, which could capture the global information of a network.
- WalkGAN is applied to three real-world applications: link prediction, node classification, and visualization. The experimental results show that WalkGAN achieves a significant performance gain on network embedding.

The rest of this article is arranged as follows. Section 2 introduces the related work. Section 3 presents the proposed method. Section 4 reports the experimental results. Finally, we concludes the article in Section 5.

## II. Related work

For clarification of difference between networks data and neural networks learning models, we use *networks* to represent the networks data and use *nets* to represent the neural networks learning models.

Networks embedding transfers networks into a set of low dimensional vectors. Those vectors keep the networks topology, relationships between different vertices and other relevant information in the networks. For instance, DeepWalk [7] uses random walk strategy to transfer the networks into various sequences, where the Skipgram model is used to derive the vertex embeddedings. Node2Vec [8] leverages the random walk approach to generate the networks neighborhoods for each vertex, which can capture both long-term or short-term relationships among the vertices. LINE [6] leverages the $1^{st}$-order and $2^{nd}$-order similarity metrics to derive the vertex embeddings, where the $1^{st}$-order similarity is used to measure the pairwise similarity between two vertices and the $2^{nd}$-order similarity is used to measure the similarity between the neighborhoods of two vertices. SDNE [5] is a deep auto-encoder using a reconstruction loss, which leverages both $1^{st}$-order and $2^{nd}$-order similarity to supervise the training process. Struc2Vec [13] designs a more complicate structural similarity, which uses a hierarchy to measure vertex similarity at different scales for constructing a multi-layer network. To extract the long term relations between different nodes, HARP [14] leverages a collapsing strategy to generate a smaller networks, which can be used as the input of Node2Vec, DeepWalk or LINE. Walklets [15] uses a skipping random walk to generate a corpus of vertex pairs, which are reachable via the paths of a fixed length. This corpus can be used to learn a series of latent representations in a way that is analytically derivable. DNGR [16] uses random surfing instead of random walk to capture the graph structural information directly and applies the denoising autoencoder to extract complex features.

The aforementioned networks embedding methods perform well on the no-attributed networks. There are also many approaches for the attributed graphs. To enable neighboring vertices to share the networks dependency structure, Semi-implicit graph variational auto-encoder (SIG-VAE) [17] employs a hierarchical variational framework. As SIG-VAE leverages GCN to derive the vertex embeddings, SIG-VAE can hardly generate promising performance for the networks without the vertex features. Gat2Vec [18] uses the networks attributes to generate attribute contexts and networks topology. After the attribute and structural contexts are generated, Gat2Vec learns a joint representation by a shallow neural nets.

Graph convolutional networks (nets) aggregate the features by employing the convolutional operation on the network data, which can be divided into both spectral-based models and spatial-based models. *On the one hand*, the spectral-based models define the convolution in the graph Fourier domain, which is involved in the eigendecomposition of the Laplacian matrix [19]. Thus, the spectral-based models usually have high computational complexity, which is intractable for large the networks. ChebNet [20] approximates the graph filter as Chebyshev polynomials of the diagonal matrix of eigenvalues, which enhances the computation efficiency. CayleyNet [21] introduces the parametric rational complex functions-Cayley polynomials to capture narrow frequency bands. GCNs [22] relies on a first-order approximation of ChebNet with less computation cost. Dual Graph Convolutional Network (DualGCN) [23] has two graph convolutional layers to work in parallel. *On the other hand*, the spatial-based models define the groups of filters directly on the vertex domain in the spatial space. Neural Networks for Graphs (NN4G) [24] performs graph convolutional operation on the spatial space. PATCHY-SAN [25] ranks the adjacent nodes of each node based on the graph labeling, and then applies a standard 1D convolutional filter to aggregate the neighborhood features. The diffusion Convolutional Neural Network (DCNN) [26] treats graph

convolutions as a diffusion process to efficiently learn the feature that is invariant under isomorphism. Graph Attention Network (GAT) [27] introduces masked self-attentional layers to assign different weights to the neighboring nodes, leading to the learnable filter weights. The Mixture Model Network (MoNet) [28] introduces node pseudo-coordinates to assign different weights to neighbors of each node. The Message Passing Neural Network (MPNN) [29] considers graph convolution as the message passing process among nodes. Large-scale Graph Convolutional Network (LGCN) [30] ranks a node's neighbors via the node feature values. Then, multiple 1D convolutional layers are stacked for feature aggregation. DGI [31] relies on maximizing mutual information between patch representations and corresponding high-level summaries of graphs both derived using established graph convolutional network architectures.

Compared with the graph convolutional networks, networks embedding has the following advantages: (1) Most of the networks embedding methods apply on the non-attributed networks, which means networks embedding is more generic than graph convolutional networks (nets), since the attributed networks could be treated as non-attributed networks by ignoring the vertex features, while graph convolutional networks require the attributed networks as an input. (2) The experimental results have shown that the networks embedding methods extract the networks topology information better than graph convolutional networks (nets) for the non-attributed networks. (3) vertex embeddings generated by networks embedding can be used in various kind of tasks such as node classification and link predication. However, graph convolutional networks (nets) are mostly the end-to-end methods, which usually only focus on one specific task.

