Learning Deep Network Representations with Adversarially Regularized Autoencoders*

Wenchao Yu[1], Cheng Zheng[1], Wei Cheng[2], Charu C. Aggarwal[3], Dongjin Song[2], Bo Zong[2], Haifeng Chen[2], and Wei Wang[1]
[1]Department of Computer Science, University of California Los Angeles
[2]NEC Laboratories America, Inc. [3]IBM Research AI
{wenchaoyu,chengzheng,weiwang}@cs.ucla.edu,{weicheng,bzong,dsong,haifeng}@nec-labs.com,charu@us.ibm.com

ABSTRACT

The problem of network representation learning, also known as network embedding, arises in many machine learning tasks assuming that there exist a small number of variabilities in the vertex representations which can capture the "semantics" of the original network structure. Most existing network embedding models, with shallow or deep architectures, learn vertex representations from the sampled vertex sequences such that the low-dimensional embeddings preserve the locality property and/or global reconstruction capability. The resultant representations, however, are difficult for model generalization due to the intrinsic sparsity of sampled sequences from the input network. As such, an ideal approach to address the problem is to generate vertex representations by learning a probability density function over the sampled sequences. However, in many cases, such a distribution in a low-dimensional manifold may not always have an analytic form. In this study, we propose to learn the network representations with adversarially regularized autoencoders (NetRA). NetRA learns smoothly regularized vertex representations that well capture the network structure through jointly considering both locality-preserving and global reconstruction constraints. The joint inference is encapsulated in a generative adversarial training process to circumvent the requirement of an explicit prior distribution, and thus obtains better generalization performance. We demonstrate empirically how well key properties of the network structure are captured and the effectiveness of NetRA on a variety of tasks, including network reconstruction, link prediction, and multi-label classification.

KEYWORDS

Network embedding, autoencoder, generative adversarial networks

ACM Reference Format:
Wenchao Yu[1], Cheng Zheng[1], Wei Cheng[2], Charu C. Aggarwal[3], Dongjin Song[2], Bo Zong[2], Haifeng Chen[2], and Wei Wang[1]. 2018. Learning Deep Network Representations with Adversarially Regularized Autoencoders. In *KDD '18: The 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, August 19–23, 2018, London, United Kingdom.* ACM, New York, NY, USA, Article 4, 9 pages. https://doi.org/10.1145/3219819.3220000

*Wei Cheng, Haifeng Chen and Wei Wang are corresponding authors.

KDD '18, August 19–23, 2018, London, United Kingdom

ACM ISBN 978-1-4503-5552-0/18/08…$15.00
https://doi.org/10.1145/3219819.3220000

1 INTRODUCTION

Network analysis has been attracting many research interests with its enormous potential in mining useful information which benefits the downstream tasks such as link prediction, community detection and anomaly detection on social network [34], biological networks [31] and language networks [28], to name a few.

To analyze network data, one fundamental problem is to learn a low-dimensional vector representation for each vertex, such that the network structure is preserved in the learned vector space [23]. For this problem, there are two major challenges: (1) *preservation of complex structure property.* The objective of network embedding is to train a model to "fit" the training networks, that is, to preserve the structure property of networks [23, 26]. However, the latent structure of the network is too complex to be portrayed by an explicit form of probability density which can capture both the local network neighborhood information and global network structure. (2) *sparsity of network sampling.* Current research on network embedding employs network sampling techniques, including random walk sampling, breadth-first search etc., to derive vertex sequences as training datasets. However the sampled data represent only a small proportion of all the vertex sequences. An alternative approach is to encode these discrete structures in a continuous code space [37]. Unfortunately, learning continues latent representations of discrete networks remains a challenging problem since in many cases, the prior distribution may not exist in a low dimensional manifold [26].

Recent work on network embedding has shown fruitful progress in learning vertex representations of complex networks [23, 26, 37]. These representations employ nonlinear transformations to capture the "semantics" of the original networks. Most existing methods first employ a random walk technique to sample a bunch of vertex sequences from the input network, then feed a learning model with these sequences to infer the optimal low-dimensional vertex embeddings. However, the sampling strategy suffers from the data sparsity problem since the total amount of vertex sequences is usually very large in real networks and it is often intractable to enumerate all. Subsequently, learning on a sparse sample set tends to produce an overly complex model to explain the sampled dataset, which eventually causes overfitting. Though autoencoders are adopted to encode the inputs into continuous latent representations [37], regularizations are still desirable to force the learned representations

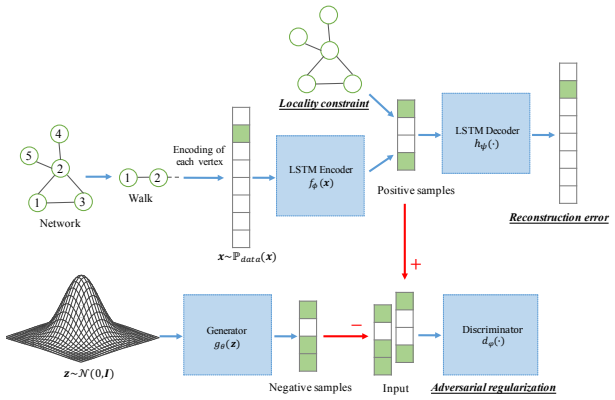


Figure 1: Illustration of the deep network embedding architecture with adversarially regularized autoencoders

remain on the latent manifold. Ideally we could generate the continuous vertex representations with a prior distribution. However, in many cases, it is difficult, if not impossible, to pre-define an explicit form of the prior distribution in a low-dimensional manifold. For example, Dai et al. [6] proposed to train a discriminator to distinguish samples generated from a fixed prior distribution and the input encoding, and thereby pushing the embedding distribution to match the fixed prior. While this gives more flexibility, it suffers from the mode-collapse problem [16]. Moreover, most network embedding models with deep architectures usually do not consider the order of the vertices in the sampled vertex sequences [37]. Thus, the information of proximity orders cannot be well considered.

To address the aforementioned challenges, in this study, we propose a novel model to learn the network representations with adversarially regularized autoendoers (NetRA). NetRA jointly minimizes network locality-preserving loss and the reconstruction error of autoencoder which utilizes a long short-term memory network (LSTM) as an encoder to map the input sequences into a fixed length representation. The joint embedding inference is encapsulated in a generative adversarial training process to circumvent the requirement of an explicit prior distribution. As visually depicted in Figure 1, our model employs a discrete LSTM autoencoder to learn continuous vertex representations with sampled sequences of vertices as inputs. In this model, besides minimizing the reconstruction error in the LSTM autoencoder, the locality-preserving loss at the hidden layer is also minimized simultaneously. Meanwhile, the continuous space generator is also trained by constraining to agree in distribution with the encoder. The generative adversarial training can be regarded as a complementary regularizer to the network embedding process.

NetRA exhibits desirable properties that a network embedding model requires: 1) *structure property preservation*, NetRA leverages LSTM as an encoder to capture the neighborhood information among vertices in each sequence sampled from the network. Additionally, the model is also trained simultaneously with the locality-preserving constraint. 2) *generalization capability*, the generalization capability requires a network embedding model to generalize well on unseen vertex sequences which follow the same distribution as the population. The generative adversarial training process enables the proposed model to learn smoothly regularized representations without pre-defining an explicit density distribution which overcomes the sparsity issue from the input sequences of vertices. We present experimental results to show the embedding capability of NetRA on a variety of tasks, including network reconstruction, link prediction and multi-label classification. To summarize, the main contributions of this work are as follows:

- We propose a novel deep network embedding model with adversarially regularized autoencoders, NetRA, to learn vertex representations by jointly minimizing locality-preserving loss and global reconstruction error using generative adversarial training process. The resultant representations are robust to the sparse inputs derived from the network.
- NetRA learns an adversarially regularized LSTM encoder that can produce useful vertex representations from discrete inputs, without a pre-defined explicit latent-space prior.
- We conduct extensive experiments on tasks of network reconstruction, link prediction and multi-label classification using real-world information networks. Experimental results demonstrate the effectiveness and efficiency of NetRA.

The rest of this paper is organized as follows. In Section 2, we review the preliminary knowledge of autoencoders, generative adversarial networks and network embedding algorithms. In Section 3, we describe NetRA framework of learning a low dimensional mapping with generative adversarial training process. In Section 4, we demonstrate the performance of NetRA by adapting this joint learning framework on tasks of network reconstruction, link prediction and multi-label classification. In Section 5, we compare NetRA framework to other network embedding algorithms and discuss several related work. Finally, in Section 6 we conclude this study and mention several directions for future work.

2 PRELIMINARIES

2.1 Autoencoder Neural Networks

An autoencoder neural network is trained to set the target values to be equal to the inputs. The network consists of two parts: an encoder $f_\phi(\cdot)$ that maps inputs ($\mathbf{x} \in \mathbb{R}^n$) to latent low-dimensional representations and a decoder $h_\psi(\cdot)$ that produces a reconstruction of the inputs. Specifically, given a data distribution $\mathbb{P}_{data}$, from which $\mathbf{x}$ is drawn from, i.e., $\mathbf{x} \sim \mathbb{P}_{data}(\mathbf{x})$, we want to learn representations $f_\phi(\mathbf{x})$ such that the output hypotheses $h_\psi(f_\phi(\mathbf{x}))$ is approximately equal to $\mathbf{x}$. The learning process is described simply as minimizing a cost function

$$\min \mathbb{E}_{\mathbf{x}\sim\mathbb{P}_{data}(\mathbf{x})}[\text{dist}(\mathbf{x}, h_\psi(f_\phi(\mathbf{x})))], \tag{1}$$

where $\text{dist}(\cdot)$ is some similarity metric in the data space. In practice, there are many options for the distance measure. For example, if we use ℓ_2 norm to measure the reconstruction error, then the objective function can be defined as $\mathcal{L}_{\text{AE}}(\phi, \psi; \mathbf{x}) = \mathbb{E}_{\mathbf{x}\sim\mathbb{P}_{data}(\mathbf{x})}\|\mathbf{x} - h_\psi(f_\phi(\mathbf{x}))\|^2$. Similarly the objective function for cross-entropy loss can be defined as,

$$-\mathbb{E}_{\mathbf{x}\sim\mathbb{P}_{data}(\mathbf{x})}[\mathbf{x}\log h_\psi(f_\phi(\mathbf{x})) + (\mathbf{1}-\mathbf{x})\log(\mathbf{1} - h_\psi(f_\phi(\mathbf{x})))] \tag{2}$$

The choice of encoder $f_\phi(\cdot)$ and decoder $h_\psi(\cdot)$ may vary across different tasks. In this paper, we use LSTM autoencoders [27] which are capable of dealing with sequences as inputs.

2.2 Generative Adversarial Networks

The Generative Adversarial Networks (GANs) [11] build an adversarial training platform for two players, namely *generator* $g_\theta(\cdot)$ and *discriminator* $d_w(\cdot)$, to play a minimax game.

$$\min_\theta \max_w \mathop{\mathbb{E}}_{\mathbf{x}\sim\mathbb{P}_{data}(\mathbf{x})}[\log d_w(\mathbf{x})] + \mathop{\mathbb{E}}_{\mathbf{z}\sim\mathbb{P}_g(\mathbf{z})}[\log(1 - d_w(g_\theta(\mathbf{z})))] \quad (3)$$

The *generator* $g_\theta(\cdot)$ tries to map the noise to the input space as closely as the true data, while the *discriminator* $d_w(\mathbf{x})$ represents the probability that $\mathbf{x}$ came from the data rather than the noise. It aims to distinguish *real* data distribution $\mathbb{P}_{data}(\mathbf{x})$ and *fake* sample distribution $\mathbb{P}_g(\mathbf{z})$, e.g. $\mathbf{z} \sim \mathcal{N}(0, \mathbb{I})$. Wasserstein GANs [1] overcome unstable training problem by replacing Jensen-Shannon divergence with Earth-Mover (Wasserstein-1) distance, which considers solving the problem

$$\min_\theta \max_{w\in\mathcal{W}} \mathop{\mathbb{E}}_{\mathbf{x}\sim\mathbb{P}_{data}(\mathbf{x})}[d_w(\mathbf{x})] - \mathop{\mathbb{E}}_{\mathbf{z}\sim\mathbb{P}_g(\mathbf{z})}[d_w(g_\theta(\mathbf{z}))]. \quad (4)$$

The Lipschitz constraint $\mathcal{W}$ on discriminator has been kept by clipping the weights of the discriminator within a compact space [-c, c].

2.3 Network Embedding

Network embedding approaches seek to learn representations that encode structural information about the network. These approaches learn a mapping that embeds vertices as points into a low-dimensional space. Given the encoded vertex set $\{\mathbf{x}^{(1)}, ..., \mathbf{x}^{(n)}\}$, finding an embedding $f_\phi(\mathbf{x}^{(i)})$ of each $\mathbf{x}^{(i)}$ can be formalized as an optimization problem [39, 41]

$$\min_\phi \sum_{1\le i<j\le n} L(f_\phi(\mathbf{x}^{(i)}), f_\phi(\mathbf{x}^{(j)}), \varphi_{ij}), \quad (5)$$

where $f_\phi(\mathbf{x}) \in \mathbb{R}^d$ is the embedding result for a given input $\mathbf{x}$. $L(\cdot)$ is the loss function between a pair of inputs. φ_{ij} is the weight between $\mathbf{x}^{(i)}$ and $\mathbf{x}^{(j)}$.

We consider the Laplacian Eigenmaps (LE) that well fits the framework. LE enables the embedding to preserve the locality property of network structure. Formally, the embedding can be obtained by minimizing the following objective function

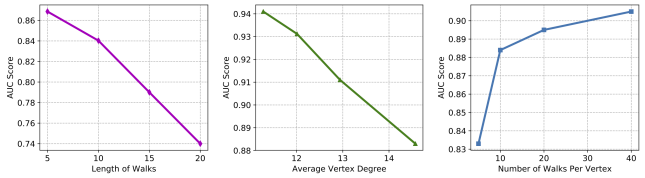
$$\mathcal{L}_{LE}(\phi; \mathbf{x}) = \sum_{1\le i<j\le n} \|f_\phi(\mathbf{x}^{(i)}) - f_\phi(\mathbf{x}^{(j)}\|^2 \varphi_{ij}. \quad (6)$$

3 APPROACH

In this section, we present NetRA, a deep network embedding model using adversarially regularized autoencoders, to learn smoothly regularized vertex representations with sequences of vertices as inputs. The resultant representations can be used in the downstream tasks, such as link prediction, network reconstruction and multi-class classification.

3.1 Random Walk Generator

Given network $G(V, E)$, the random walk generator in DeepWalk [23] is utilized to obtain truncated random walks (i.e. sequences of vertices) rooted on each vertex $v \in V$ in $G(V, E)$. A walk is sampled randomly from the neighbors of the last visited vertex until the preset maximum length is reached.



(a) Walk length **(b) Vertex degree** **(c) Number of samples**

Figure 2: Sparsity of network sampling.