Considering the advantages of networks embedding, we propose a novel networks embedding method, termed WalkGAN. WalkGAN can capture the networks structure via Generative Adversarial Nets, whose performance is evaluated on multiple learning tasks including link prediction, node classification, and visualization.

## III. WalkGAN

In this section, we first provide an overview of the proposed framework of WalkGAN and then present the details of the implementation and optimization of a generator network and a discriminator, separately (see Fig. 1).

### A. Overview of our model

The workflow of WalkGAN is illustrated as follows: Firstly, WalkGAN collects the real sequences of the vertices $S$ by randomly walking on the networks to provide positive samples for the subsequent training of a discriminator. Secondly, a generator $G_\theta$ and a discriminator $D_\phi$ with random weights are initialized to have an pre-trained operation before training. Thirdly, the synthetic sequences of the vertices are collected from a generator until the model is converged. Finally, the networks representation are derived.

A generator $G_\theta$ is designed to generate synthetic sequences of the vertices $Y_{1:T} = (y_1, \cdots, y_t, \cdots, y_T), y_t \in V$ from the networks, where $V$ represents all candidate vertices. Through the generated sequence, the underlying pairwise relationship between two vertices can be effectively discovered. In other words, we can infer the existence of the edge between two vertices in the networks.

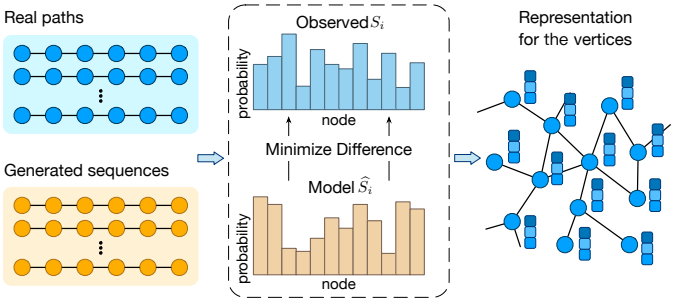



Fig. 1. WalkGAN learns the representation for vertices from those generated sequences that contain reliable inference about unobserved links.

To evaluate the sequences of vertices generated from the networks by a generator, a discriminator $D_\phi$ is trained by positive samples that are collected from the real sequences of the vertices $S$ via random walk on the networks and negative samples that are collected from the generated sequences by a generator.

As the discrete property of the sequence of the vertices, we follow the Sequence-based Generative Adversarial Networks framework [32], which leverages the policy gradient approach in reinforcement learning [33] and a Monte Carlo search [34] to update a generator $G_\theta$ based on the reward received from a discriminator $D_\phi$. The reward is estimated by the likelihood that it would fool the discriminator. The training of WalkGAN is illustrated in Fig. 2 and Algorithm 1 lists the main steps of WalkGAN.

In the following, we will introduce the details of WalkGAN.

### B. Random Walk on the networks

Random walk is a stochastic process with a sequence of the random variables. WalkGAN adopts random walk to generate the real sequences of the vertices from the networks. Due to the promising performance in real applications, we leverage the DeepWalk framework [7] as a random walk generator of WalkGAN, which samples uniformly a random vertex from the networks as the beginning of a random walker. And then, this walker randomly samples one vertex from the neighboring vertices of the last visited vertex in the networks. The aforementioned procedure is repeated until the fixed length of the sequence of the vertices are derived. ( For the greater details of random walk for generating a sequence of the vertices, please refer to [7]).

### C. Generator

Recurrent Neural Networks (RNNs) with Long Short-Term Memory (LSTM) [35] have shown excellent performance in dealing with the common vanishing and exploding gradient

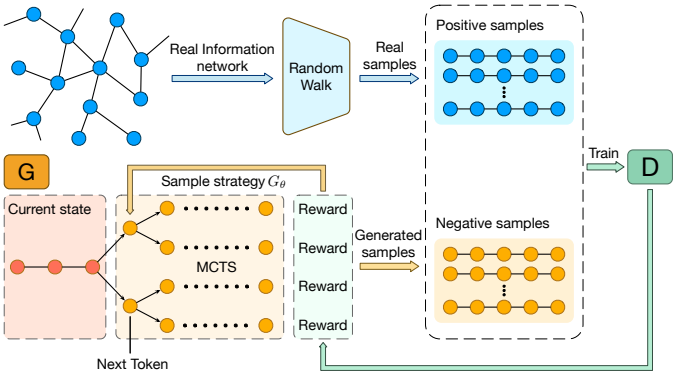



Fig. 2. The training of WalkGAN. A discriminator is trained by the positive samples collected by randomly walk on the original networks and the negative samples generated by a generator. A generator is trained by a gradient descent strategy, where the reward provided by a discriminator can be turned to the intermediate state by Monte Carlo search.

problem. Considering the speciality of sequence in memorizing, we use RNNs as a generator, which is widely used in solving the sequence problems.

A generator of WalkGAN is aiming to generate the sequences of vertices from the networks, which can indicate whether the edge exists between two vertices in the networks. If two vertices come from the generated sequence, which indicates that the edges may exist between two vertices.