The random walk sampling technique is widely adopted in network embedding research [12, 23, 37]. However, it suffers from the sparsity problem in network sampling. For each vertex in given network, if we assume that the average node degree is $\bar{d}$, the walk length is l and the number of samples is k, then the sampling fraction of walks can be calculated by

$$p_{frac} \propto \frac{|V| \times k}{|V| \times \bar{d}^l} = \frac{k}{\bar{d}^l} \times 100\%. \quad (7)$$

The effect of the sampling fraction is presented in Figure 2. In the example, DeepWalk is used to perform link prediction task on the UCI message network described in Section 4.1. Figure 2(a) and Figure 2(b) show that if the walk length or the average vertex degree increases, the performance decreases dramatically[1]. According to Eq. (7), obviously, when l or $\bar{d}$ increases, the sampling fraction of walks is getting smaller. Thereby, the trained model is prone to overfitting because of the sparse inputs. On the contrary, if the number of samples k increases, the performance is getting better as shown in Figure 2(c). However, more sampled walks also call for more computing burden on model training. Therefore, it is desirable to develop effective models with better capabilities of generalization on sparsely sampled network walks.

3.2 Embedding with Adversarially regularized Autoencoders

In this paper, we propose NetRA, a network embedding model with adversarially regularized autoencoders, to address the sparsity problem. Autoencoders are popularly used for data embedding, such as images and documents. It provides informative low dimensional representations of input data by mapping the them to the latent space. Unfortunately, if the encoder and decoder are allowed too much capacity, the autoencoder can learn to perform the copying task without extracting useful information about the distribution of the data [10]. We proposed to use a generative adversarial training process as a complementary regularizer. The process has two advantages. On one hand, the regularizer can guide the extraction of useful information about data [10]. On the other hand, the generative adversarial training provides more robust discrete-space representation learning that can well address the overfitting problem on sparsely sampled walks [19]. Specifically, in NetRA, the discriminator updates by comparing the samples from the latent space of the autoencoder with the fake samples from the generator,

[1] In Figure 2(a), the window size of DeepWalk is set to be equal to the walk length. The reason is that, if the window size is set to a small value against a long walk length, it turns out to be equivalent to increase the samples per vertex with a short walk length. In Figure 2(b), we reduce the degree of the dataset by removing vertices with large degrees [2].

as shown in Figure 1. The latent space of autoencoder provides optimal embedding for the vertices in the network with the simultaneous update of encoder and discriminator. In this study, we use the LSTM as the encoder and decoder networks [27] because it takes the order information of the sampled walks into consideration.

This joint architecture requires dedicated training objective for each part. The autoencoder can be trained individually by minimizing the negative log-likelihood of reconstruction, which is indicated by cross entropy loss in the implementation

$$\mathcal{L}_{\text{AE}}(\phi, \psi; \mathbf{x}) = -\mathbb{E}_{\mathbf{x}\sim\mathbb{P}_{data}(\mathbf{x})}[\text{dist}(\mathbf{x}, h_\psi(f_\phi(\mathbf{x})))], \tag{8}$$

where $\text{dist}(\mathbf{x}, \mathbf{y}) = \mathbf{x}\log\mathbf{y} + (\mathbf{1}-\mathbf{x})\log(\mathbf{1}-\mathbf{y})$. Here $\mathbf{x}$ is the sampled batch from training data. $f_\phi(\mathbf{x})$ is embedded latent representation of $\mathbf{x}$, which is also the positive samples for discriminator, indicated by the arrow with "+" in Figure 1. ϕ and ψ are parameters of the encoder and decoder functions, respectively. In the training iteration of autoencoder, not only the encoder and decoder are updated, the locality-preserving loss (Eq. (6)) is jointly minimized.

As depicted in Figure 1, NetRA minimizes the distributions between the learned representations from the encoder function $f_\phi(\mathbf{x}) \sim \mathbb{P}_\phi(\mathbf{x})$, and the representations from the continuous generator model $g_\theta(\mathbf{z}) \sim \mathbb{P}_\theta(\mathbf{z})$. The dual form of the Earth Mover distance between $\mathbb{P}_\phi(\mathbf{x})$ and $\mathbb{P}_\theta(\mathbf{z})$ can be described as follows [1]

$$W(\mathbb{P}_\phi(\mathbf{x}), \mathbb{P}_\theta(\mathbf{z})) = \sup_{\|d(\cdot)\|_{L\le 1}} \mathbb{E}_{\mathbf{y}\sim\mathbb{P}_\phi(\mathbf{x})}[d(\mathbf{y})] - \mathbb{E}_{\mathbf{y}\sim\mathbb{P}_\theta(\mathbf{z})}[d(\mathbf{y})] \tag{9}$$

where $\|d(\cdot)\|_{L\le 1}$ is the Lipschitz continuity constraint (with Lipschitz constant 1). If we have a family of functions $\{d_w(\cdot)\}_{w\in\mathcal{W}}$ that are all K-Lipschitz for some K, then we have

$$W(\mathbb{P}_\phi(\mathbf{x}), \mathbb{P}_\theta(\mathbf{z})) \propto \max_{w\in\mathcal{W}} \mathbb{E}_{\mathbf{x}\sim\mathbb{P}_{data}(\mathbf{x})}[d_w(f_\phi(\mathbf{x}))] - \mathbb{E}_{\mathbf{z}\sim\mathbb{P}_g(\mathbf{z})}[d_w(g_\theta(\mathbf{z}))] \tag{10}$$

We can separate the training of generator and discriminator. As for the generator, the cost function can be defined as,

$$\mathcal{L}_{\text{GEN}}(\theta; \mathbf{x}, \mathbf{z}) = \mathbb{E}_{\mathbf{x}\sim\mathbb{P}_{data}(\mathbf{x})}[d_w(f_\phi(\mathbf{x}))] - \mathbb{E}_{\mathbf{z}\sim\mathbb{P}_g(\mathbf{z})}[d_w(g_\theta(\mathbf{z}))] \tag{11}$$

and the cost function for discriminator is,

$$\mathcal{L}_{\text{DIS}}(w; \mathbf{x}, \mathbf{z}) = -\mathbb{E}_{\mathbf{x}\sim\mathbb{P}_{data}(\mathbf{x})}[d_w(f_\phi(\mathbf{x}))] + \mathbb{E}_{\mathbf{z}\sim\mathbb{P}_g(\mathbf{z})}[d_w(g_\theta(\mathbf{z}))] \tag{12}$$

NetRA learns smooth representations by jointly minimizing the autoencoder reconstruction error and the locality-preserving loss in an adversarial training process. Specifically, we consider solving the joint optimization problem with objective function

$$\mathcal{L}_{\text{NetRA}}(\phi, \psi, \theta, w) = \mathcal{L}_{\text{AE}}(\phi, \psi; \mathbf{x}) + \lambda_1\mathcal{L}_{\text{LE}}(\phi; \mathbf{x}) + \lambda_2 W(\mathbb{P}_\phi(\mathbf{x}), \mathbb{P}_\theta(\mathbf{z})) \tag{13}$$

Theorem 3.1. *Let $\mathbb{P}_\phi(\mathbf{x})$ be any distribution. Let $\mathbb{P}_\theta(\mathbf{z})$ be the distribution of $g_\theta(\mathbf{z})$ with $\mathbf{z}$ being a sample drawn from distribution $\mathbb{P}_g(\mathbf{z})$ and $g_\theta(\cdot)$ being a function satisfying the local Lipschitz constants $\mathbb{E}_{\mathbf{z}\sim\mathbb{P}_g(\mathbf{z})}[L(\theta, \mathbf{z})] < +\infty$. Then we have*

$$\begin{aligned}
\nabla_\theta\mathcal{L}_{NetRA} &= -\lambda_2\nabla_\theta\mathbb{E}_{\boldsymbol{z}\sim\mathbb{P}_g(\mathbf{z})}[d_w(g_\theta(\boldsymbol{z}))] &(14)\\
\nabla_w\mathcal{L}_{NetRA} &= -\lambda_2\nabla_w\mathbb{E}_{\boldsymbol{x}\sim\mathbb{P}_{data}(\boldsymbol{x})}[d_w(f_\phi(\boldsymbol{x}))] \\
&\quad +\lambda_2\nabla_w\mathbb{E}_{\boldsymbol{z}\sim\mathbb{P}_g(\mathbf{z})}[d_w(g_\theta(\boldsymbol{z}))] &(15)\\
\nabla_\phi\mathcal{L}_{NetRA} &= \lambda_1\nabla_\phi \sum_{1\le i<j\le n} \|f_\phi(\mathbf{x}^{(i)}) - f_\phi(\mathbf{x}^{(j)})\|^2\varphi_{ij} \\
&\quad -\nabla_\phi\mathbb{E}_{\mathbf{x}\sim\mathbb{P}_{data}(\mathbf{x})}[dist(\mathbf{x}, h_\psi(f_\phi(\mathbf{x})))] \\
&\quad +\lambda_2\nabla_\phi\mathbb{E}_{\boldsymbol{x}\sim\mathbb{P}_{data}(\boldsymbol{x})}[d_w(f_\phi(\boldsymbol{x}))] &(16)\\
\nabla_\psi\mathcal{L}_{NetRA} &= -\nabla_\psi\mathbb{E}_{\mathbf{x}\sim\mathbb{P}_{data}(\mathbf{x})}[dist(\mathbf{x}, h_\psi(f_\phi(\mathbf{x})))] &(17)
\end{aligned}$$

Proof. Let $\mathcal{X} \subseteq \mathbb{R}^n$ be a compact set, and

$$\begin{aligned}
V(\tilde{d}, \theta) &= \mathbb{E}_{\mathbf{y}\sim\mathbb{P}_\phi(\mathbf{x})}[\tilde{d}(\mathbf{y})] - \mathbb{E}_{\mathbf{y}\sim\mathbb{P}_\theta(\mathbf{z})}[\tilde{d}(\mathbf{y})] \\
&= \mathbb{E}_{\mathbf{y}\sim\mathbb{P}_\phi(\mathbf{x})}[\tilde{d}(\mathbf{y})] - \mathbb{E}_{\mathbf{z}\sim\mathbb{P}_g(\mathbf{z})}[\tilde{d}(g_\theta(\mathbf{z}))] &(18)
\end{aligned}$$

where $\tilde{d}$ lies in $\mathcal{D} = \{\tilde{d} : \mathcal{X} \to \mathbb{R}, \tilde{d} \text{ is continuous and bounded}, \|\tilde{d}\| \le 1\}$. Since $\mathcal{X}$ is compact, we know by the Kantorovich-Rubinstein duality [1] that there exists a $d \in \mathcal{D}$ that attains the value

$$W(\mathbb{P}_\phi(\mathbf{x}), \mathbb{P}_\theta(\mathbf{z})) = \sup_{\tilde{d}\in\mathcal{D}} V(\tilde{d}, \theta) = V(d, \theta) \tag{19}$$

and $D^*(\theta) = \{d \in \mathcal{D} : V(d, \theta) = W(\mathbb{P}_\phi(\mathbf{x}), \mathbb{P}_\theta(\mathbf{z}))\}$ is non-empty. According to the envelope theorem [21], we have

$$\nabla_\theta W(\mathbb{P}_\phi(\mathbf{x}), \mathbb{P}_\theta(\mathbf{z})) = \nabla_\theta V(d, \theta) \tag{20}$$

for any $d \in D^*(\theta)$. Then we get

$$\begin{aligned}
\nabla_\theta W(\mathbb{P}_\phi(\mathbf{x}), \mathbb{P}_\theta(\mathbf{z})) &= \nabla_\theta V(d, \theta) \\
&= \nabla_\theta\mathbb{E}_{\mathbf{y}\sim\mathbb{P}_\phi(\mathbf{x})}[d(\mathbf{y})] - \mathbb{E}_{\mathbf{z}\sim\mathbb{P}_g(\mathbf{z})}[d(g_\theta(\mathbf{z}))] \\
&= -\nabla_\theta\mathbb{E}_{\mathbf{z}\sim\mathbb{P}_g(\mathbf{z})}[d_w(g_\theta(\mathbf{z}))] &(21)
\end{aligned}$$

Therefore, we have $\nabla_\theta\mathcal{L}_{\text{NetRA}} = -\lambda_2\nabla_\theta\mathbb{E}_{\mathbf{z}\sim\mathbb{P}_g(\mathbf{z})}[d_w(g_\theta(\mathbf{z}))]$.

Eq.(15)-(17) are straightforward applications of the derivative definition. □

We now have all the derivatives needed. To train the model, we use a block coordinate descent to alternate between optimizing different parts of the model: (1) locality-preserving loss and autoencoder reconstruction error (update ϕ and ψ), (2) the discriminator in the adversarial training process (update w), and (3) the generator (update θ). Pseudocode of the full approach is given in Algorithm 1.

The training process of NetRA consists of the following steps: Firstly, given a network $G(V, E)$, we run random walk generator acquiring random walks of length l. Then, one hot representation $\mathbf{x}^{(i)}$ of each vertex is taken as input to LSTM cells. We pass the random walks through encoding layers and obtain the vector representations of vertices. After the decoder network, the vertex representations will be transformed back into n dimensions. Cross-entropy loss is calculated between the inputs and outputs by minimizing the reconstruction error in autoencoder operation. Meanwhile, locality-preserving constraint ensures that the adjacent vertices are in close proximity (Step 2-7 in Algorithm 1). The latent representation of encoder and the output of generator will be fed into discriminator to get adversarial loss (Step 10-17). Additionally, the generator transforms Gaussian noise into the latent space as closely as the true data, by passing through multilayer perceptron (Step 20-23). After the training of NetRA, we obtain the vertex representations $f_\phi(\mathbf{x})$ of the network by passing the input walks through the encoder function.

Optimality Analysis. NetRA, as illustrated in Figure 1, can be interpreted as minimizing the divergence between two distributions, namely $\mathbb{P}_\phi(\mathbf{x})$ and $\mathbb{P}_\theta(\mathbf{z})$. We provide the following proposition

Algorithm 1 NetRA Model Training

Require: the walks generated from input graph, maximum training epoch n_{epoch}, the number of discriminator training per generator iteration n_D.

for $epoch = 0; epoch < n_{epoch}$ **do**
  Minimizing $\mathcal{L}_{\text{LE}}(\phi; \mathbf{x})$ **with autoencoder** $\mathcal{L}_{\text{AE}}(\phi, \psi; \mathbf{x})$
  Sample $\{\mathbf{x}^{(i)}\}_{i=1}^{B} \sim \mathbb{P}_{data}(\mathbf{x})$ a batch from the walks
  Compute latent representation $f_\phi(\mathbf{x}^{(i)})$
  Compute reconstruction output $h_\psi(f_\phi(\mathbf{x}^{(i)}))$
  Compute $\mathcal{L}_{\text{AE}}(\phi, \psi)$ and $\mathcal{L}_{\text{LE}}(\phi)$ using Eq.(8) and Eq.(6)
  Backpropagate loss and update ϕ and ψ using Eq.(16)-(17)

  Discriminator training
  for $n = 0$, $n < n_D$ **do**
    Sample $\{\mathbf{x}^{(i)}\}_{i=1}^{B} \sim \mathbb{P}_{data}(\mathbf{x})$ a batch from the walks
    Sample $\{\mathbf{z}^{(i)}\}_{i=1}^{B} \sim \mathbb{P}_{g}(\mathbf{z})$ a batch from the noise
    Compute representations $f_\phi(\mathbf{x}^{(i)})$ and $g_\theta(z^{(i)})$
    Compute $\mathcal{L}_{\text{DIS}}(w)$ using Eq.(12)
    Backpropagate loss and update w using Eq.(15)
    clip the weight w within $[-c, c]$
  end for

  Generator training
  Sample $\{\mathbf{z}^{(i)}\}_{i=1}^{B} \sim \mathbb{P}_{g}(\mathbf{z})$ a batch from the noise
  Compute the representation $g_\theta(\mathbf{z}^{(i)})$
  Compute $\mathcal{L}_{\text{GEN}}(\theta)$ using Eq.(11)
  Backpropagate loss and update θ using Eq.(14)
end for

which shows that under our parameter settings, if the Wasserstein distance converges, the encoder distribution $f_\phi(\mathbf{x}) \sim \mathbb{P}_\phi(\mathbf{x})$ converges to the generator distribution $g_\theta(\mathbf{z}) \sim \mathbb{P}_\theta(\mathbf{z})$.