For generating a simulated sequences of vertices from the networks, a generator generates a sequence of the vertex from the initial state $s_0$ to maximize its expected ending-reward, defined as follows:

$$J(\theta) = \mathbb{E}\left[R_T | s_0, \theta\right] = \sum_{y_1 \in Y} G_\theta\left(y_1 | s_0\right) \cdot Q_{D_\phi}^{G_\theta}\left(s_0, y_1\right), \quad (1)$$

where $R_T$ is a reward of a complete sequence received from a discriminator. $Q_{D_\phi}^{G_\theta}(s,a)$ is the action-value function of a sequences, i.e., the expected accumulative reward starting from state $s$, taking action $a$, and then following the strategy $G_\theta$. Then, we need to know how to calculate the action-value function. Because a generator aims to generate a sequence of the vertices from the networks, which makes the discriminator consider it is real, we adopt the feedback of the final sequence from a discriminator $D_\theta\left(Y_{1:T}^n\right)$ as reward:

$$Q_{D_\phi}^{G_\theta}\left(s = Y_{1:T-1}, a = y_T\right) = D_\phi\left(Y_{1:T}\right). \quad (2)$$

Due to the discrete property of the generated sequences of the vertices, the intermediate rewards need to know how to choose the next tokens. However, a discriminator only evaluates the final state. At timestep $t$, both the accumulated reward of previous tokens and the resulted outcome in the future are considered. The greedy choice is not always the best for the target state. Sometimes, the best choice at the current state should be abandoned, since we actually only care about the long-term reward. To calculate the action-value function considering the final rewards for an intermediate state, we uses $N$-time Monte Carlo search to sample the remaining unknown $T-t$ vertices, which is denoted as

$$\left\{Y_{1:T}^1, \cdots, Y_{1:T}^N\right\} = \mathrm{MC}^{G_\beta}\left(Y_{1:t}; N\right), \quad (3)$$

where $Y_{1:t}^n = (y_1, \ldots, y_t)$, and $Y_{t+1:T}^n$ are sampled based on the current state and the sampling strategy $G_\beta$.

To calculate the action value more accurately, we leverage the roll-out policy $G_\beta$ from the intermediate state to the end of sequences of the vertices for $N$ times, resulting in a batch of samples. The action-value function at that moment is calculated as follows:

$$Q_{D_\phi}^{G_\theta}\left(s = Y_{1:t-1}, a = y_t\right) = \begin{cases} \frac{1}{N}\sum_{n=1}^{N} D_\phi\left(Y_{1:T}^n\right), Y_{1:T}^n \in \mathrm{MC}^{G_\beta}\left(Y_{1:t}; N\right) & t < T \\ D_\phi\left(Y_{1:t}\right) & t = T \end{cases} \quad (4)$$

The reward of an current action is derived by continuously sampling from the state $S' = Y_{1:t}$ until the end of a sequence, and the process of maximizing the long-term reward equals to optimizing a generator. The generator can be trained dynamically by the rewards passed from a discriminator. Once a set of more realistic sequences of the vertices are generated, these sampled sequences will be used to re-train a discriminator. We used the gradient descent strategy to update a generator, and the derivative of objective function $J(\theta)$ is defined as follows:

$$\nabla_\theta J(\theta) = \sum_{t=1}^{T} \mathbb{E}_{Y_{1:t-1} \sim G_\theta} \left[\sum_{y_t \in \mathcal{Y}} \nabla_\theta G_\theta\left(y_t | Y_{1:t-1}\right) \cdot Q_{D_\phi}^{G_\theta}\left(Y_{1:t-1}, y_t\right)\right] \quad (5)$$

After that, the parameters of a generator could be updated as follows:

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta), \quad (6)$$

where $\alpha$ is the learning rate.

### D. Discriminator

Recently, deep neural network (DNN) [36], convolutional neural network(CNN) [37] and recurrent convolutional neural network (RCNN) [38] have been shown promising performance for sequence based classification problems. In this article, we choose the CNN as a discriminator, since CNN has shown promising performance for text classification [39]. We firstly map the input sequence of the vertices $x_1, \cdots, x_T$ into the projected space as follows:

$$\mathcal{E}_{1:T} = \boldsymbol{x}_1 \oplus \boldsymbol{x}_2 \oplus \cdots \oplus \boldsymbol{x}_T, \quad (7)$$

where $\boldsymbol{x}_t \in \mathbb{R}^k$ is the $k$-dimensional representation for the vertex, and $\oplus$ is a concatenation operator to form the matrix $\mathcal{E}_{1:T} \in \mathbb{R}^{T \times k}$. A weighted matrix $\boldsymbol{w} \in \mathbb{R}^{l \times k}$ is applied to a convolutional operation to derive the new feature embeddings:

$$c_i = \rho\left(\boldsymbol{w} \otimes \mathcal{E}_{i:i+l-1} + b\right), \quad (8)$$

where $\otimes$ operator is the summation of element-wise production, $l$ represents the window size that corresponds to a convolution kernel, $b$ is a bias term, and $\rho(\cdot)$ is a nonlinear function.