Proposition 3.2. *Let $\mathbb{P}$ be a distribution on a compact set $\mathcal{X}$, and $(\mathbb{P}_n)_{n\in\mathbb{N}}$ be a sequence of distributions on $\mathcal{X}$. Considering $W(\mathbb{P}_n, \mathbb{P}) \to 0$ as $n \to \infty$, the following statements are equivalent:*

(1) $\mathbb{P}_n \overset{\mathcal{D}}{\rightsquigarrow} \mathbb{P}$ *where* $\overset{\mathcal{D}}{\rightsquigarrow}$ *represents convergence in distribution for random variables.*
(2) $\mathbb{E}_{\mathbf{x}\sim\mathbb{P}_n}[F(\mathbf{x})] \to \mathbb{E}_{\mathbf{x}\sim\mathbb{P}}[F(\mathbf{x})]$, *where* $F(\mathbf{x}) = \Pi_{i=1}^{n}\mathbf{x}_i^{p_i}, \mathbf{x} \in \mathbb{R}^n, \sum_{i=1}^{n} p_i = k, k > 1, k \in \mathbb{N}$.

Proof. (1) As shown in [36], $\mathbb{P}_n$ converges to $\mathbb{P}$ is equivalent to $W(\mathbb{P}_n, \mathbb{P}) \to 0$.

(2) According to the *Portmanteau Theorem* [36], $\mathbb{E}_{\mathbf{x}\sim\mathbb{P}_n}[F(\mathbf{x})] \to \mathbb{E}_{\mathbf{x}\sim\mathbb{P}}[F(\mathbf{x})]$ holds if $F : \mathbb{R}^n \to \mathbb{R}$ is a bounded continuous function. Our encoder $f_\phi(\cdot)$ is bounded as the inputs are normalized to lie on the unit sphere, and our generator $g_\theta(\cdot)$ is also bounded to lie in $(-1, 1)^n$ by *tanh* function. Therefore, $F(\mathbf{x}) = \Pi_{i=1}^{n}\mathbf{x}_i^{p_i}$ is a bounded continuous function for all $p_i > 0$, and

$$\mathbb{E}_{\mathbf{x}\sim\mathbb{P}_n}[\Pi_{i=1}^{n}\mathbf{x}_i^{p_i}] \to \mathbb{E}_{\mathbf{x}\sim\mathbb{P}}[\Pi_{i=1}^{n}\mathbf{x}_i^{p_i}] \tag{22}$$

such that $\sum_{i=1}^{n} p_i = k, \forall k > 1, k \in \mathbb{N}$. □

Computational Analysis. Given a network $G(V, E)$, where $|V| = n, |E| = m$, according to the definition in Eq.(6), the overall complexity of Laplacian Eigenmaps embedding is $O(n^2)$. In our implementation, we only consider the vertex pairs $(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$ that have edges between them, thus the size of the sampled pairs is $O(m)$, which is much smaller than $O(n^2)$ because real networks are sparse in real settings.

The computational complexity of learning LSTM autoencoders is proportional to the number of parameters $|\phi|$ and $|\psi|$ in each iteration. Therefore, the learning computational complexity for LSTM autoencoders is $O(n_{epoch} \times (|\phi| + |\psi|))$. Similarly, for the generator and discriminator, each invocation of backpropagation is typically linear in the number of parameters $O(|\theta|)$ and $O(|w|)$. Thus the computational complexity for generator and discriminator is $O(n_{epoch} \times (n_D \times |w| + |\theta|))$. It is basically quadratic if the input and hidden layers are of roughly the same size. However, if we set the size of embedding layers much less than that of the inputs, the time complexity reduces to $O(n)$.

Table 1: Statistics of the real-world network datasets

Dataset	$\lvert V\rvert$	$\lvert E\rvert$	Avg. degree	#label	Type
UCI	1,899	27,676	14.57	-	Directed
JDK	6,434	53,892	8.38	-	Directed
BLOG	10,312	333,983	32.96	-	Undirected
DBLP	180,768	382,732	4.23	-	Undirected
PPI	3,890	76,584	19.69	50	Directed
WIKI	4,777	184,812	38.69	40	Directed

4 EVALUATION

We evaluate the performance of our model with extensive experiments on tasks including network reconstruction, link prediction and multi-label classification, using a variety of network datasets.

4.1 Datasets

To verify the performance of the proposed network embedding model, we conduct experiments on a variety of networks from different domains including the social network, software dependency network, biological network and language network, as summarized in Table 1.

- UCI message (UCI) [22] is a directed communication network containing sent messages (edges) between the users (vertices) of an online community of students from the University of California Irvine.
- JDK dependency (JDK)[2] is the software class dependency network of the JDK 1.6.0.7 framework. The network is directed, with vertices representing Java classes and an edge between two vertices indicating there exists a dependency between the two classes.
- Blogcatalog (BLOG) [29] is an undirected social network from BlogCatalog website which manages the bloggers and their blogs. The vertices represent users and edges represent friendship between users.
- DBLP[3] is an undirected collaboration graph of authors from the DBLP computer science bibliography. The vertices in this network represent the authors, and the edges represent the co-authorships between two authors.
- Wikipedia (WIKI) [12] is a directed word network. Vertex labels represent the Part-of-Speech (POS) tags inferred using the Stanford POS-Tagger [33].
- Protein-Protein Interactions (PPI) [3] is a subgraph of the PPI network for Homo Sapiens, which is a network depicting interactions between human proteins. The vertex label indicates biological states of proteins.

[2] http://konect.uni-koblenz.de/networks/subelj_jdk
[3] http://dblp.uni-trier.de/xml

4.2 Comparing Algorithms

To evaluate the performance of our network embedding model, the competitors used in this paper are summarized as follows.

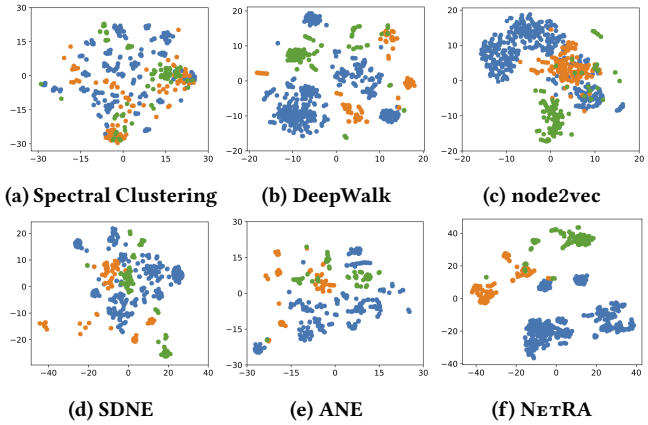
- Spectral Clustering (SC) [30]: SC is an approach based on matrix factorization, generating the vertex representation with the smallest d eigenvectors of the normalized Laplacian matrix of the graph.
- DeepWalk [23]: DeepWalk is a skip-gram [20] based model which learns the graph embedding with truncated random walks.
- node2vec [12]: This approach combines the advantage of breadth-first traversal and depth-first traversal algorithms. The random walks generated by node2vec can better represent the structural equivalence.
- Structural Deep Network Embedding (SDNE) [37]: SDNE is a deep learning based network embedding model which uses autoencoder and locality-preserving constraint to learn vertex representations that capture the highly non-linear network structure.
- Adversarial Network Embedding (ANE) [6]:ANE proposes to train a discriminator to push the embedding distribution to match the fixed prior.

For fair comparison [18], we run each algorithm to generate 300 dimensional vertex representations on different datasets, unless noted otherwise. The number of walks per vertex in DeepWalk and node2vec is set to 10 with walk length 30, which is the same as the random walk generation step of NetRA. The window size of DeepWalk and node2vec is optimized to 10. node2vec is optimized with grid search over its return and in-out parameters $(p, q) \in \{0.25, 0.50, 1, 2, 4\}$. For SDNE, we utilize the default parameter setting as described in [37]. For NetRA, the gradient clipping is performed in every training iteration to avoid the gradient explosion, and we use stochastic gradient descent as the optimizer of autoencoder networks. The multilayer perceptron (MLP) is used in the generator and discriminator. The evaluation of different algorithms is based on applying the embeddings they learned to the downstream tasks, such as link prediction, network reconstruction, and multi-label classification as will be illustrated in the subsequent sections.

4.3 Visualization

In order to demonstrate how well key properties of network structure are captured by the network embedding models, we visualize the embeddings of each compared method. We run different embedding algorithms described in Section 4.2 to obtain low dimensional representations of each vertex and map vertex vectors onto a two dimensional space using t-SNE [35]. With vertex colored by its label, we perform the visualization task on JDK dependency network, as shown in Figure 3.

As observed in Figure 3, three classes are presented: red points for *org.omg*, green points for *org.w3c* and blue points for *java.beans*. It can be seen that the eigenvector-based method Spectral Clustering cannot effectively identify different classes. Other baselines can detect the classes to varying extents. NetRA performs best as it can separate these three classes with large boundaries, except for a small overlap between green and red vertices.



(a) Spectral Clustering (b) DeepWalk (c) node2vec

(d) SDNE (e) ANE (f) NetRA

Figure 3: Visualization results of the compared methods on JDK dependency network: the red points belong to class *org.omg*; the green points belong to class *org.w3c*; the blue points belong to class *java.beans*.

Table 2: AUC score of link prediction

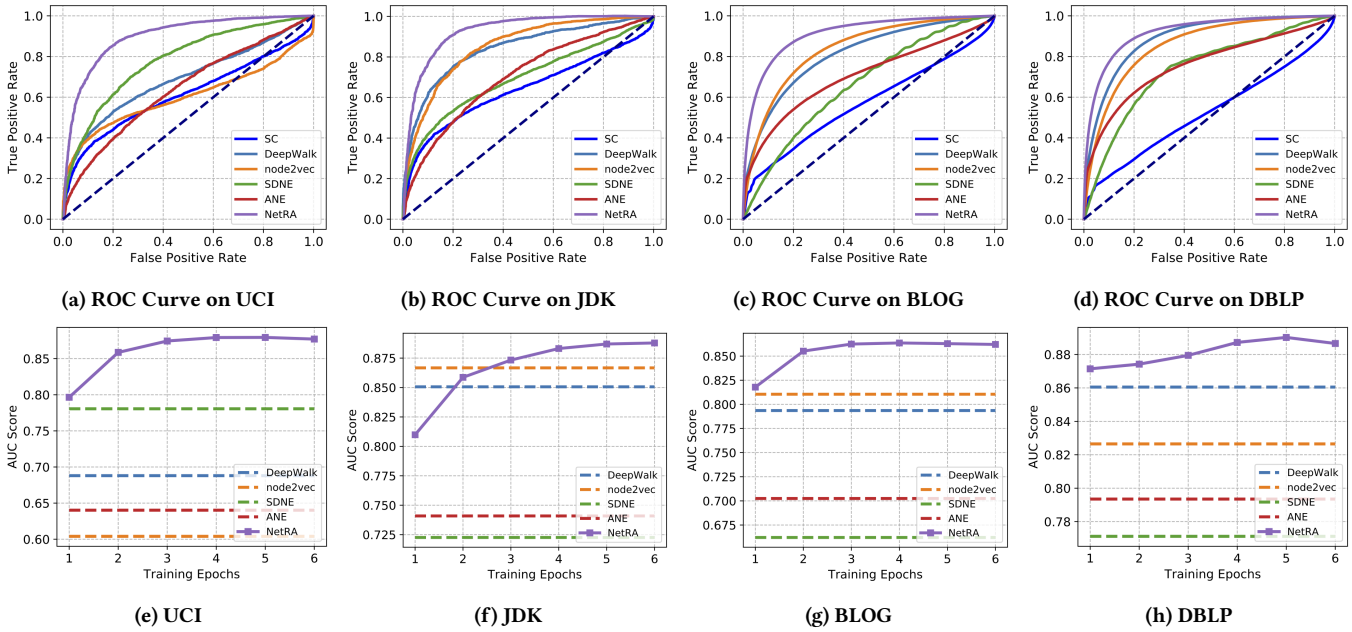
Method	UCI	JDK	BLOG	DBLP
SC	0.6128	0.6686	0.6014	0.5740
DeepWalk	0.6880	0.8506	0.7936	0.8605
node2vec	0.6040	0.8667	0.8105	0.8265
SDNE	0.7806	0.7226	0.6621	0.7712
ANE	0.6402	0.7409	0.7025	0.7935
NetRA	**0.8879**	**0.8913**	**0.8627**	**0.8902**

4.4 Link Prediction

The objective of link prediction task is to infer missing edges given a network with a certain fraction of edges removed. We randomly remove 50% of edges from the network, which serve as positive samples, and select an equal number of vertex pairs without linkage between them as negative samples. With vertex representation learned by network embedding algorithms, we obtain the edge feature from the ℓ_2 norm of two vertex vectors, and use it directly to predict missing edges. Because our focus is network embedding model, this simple experimental setup can evaluate the performance based on the assumption that the representations of two connected vertices should be closer in the Euclidean space. We use the area under curve (AUC) score for evaluation on link prediction task. The results are shown in Table 2.

Obviously, we observe that NetRA outperforms the baseline algorithms across all datasets by a large margin. It can be seen that NetRA achieves 3% to 32% improvement based on the AUC score on the four datasets. By comparing NetRA, node2vec and DeepWalk, which all use random walks as inputs, we can see the effectiveness of generative adversarial regularization for improving the generalization performance in NetRA model. With same random walk sequences, NetRA can overcome the sparsity issue from the sampled sequences of vertices.

We also plot the ROC curve of these four datasets, as shown in Figure 4(a)-(d). The ROC curve of NetRA dominates other approaches and is very close to the (0, 1) point. We train the NetRA



(a) ROC Curve on UCI (b) ROC Curve on JDK (c) ROC Curve on BLOG (d) ROC Curve on DBLP

(e) UCI (f) JDK (g) BLOG (h) DBLP

Figure 4: Link prediction using vertex representation. Evaluated with AUC ROC score versus training epochs.

model with different epochs for different datasets and embed the vertices to get representations after each training epoch. The results are shown in Figure 4(e)-(h). Generally, we can observe that NetRA converges pretty fast with high AUC score almost after the first epoch. When comparing with Deepwalk, node2vec, SDNE and ANE, we can clearly see the better performance of NetRA on these datasets.