**Algorithm 1** WalkGAN framework

**Input:** generator $G_\theta$, discriminator $D_\phi$, an information networks $W$

**Output:** vertex representations

Collect real sequences of the vertices $S$ via random walk on $W$
Randomly initialize $G_\theta$ and $D_\theta$;
Pre-train $G_\theta$ on $S$ to generate the negative samples
Pre-train $D_\theta$ by minimizing the cross entropy
**while** WalkGAN is not converged **do**
  **for** Generator training **do**
    Generate sequence $Y_{1:T} = (y_1, \ldots, y_T)$ by a generator ;
    **for** $t$ in $1 : T$ **do**
      Calculate $Q\left(s = Y_{1:t-1}, a = y_t;\right)$ according to Eq. (4);
    **end for**
    Update the generator parameters according to Eq. (6);
  **end for**
  **for** Discriminator training **do**
    Collect synthetic sequences of the vertices from $G_\theta$ as negative samples, collect real sequences of the vertices from $S$ as positive samples ;
    Update the discriminator parameters according to Eq. (10);
  **end for**
**end while**
Input the generated sequences into the Skip-gram model to derive the vertex representations.

Different features can be extracted by various kernels with different window sizes. Here, we choose $Relu(x) = Max(0, x)$ to non-linearly transform the concatenation of the original representation. Then, a maximized pooling operation is applied to derive the feature maps:

$$\tilde{c} = \max\left\{c_1, \cdots, c_{T-l+1}\right\}. \tag{9}$$

Finally, a fully-connected layer with Sigmoid activation function is used to calculate the probability that the input sequence is the real sequence of the vertices connected by the edges. An objective function is to minimize the cross-entropy between the real label and the probability, which is formulated as

$$\min_{\phi} -\mathbb{E}_{Y\sim p_{\text{data}}}\left[\log D_\phi(Y)\right] - \mathbb{E}_{Y\sim G_\theta}\left[\log\left(1 - D_\phi(Y)\right)\right], \tag{10}$$

which equals to the optimization of a discriminator.

## IV. Experiments

In this section, we evaluate the performance of WalkGAN on four real-world networks datasets. Specifically, we choose three real-world applications,i.e., link prediction, node classification and visualization to demonstrate the propose method.

### A. Dataset

We conduct experimental results on four datasets including one social networks, two citation networks and one language networks. For each scenario, we use one or two datasets to demonstrate the performance of our model. The detailed descriptions of the networks datasets are listed as follows:

- arXiv-GrQc[1] covers scientific collaborations among the authors, whose works belong to the General Relativity and Quantum Cosmology categories. This dataset contains 5,242 vertices and 14,496 edges.
- arXiv-AstroPh[2] is from the arXiv and covers scientific collaborations between authors with papers submitted to the Astro Physics category. This dataset contains 18,772 vertices and 198,110 edges.
- BlogCatalog[3] is a social network that preserves the relationships between the bloggers. The label of vertices represents the interests of bloggers which inferred by the metadata provided by the bloggers. It owns 10,312 vertices, 333,982 edges, and 39 different labels.
- Wikipedia[4] is a co-occurrence network of words which appearing in the first $10^9$ bytes of the English Wikipedia dump. Words are classified following the inferred Part-of-Speech (POS). This dataset contains 4,777 vertices, 184,812 edges, and 40 different labels.

### B. Experimental setting

To evaluate the effectiveness of those sequences of the vertices generated by WalkGAN, we design two configurations as follows:

- WalkGAN(fake). WalkGAN(fake) learns the vertex representation from those synthetic sequences of the vertices.
- WalkGAN(extra). WalkGAN(extra) combines the real sequences of the vertices and the synthetic sequences of the vertices to learn the network representations.

To measure the efficacy of the proposed method, we design a compared method, termed WalkGAN (random). Except for the training strategy of a generator, WalkGAN (random) is the same as the proposed method, where WalkGAN (random) randomly selects the next token to generate the generated sequences of the vertices.

To learn the network embedding, vertex representations are derived by inputting those sequences into Skip-gram model. Furthermore, we compare our proposed WalkGAN with the following methods:

- DeepWalk [7]. DeepWalk employs random walk to derive the sequences of the vertices and learn vertex representations from the sequences via Skipgram.
- Node2Vec [8]. Node2Vec considers the local and global structure. Compared with DeepWalk by the depth-first and breadth-first searches, Node2Vec randomly walks on the networks.
- LINE [6]. LINE learns the first-order proximity and the second-order proximity between the vertices in the networks and concatenates them together.
- SDNE [5]. SDNE uses an Auto-encoder that trained by edges to extract the topological structure of a network

[1] https://snap.stanford.edu/data/ca-GrQc.html
[2] https://snap.stanford.edu/data/ca-AstroPh.html
[3] http://socialcomputing.asu.edu/datasets/BlogCatalog
[4] http://www.mattmahoney.net/dc/textdata

| Dataset | $p$ | $l$ | $d$ |
|---|---|---|---|
| arXiv-GrQc | 20 | 60 | 64 |
| arXiv-AstroPh | 30 | 100 | 128 |
| Blocatalog | 30 | 80 | 64 |
| Wikipedia | 20 | 60 | 64 |

TABLE I. The parameter setting of $p$, $l$ and $d$.

and uses Deep Neural Network to derive the sub-linear node embeddings.