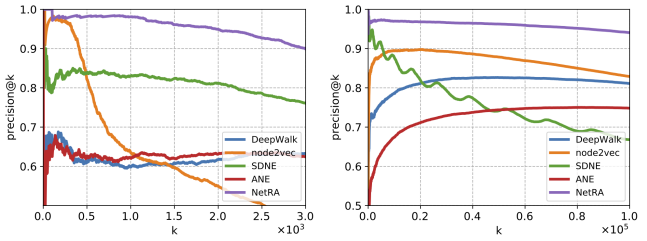
4.5 Network Reconstruction

Network embeddings are considered as effective representations of the original network. The vertex representations learned by networking embedding maintain the edge information for network reconstruction. We randomly select vertex pairs as edge candidates and calculate the Euclidean distance between the vertices. We use the *precision@k*, the fraction of correct predictions in the top k predictions, for evaluation.

$$precision@k = \frac{1}{k} \times |E_{pred}(1:k) \cap E_{obs}|, \tag{23}$$

where $E_{pred}(1:k)$ represents the top k predictions and E_{obs} represents observed edges in original network. In the evaluation, the UCI message and Blogcatalog datasets have been utilized to illustrate the performance of NetRA, with results shown in Figure 5.

As it can be seen from the *precision@k* curves, the NetRA model achieves higher precision in the network reconstruction task. The total number of edge candidates selected in this task is $8k$ for UCI message and $300k$ for Blogcatalog. The reconstruction given by NetRA is very accurate in predicting most positive samples (results on JDK and DBLP datasets show similar trends which haven't been included here). DeepWalk and node2vec can give reasonable reconstruction but the results are worse than NetRA for most k's. By learning smoothly regularized vertex representations using generative adversarial training process [11], our model well integrates the



(a) UCI (b) BLOG

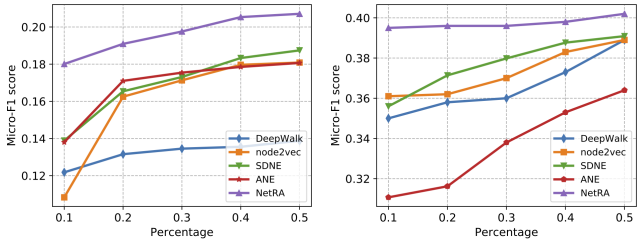
Figure 5: Network reconstruction results on UCI message and Blogcatalog, evaluated by *precision@k*.

locality-preserving and global reconstruction constraints to learn embeddings that capture the "semantic" information.

4.6 Multi-label Classification

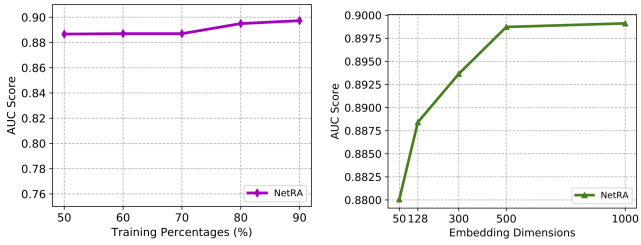
The task of predicting vertex labels with representations learned by network embedding algorithms is widely used in recent studies for performance evaluation [12, 23, 37]. An effective network embedding algorithm should capture network topology and extract most useful features for downstream machine learning tasks. In this section, we use vertex features as input to a one-vs-rest logistic regression using the LIBLINEAR [9] package to train the classifiers. For the Wikipedia and PPI datasets, we randomly sample 10% to 50% of the vertex labels as the training set and use the remaining vertices as the test set. We report *Micro-F1* [37]as evaluation metrics. Each result is averaged by five runs, as shown in Figure 6.

It is evident from the figure that NetRA outperforms the state-of-the-art embedding algorithms on multi-label classification task. In the PPI dataset, NetRA achieves higher *Micro-F1* scores than the baseline models by over 10% in all experiment settings. In the



(a) PPI (b) WIKI

Figure 6: Multi-label classification on PPI and Wikipedia



(a) Edge percentage for training (b) Embedding dimensions

Figure 7: Parameter sensitivity analysis

Wikipedia dataset, NetRA model performs better even with lower percentage training set. This well illustrates the good generalization performance when the training set is sparse. The multi-label classification task shows that, with adversarially regularized LSTM autoencoders, the neighborhood information can be well captured by the low dimensional representations.

4.7 Parameter Sensitivity

In this section, we investigate the parameter sensitivity in NetRA for link prediction. We study how the training set size, embedding dimension and locality-preserving constraint parameter λ_1 will affect the performance of link prediction. Also by changing the architecture of the NetRA model, we can investigate roles of different components in NetRA. Note that similar observations can be made on multi-label classification and network reconstruction tasks.

In Figure 7(a), we vary the training percentage of edges in the UCI message network. As it can be seen, the performance increases as the training ratio increases. Comparing with other algorithms, NetRA can capture the network topology even with a small proportion of edges for training, which demonstrates the generalization capability of the NetRA model. In Figure 7(b), we vary the embedding dimension from 50 to 1000. The prediction performance gets saturated as the dimension increases. Considering that the embedding dimension is related to the parameter volume in NetRA, there exists a trade off between the performance and the efficiency during model training.

The parameter λ_1 is defined by the relative strength between locality-preserving constraint and autoencoder constraint. The higher the λ_1, the larger the gradient comes from the locality-preserving constraint. As observed from the Figure 8, a higher λ_1 enhances the link prediction performance on the UCI message network, indicating the important role of local proximity.

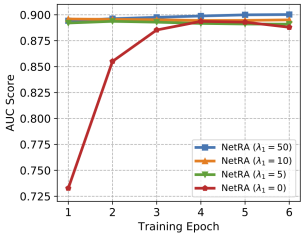


Figure 8: Performance on different λ_1 for $\mathcal{L}_{\text{LE}}$

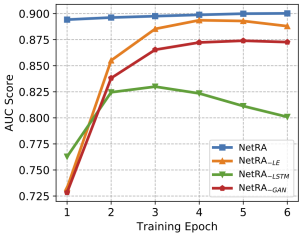


Figure 9: Performance on different NetRA architectures

We also include three variants of NetRA to demonstrate the importance of individual components in NetRA, including NetRA$_{-LE}$, NetRA$_{-LSTM}$, and NetRA$_{-GAN}$. NetRA$_{-LE}$ and NetRA$_{-GAN}$ remove the locality-preserving constraint $\mathcal{L}_{\text{LE}}$ and adversarial regularization $W(\mathbb{P}_\phi(\mathbf{x}), \mathbb{P}_\theta(\mathbf{z}))$, respectively. As for NetRA$_{-LSTM}$, we replace LSTM with multilayer perceptron. It's evident from Figure 9 that LSTM autoencoder, locality-preserving constraint, and adversarial regularization play important roles in NetRA model. The overfitting becomes obvious in the training of NetRA$_{-LSTM}$ and NetRA$_{-GAN}$.

5 RELATED WORK

Recently, we have witnessed the emergence of random walk based methods [8, 12, 23], inspired by the success of natural language processing [23]. These models build connections between network structure and natural language. The training input of these algorithms changes from matrices to sentence-like vertex sequences generated by random walks among connected vertices. The skip-gram algorithm [20] maximizes the co-occurrence probability among the vertices within a certain window in a random walk. DeepWalk [23] obtains effective embeddings using truncated random walks. Node2vec [12] extends the model with flexibility between homophily and structural equivalence [42]. These last two methods motivate the study of network embedding taking advantage of language models.

Deep learning embedding models [4, 32, 37] have also been applied to solve the network embedding problem. Autoencoder based approaches [4, 37] were proposed, utilizing its ability of learning highly non-linear properties. By carefully constructing the learning objective, [37] preserves the first and second proximity of networks which delivers the state-of-the-art performance. Recent works on graph convolutional networks [7, 17] have demonstrated effective convolution operation on network data. Inductive and unsupervised GraphSAGE [14] leverages vertex features [15] and aggregates features among vertex neighborhood.

The rapid advances in deep learning research in last decades have provided novel methods for studying highly non-linear data. One such model is the Generative adversarial networks (GANs) [11] which has achieved great success in generating and learning the latent presentation of high dimensional data, such images [24]. There have been several successful attempts [13, 16, 25] of implementing GANs on discrete structures, such as text and discrete images, which inspired us to investigate network representation learning using GANs. Using GANs to learn the representation of discrete contents like natural languages and social networks remains a challenging

problem due to the difficulty in back-propagation through discrete random variables. Recent work on GANs such as GraphGAN [38] and ANE [6] for discrete data is either though the use of discrete structures [5, 40] or the improved autoencoders [16].

6 CONCLUSION

In this study we proposed NetRA, a deep network embedding model for encoding each vertex in a network as a low-dimensional vector representation with adversarially regularized autoencoders. Our model demonstrated the ability of generative adversarial training process in extracting informative representations. The proposed model has better generalization capability, without requiring an explicit prior density distribution for the latent representations. Specifically, we leveraged LSTM autoencoders that take the sampled sequences of vertices as input to learn smooth vertex representations regularized by locality-preserving constraint and generative adversarial training process. The resultant representations are robust to the sparse vertex sequences sampled from the network. Empirically, we evaluated the learned representations with a variety of network datasets on different tasks such as network reconstruction, link prediction and multi-label classification. The results showed substantial improvement over the state-of-the-art network embedding competitors.

ACKNOWLEDGEMENT

The work is partially supported by NIH U01HG008488, NIH R01GM115833, NIH U54GM114833, and NSF IIS-1313606. Research of the fourth author was sponsored by the Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF-09-2-0053. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on. We thank the anonymous reviewers for their careful reading and insightful comments on our manuscript.

REFERENCES

[1] Martin Arjovsky, Soumith Chintala, and Léon Bottou. 2017. Wasserstein generative adversarial networks. In *ICML*. 214–223.
[2] Peter Borg and Kurt Fenech. 2017. Reducing the maximum degree of a graph by deleting vertices. *Australasian Journal Of Combinatorics* 69, 1 (2017), 29–40.
[3] Bobby-Joe Breitkreutz, Chris Stark, Teresa Reguly, et al. 2007. The BioGRID interaction database: 2008 update. *Nucleic acids research* 36, suppl_1 (2007), D637–D640.
[4] Shaosheng Cao, Wei Lu, and Qiongkai Xu. 2016. Deep Neural Networks for Learning Graph Representations.. In *AAAI*. 1145–1152.
[5] Tong Che, Yanran Li, Ruixiang Zhang, R Devon Hjelm, Wenjie Li, Yangqiu Song, and Yoshua Bengio. 2017. Maximum-likelihood augmented discrete generative adversarial networks. *arXiv preprint arXiv:1702.07983* (2017).
[6] Quanyu Dai, Qiang Li, Jian Tang, and Dan Wang. 2017. Adversarial Network Embedding. *arXiv preprint arXiv:1711.07838* (2017).
[7] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *NIPS*. 3844–3852.
[8] Yuxiao Dong, Nitesh V Chawla, and Ananthram Swami. 2017. metapath2vec: Scalable representation learning for heterogeneous networks. In *KDD*. ACM, 135–144.
[9] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. 2008. LIBLINEAR: A library for large linear classification. *JMLR* 9, Aug (2008), 1871–1874.
[10] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. 2016. *Deep learning*. Vol. 1. MIT press Cambridge.
[11] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In *NIPS*. 2672–2680.
[12] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *KDD*. ACM, 855–864.
[13] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. 2017. Improved training of wasserstein gans. *arXiv preprint arXiv:1704.00028* (2017).
[14] William L Hamilton, Rex Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. *arXiv preprint arXiv:1706.02216* (2017).
[15] Xiao Huang, Jundong Li, and Xia Hu. 2017. Label informed attributed network embedding. In *WSDM*. ACM, 731–739.
[16] Yoon Kim, Kelly Zhang, Alexander M Rush, Yann LeCun, et al. 2017. Adversarially Regularized Autoencoders for Generating Discrete Structures. *arXiv preprint arXiv:1706.04223* (2017).
[17] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).
[18] Omer Levy, Yoav Goldberg, and Ido Dagan. 2015. Improving distributional similarity with lessons learned from word embeddings. *Transactions of the Association for Computational Linguistics* 3 (2015), 211–225.
[19] Alireza Makhzani, Jonathon Shlens, Navdeep Jaitly, and Ian Goodfellow. 2016. Adversarial Autoencoders. In *ICLR*.
[20] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *NIPS*. 3111–3119.
[21] Paul Milgrom and Ilya Segal. 2002. Envelope theorems for arbitrary choice sets. *Econometrica* 70, 2 (2002), 583–601.
[22] Tore Opsahl and Pietro Panzarasa. 2009. Clustering in weighted networks. *Social networks* (2009), 155–163.
[23] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *KDD*. ACM, 701–710.
[24] Alec Radford, Luke Metz, and Soumith Chintala. 2015. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434* (2015).
[25] Sai Rajeswar, Sandeep Subramanian, Francis Dutil, Christopher Pal, and Aaron Courville. 2017. Adversarial Generation of Natural Language. *arXiv preprint arXiv:1705.10929* (2017).
[26] Leonardo F.R. Ribeiro, Pedro H.P. Saverese, and Daniel R. Figueiredo. 2017. Struc2Vec: Learning Node Representations from Structural Identity. In *KDD*. ACM, 385–394.
[27] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *NIPS*. 3104–3112.
[28] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. Line: Large-scale information network embedding. WWW, 1067–1077.
[29] Lei Tang and Huan Liu. 2009. Relational learning via latent social dimensions. In *KDD*. ACM, 817–826.
[30] Lei Tang and Huan Liu. 2011. Leveraging social media networks for classification. *Data Mining and Knowledge Discovery* 23, 3 (2011), 447–478.
[31] Athanasios Theocharidis, Stjin Van Dongen, Anton J Enright, and Tom C Freeman. 2009. Network visualization and analysis of gene expression data using BioLayout Express3D. *Nature protocols* 4, 10 (2009), 1535–1550.
[32] Fei Tian, Bin Gao, Qing Cui, Enhong Chen, and Tie-Yan Liu. 2014. Learning Deep Representations for Graph Clustering.. In *AAAI*. 1293–1299.
[33] Kristina Toutanova, Dan Klein, Christopher D Manning, and Yoram Singer. 2003. Feature-rich part-of-speech tagging with a cyclic dependency network. Association for Computational Linguistics, 173–180.
[34] Tomasz Tylenda, Ralitsa Angelova, and Srikanta Bedathur. 2009. Towards time-aware link prediction in evolving social networks. In *Proceedings of the 3rd workshop on social network mining and analysis*. ACM, 9.
[35] Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing Data using t-SNE. *JMLR* 9 (2008), 2579–2605.
[36] Cédric Villani. 2008. *Optimal transport: old and new*. Vol. 338. Springer Science & Business Media.
[37] Daixin Wang, Peng Cui, and Wenwu Zhu. 2016. Structural deep network embedding. In *KDD*. ACM, 1225–1234.
[38] Hongwei Wang, Jia Wang, Jialin Wang, Miao Zhao, Weinan Zhang, Fuzheng Zhang, Xing Xie, and Minyi Guo. 2018. GraphGAN: Graph Representation Learning with Generative Adversarial Nets. *AAAI* (2018).
[39] J. Weston, F. Ratle, and R. Collobert. 2008. Deep learning via semi-supervised embedding. In *ICML*.
[40] Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. 2017. SeqGAN: Sequence Generative Adversarial Nets with Policy Gradient. In *AAAI*. 2852–2858.
[41] Wenchao Yu, Guangxiang Zeng, Ping Luo, Fuzhen Zhuang, Qing He, and Zhongzhi Shi. 2013. Embedding with autoencoder regularization. In *ECMLPKDD*. Springer, 208–223.
[42] Daokun Zhang, Jie Yin, Xingquan Zhu, and Chengqi Zhang. 2016. Homophily, Structure, and Content Augmented Network Representation Learning. In *ICDM*. IEEE, 609–618.