- GraphGAN [10]. GraphGAN leverages GAN to derive the vertex representations based on the neighbors of a vertex in the networks.

The key parameters of WalkGAN include the number of sequences $p$, which is initialized from a vertex, the length of each sequence $l$ and the dimension of the node embeddings $d$. For the three testing scenarios, the candidate selections of the number of sequences ( for each vertex) and the length of each sequence varies in {60,80,100,120} and {10, 20, 30, 40}, respectively. It is wise to set $p$ and $l$ to be bigger as the scale of networks grows. Table 1 lists the configuration of $p$, $l$ and $d$ used in our experiments for different real-world networks. Finally, we perform stochastic gradient descent scheme with learning rate 0.001 to update the parameters of WalkGAN.

### C. Link Prediction

In this subsection, we turn to the link prediction task and conduct two experiments on the networks dataset arXiv-GrQc and arXiv-AstroPh. The first experiment is designed to evaluate the performance of link predication for different network embedding methods. The second experiment is designed to evaluate how different percent of the missing edges affects the performance.

To conduct the first experiment, we randomly remove 10% of the existing edges and use the rest of the networks to train the networks embedding methods. After training, we derive the vertex representations from the projected space and use logistic regression to predict the unobserved links. We set the removed 10% vertex pairs in the original networks as positive samples of test set and randomly select disconnected vertex pairs with the equal number as negative samples of test set. The result of Accuracy and Macro-F1 on arXiv-GrQc and arXiv-AstroPh is listed in Table 2. As shown in Table 2, we make the following observations:

Node2Vec and DeepWalk perform the better than LINE, which is attributed that Node2Vec and DeepWalk employ random-walk to collect the context information of the networks. For Node2Vec and DeepWalk, Node2Vec is superior to DeepWalk via the biased random-walk, since Node2Vec can fuse the global and local proximity between two vertices to derive the networks embedding. GraphGAN performs slightly better than Node2Vec based on the fact that GraphGAN benefits from combining a generator and a discriminator for networks representation. SDNE outperforms the compared methods because Deep neural networks (nets) can extract the non-linear features of the networks.

The proposed WalkGAN(fake) and WalkGAN(extra) consistently and significantly outperform the baselines for link prediction. Specifically, WalkGAN(fake) improves accuracy and Macro-F1 on arXiv-GrQc by 1.8% to 3.2% and 1.4% to 4.3%, respectively; WalkGAN(extra) improves accuracy and Macro-F1 on arXiv-AstroPh by 1.7% to 3.6% and 1.1% to 2.6%, respectively. WalkGAN(random) has a poor performance for the link prediction task, which has no policy for selecting the next tokens.

| Model | arXiv-GrQc | | arXiv-AstroPh | |
|---|---|---|---|---|
| | Acc | Macro-F1 | Acc | Macro-F1 |
| WalkGAN(fake) | **0.941** | **0.940** | 0.899 | 0.906 |
| WalkGAN(extra) | 0.917 | 0.927 | **0.923** | **0.922** |
| WalkGAN(random) | 0.488 | 0.486 | 0.476 | 0.479 |
| DeepWalk | 0.803 | 0.812 | 0.841 | 0.839 |
| LINE | 0.764 | 0.761 | 0.820 | 0.814 |
| Node2Vec | 0.844 | 0.842 | 0.845 | 0.854 |
| SDNE | 0.910 | 0.897 | 0.887 | 0.896 |
| GraphGAN | 0.849 | 0.853 | 0.855 | 0.859 |

TABLE II. Accuracy and Macro-F1 on arXiv-GrQc and arXiv-AstroPh for link prediction.

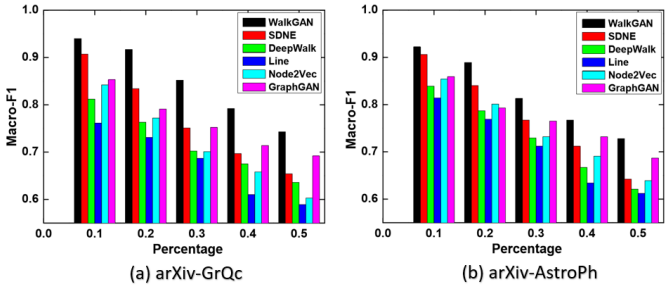



Fig. 3. Performance of network embedding of different percentages of missing edges. It shows that WalkGAN (extra) is more robust to the incomplete network.

These experimental result shows that the synthetic sequences generated by WalkGAN is much more effective than randomly generated sequences. The performance improvement of WalkGAN is attributed that WalkGAN combines random walk and GAN to generate the sequences, which captures the existing and unknown relationships between vertices in the networks and not simply considers those non-existed edges as negative samples. Thus, WalkGAN can enhance the learning performance.

For the second experiment, we randomly remove 10% to 50% of the vertices in the networks as test set and use the rest of the networks as the training samples. We observe how the learning performance of WalkGan changes when the number of missing edges increases. Fig. 3 presents the performance of networks embedding with different percentages of missing edges.

As shown in Fig. 3, WalkGAN (extra) is consistently superior to the compared methods, where the performance of WalkGAN decreases more smoothly than the other methods when the percentage of missing edges increases. The increasing of missing edges has enhanced the performance gap between WalkGAN and other methods, which fully reflects the advantages of WalkGAN for dealing with severely incomplete networks.

Note that GraphGAN is competitive to our method, which

| Model | BlogCatalog | | Wikipedia | |
|---|---|---|---|---|
| | Acc | Macro-F1 | Acc | Macro-F1 |
| WalkGAN(fake) | **0.323** | **0.251** | 0.235 | 0.217 |
| WalkGAN(extra) | 0.310 | 0.247 | **0.243** | **0.224** |
| WalkGAN(random) | 0.081 | 0.034 | 0.150 | 0.031 |
| DeepWalk | 0.225 | 0.214 | 0.194 | 0.183 |
| LINE | 0.205 | 0.192 | 0.175 | 0.164 |
| Node2Vec | 0.215 | 0.206 | 0.191 | 0.179 |
| SDNE | 0.231 | 0.220 | 0.204 | 0.190 |
| GraphGAN | 0.232 | 0.221 | 0.213 | 0.194 |

TABLE III. Accuracy, Macro-F1 on BlogCatalog and Wikipedia for node classification

is much more robust than LINE, SDNE, DeepWalk and Node2Vec. However, WalkGan still outperforms GraphGAN in a certain performance gain. The reason is that GraphGAN benefits from combining a generator and a discriminator to generate synthetic data based on the neighbors of each vertex. However, WalkGAN can generate the synthetic sequences of the vertices, which has more important supplementary effects for missing relationships in the networks, which can enhance the robustness of embedding vectors.

### *D. Node Classification*

For node classification, each vertex is identified as one of multiple categories. The vertex representations are generated via the different networks embedding methods, which is used as input of softmax regression to classify each vertex. We randomly select 90% of the vertices in the networks as training set and the rest is used as test set for employing the node classification experiments on BlogCatalog and Wikipedia. Table 3 lists the node classification results of different methods.

As shown from Table 3, compared with the other methods, the proposed WalkGAN(fake) and WalkGAN(extra) achieve the better performance. Specifically, WalkGAN(fake) achieves accuracy and Macro-F1 by 4.3% to 9.1% and by 1.9% to 3%, respectively, on BlogCatalog. WalkGAN (extra) improves accuracy and Macro-F1 by 2.1% to 3.4% and by 1.6% to 3.3%, respectively, on Wikipedia. WalkGAN (random) still has poor performance for the node classification task, which is attributed that the randomly generated sequences do not consider the potential relationships between different nodes.

### *E. Visualization*

To directly observe the performance of networks embedding, we use the representation for vertices as the input of the visualization tool t-SNE [40]. t-SNE outputs 2D feature vectors, where points can be drawn in the axis maps. We visualize the embedding results of Wikipedia, which are learned by the different methods. Because of the non-balance of the amount of vertices in various categories, we select three labels with the largest number of vertex for visualization and use the different colors to represent the categories of the vertices. Fig. 4 presents the visualization of the selected vertices in Wikipedia.

As shown from Fig. 4, the embedding results of GraphGAN, Node2Vec and DeepWalk are mixed up; and the vertex distributions learned by SDNE and LINE only form the small-scale

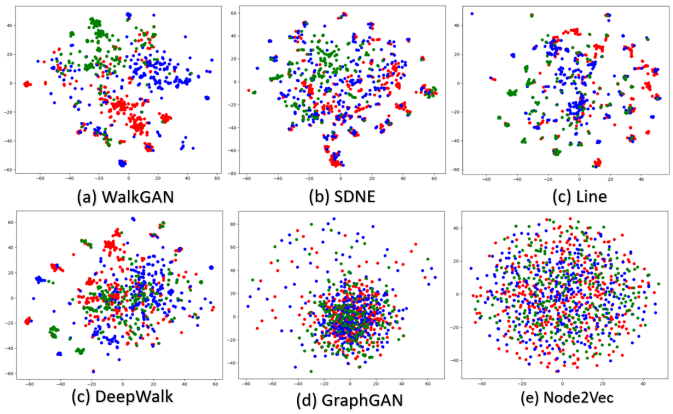



Fig. 4. Visualization of the selected vertices in Wikipedia.

| Dataset | $\alpha(sd{=}1)(\%)$ | $\alpha(sd{=}2)(\%)$ | $\alpha(sd{=}3)(\%)$ |
|---|---|---|---|
| arXiv-GrQc | 90.1 | 6.3 | 2.4 |
| arXiv-AstroPh | 92.1 | 7.5 | 0.1 |
| Blocatalog | 91.3 | 5.7 | 1.6 |
| Wikipedia | 90.8 | 6.1 | 1.9 |

TABLE IV. The connection distribution of adjacent nodes in the generated sequences

clusters, which are relatively scattered. However, the vertex distribution learned by WalkGAN (extra) makes most of the vertices with the same color are close together, resulting in the obvious clusters.

The experimental results demonstrate that WalkGAN is superior to other methods for capturing the neighborhood information of the networks. The superior performance of WalkGAN is attributed that the proposed method can not only capture the structure feature of the original networks, but also infer the possible associations between the paired vertices. Thus, the impact of the missing edges on networks embedding is reduced.

### *F. Empirical Study*

In this subsection, we conduct an empirical study to investigate the connection distribution of adjacent vertices in the generated sequences by collecting the sequences generated by a generator of WalkGAN. For the adjacent vertex pairs in the generated sequences, we compute both the shortest distance ($sd$) in a network and the proportion of vertex pairs with different shortest distances, where $\alpha(sd{=}1)$, $\alpha(sd{=}2)$ and $\alpha(sd{=}3)$ mean the percents of vertex pairs with $sd{=}1$, $sd{=}2$ and $sd{=}3$, respectively. Table 4 lists the shortest distances and the proportion of vertex pairs with different shortest distances.

As shown in Table 4, at least 90% of the adjacent vertices in the generated paths owns $sd{=}1$, which means these vertices are directly connected in a network. Thus, the generated sequences have the high similarity to the real paths. Moreover, the probability of link existence between vertex pair drops dramatically as the shortest distance increases . Note that the shortest distance of adjacent vertex pairs in the sequences generated by WalkGAN is mostly within from one to three,

which indicates that WalkGAN can infer the unobserved links with a high probability.

## V. Conclusion

In this paper, a networks representation learning method, namely WalkGAN, has been proposed. WalkGAN combines random walk and GAN to generate the real sequences and synthetic sequences of the vertices , where the synthetic sequences contain the context information of a network and meantime infer the unknown relationships between the vertices. Thus, WalkGAN can not only eliminate the impact of the missing edges on networks embedding, but also improve the robustness of embedding vectors. The comparison of the different variants of the proposed model: Walk(extra) and WalkGAN(fake) demonstrate the effectiveness of WalkGAN. For the future work, we plan to extend our model to the other graph neural networks for further enhancing performance.

## References

[1] P. Cui, X. Wang, J. Pei, and W. Zhu, "A survey on network embedding," *IEEE Trans. Knowl. Data Eng.*, vol. 31, no. 5, pp. 833–852, 2019.

[2] Y. Luo, X. Zhao, J. Zhou, J. Yang, Y. Zhang, W. Kuang, J. Peng, L. Chen, and J. Zeng, "A network integration approach for drug-target interaction prediction and computational drug repositioning from heterogeneous information," in *RECOMB*, 2017.

[3] W. Li, Z. Ye, and Q. Jin, "An integrated recommendation approach based on influence and trust in social networks," in *Future Information Technology*, 2014, pp. 83–89.

[4] J. Menche, A. Sharma, M. Kitsak, S. D. Ghiassian, M. Vidal, J. Loscalzo, and A.-L. Barabási, "Uncovering disease-disease relationships through the incomplete interactome," *Science*, vol. 347, no. 6224, 2015.

[5] D. Wang, P. Cui, and W. Zhu, "Structural deep network embedding," in *KDD*, 2016.

[6] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "LINE: large-scale information network embedding," in *WWW*, 2015.

[7] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: online learning of social representations," in *KDD*, 2014.

[8] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *KDD*, 2016.

[9] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. C. Courville, and Y. Bengio, "Generative adversarial nets," in *NeurIPS*, 2014.

[10] 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.

[11] A. Bojchevski, O. Shchur, D. Zügner, and S. Günnemann, "Netgan: Generating graphs via random walks," in *ICML*, 2018.

[12] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein generative adversarial networks," in *ICML*, 2017.

[13] L. F. R. Ribeiro, P. H. P. Saverese, and D. R. Figueiredo, "*struc2vec*: Learning node representations from structural identity," in *KDD*, 2017.

[14] H. Chen, B. Perozzi, Y. Hu, and S. Skiena, "HARP: hierarchical representation learning for networks," in *AAAI*, 2018.

[15] B. Perozzi, V. Kulkarni, H. Chen, and S. Skiena, "Don't walk, skip!: Online learning of multi-scale network embeddings," in *ASONAM 2017*, J. Diesner, E. Ferrari, and G. Xu, Eds., 2017, pp. 258–265.

[16] S. Cao, W. Lu, and Q. Xu, "Deep neural networks for learning graph representations," in *AAAI*, 2016.

[17] A. Hasanzadeh, E. Hajiramezanali, K. R. Narayanan, N. Duffield, M. Zhou, and X. Qian, "Semi-implicit graph variational auto-encoders," in *NeurIPS*, 2019.

[18] N. Sheikh, Z. T. Kefato, and A. Montresor, "gat2vec: representation learning for attributed graphs," *Computing*, vol. 101, no. 3, pp. 187–209, 2019.

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

[20] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *NeurIPS*, 2016.

[21] R. Levie, F. Monti, X. Bresson, and M. M. Bronstein, "Cayleynets: Graph convolutional neural networks with complex rational spectral filters," *IEEE Trans. Signal Process.*, vol. 67, no. 1, pp. 97–109, 2019.

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

[23] C. Zhuang and Q. Ma, "Dual graph convolutional networks for graph-based semi-supervised classification," in *WWW*, 2018.

[24] A. Micheli, "Neural network for graphs: A contextual constructive approach," *IEEE Trans. Neural Networks*, vol. 20, no. 3, pp. 498–511, 2009.

[25] M. Niepert, M. Ahmed, and K. Kutzkov, "Learning convolutional neural networks for graphs," in *ICML*, 2016.

[26] J. Atwood and D. Towsley, "Diffusion-convolutional neural networks," in *NeurIPS*, 2016.

[27] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *ICLR*, 2018.

[28] F. Monti, D. Boscaini, J. Masci, E. Rodolà, J. Svoboda, and M. M. Bronstein, "Geometric deep learning on graphs and manifolds using mixture model cnns," in *CVPR*, 2017.

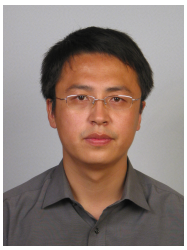
[29] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *ICML*, 2017.

[30] H. Gao, Z. Wang, and S. Ji, "Large-scale learnable graph convolutional networks," in *KDD*, 2018.

[31] P. Velickovic, W. Fedus, W. L. Hamilton, P. Liò, Y. Bengio, and R. D. Hjelm, "Deep graph infomax," in *ICLR*, 2019.

[32] L. Yu, W. Zhang, J. Wang, and Y. Yu, "Seqgan: Sequence generative adversarial nets with policy gradient,"

in *AAAI*, 2017.
[33] R. S. Sutton, A. G. Barto *et al.*, *Introduction to reinforcement learning*. MIT press Cambridge, 1998, vol. 2, no. 4.
[34] C. P. Robert and G. Casella, *Monte Carlo Statistical Methods*, 2004.
[35] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.
[36] K. Veselý, A. Ghoshal, L. Burget, and D. Povey, "Sequence-discriminative training of deep neural networks," in *INTERSPEECH*, 2013.
[37] Y. Kim, "Convolutional neural networks for sentence classification," in *EMNLP*, 2014.
[38] G. Arevian, "Recurrent neural networks for robust real-world text classification," in *WI*, 2007.
[39] J. Y. Lee and c. Franck Dernon, "Sequential short-text classification with recurrent and convolutional neural networks," in *NAACL*, 2016.
[40] L. v. d. Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of machine learning research*, vol. 9, pp. 2579–2605, 2008.



**Taisong Jin** received the Ph.D. degree in computer science from Beijing Institute of Technology, China, in 2007. From August, 2015 to September,2016, as a visiting scholar, he visited the Department of Electrical Engineering, Korea Advanced Institute of Science and Technology (Kaist). He is currently an assistant Professor in the Department of Computer Science, Xiamen University, China. His research interests include machine learning and computer vision.

**Gengchen Duan** received a Bachelor's degree in computer science from Xiamen University, Xiamen, China. Currently, He is pursue a master's degree in Xiamen University . His research interests include graph neural network and computer vision.

**ZhengTao Yu** received his Ph.D. degree in computer application technology from Beijing Institute of Technology, Beijing, China, in 2005. He is currently a professor in the School of Information Engineering and Automation, Kunming University of Science and Technology, China. His main research interests include natural language processing, information retrieval and machine learning.

**Han Luo** received a master's degree in computer science from Xiamen University, Xiamen, China. She received a bachelor's degree in Software Engineering from Sichuan Normal University and received National Inspirational Scholarships twice and Multiple Academic Scholarships. In 2020, she joined Shenzhen Securities Communication Co., Ltd, which is responsible for the communication technology of the Shenzhen Stock Exchange. Her current research interests include graph neural network, biocomputing, and bioinformatics.

**Xiangxiang Zeng** (M'11–SM'16) received the Ph.D. degree in system engineering from the Huazhong University of Science and Technology, Wuhan, China, in 2011. He was with the Department of Computer Science, Xiamen University, Xiamen, China. In 2019, he joined Hunan University, Changsha, China, where he is a Yuelu Distinguished Professor with the College of Information Science and Engineering. He was a Visiting Scholar with Oklahoma State University, Stillwater, OK, USA; and Harvard Medical School, Boston, MA, USA. He has published over 100 papers on journals and conferences. He has over 4000 Google Scholar citations. His current research interests include computational intelligence, graph neural networks, and bioinformatics.

**Min Jiang** (M'11-SM'12) received his bachelor and Ph.D. degrees in computer science from Wuhan University, China, in 2001 and 2007, respectively. Subsequently as a postdoc in the Department of Mathematics of Xiamen University. Currently he is a professor in the Department of Artificial Intelligence, Xiamen University. His main research interests are Machine Learning, Computational Intelligence and Robotics. He has a special interest in dynamic multiobjective optimization, transfer learning, the software development and in the basic theories of robotics. In 2016, he received the Outstanding Reviewer award from *IEEE Transactions on Cybernetics*. Dr. Jiang is currently serving as associate editors for the *IEEE Transactions on Neural Networks and Learning Systems* and *IEEE Transactions on Cognitive and Developmental Systems* and he is the Chair of IEEE CIS Xiamen Chapter.