Self-Attentive Attributed Network Embedding Through Adversarial Learning

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

***Abstract*—Network embedding aims to learn the low-dimensional representations/embeddings of vertices which preserve the structure and inherent properties of the networks. The resultant embeddings are beneficial to downstream tasks such as vertex classification and link prediction. A vast majority of real-world networks are coupled with a rich set of vertex attributes, which could be potentially complementary in learning better embeddings. Existing attributed network embedding models, with shallow or deep architectures, typically seek to match the representations in topology space and attribute space for each individual vertex by assuming that the samples from the two spaces are drawn uniformly. The assumption, however, can hardly be guaranteed in practice. Due to the intrinsic sparsity of sampled vertex sequences and incompleteness in vertex attributes, the discrepancy between the attribute space and the network topology space inevitably exists. Furthermore, the interactions among vertex attributes, a.k.a cross features, have been largely ignored by existing approaches. To address the above issues, in this paper, we propose NETTENTION, a self-attentive network embedding approach that can efficiently learn vertex embeddings on attributed network. Instead of sample-wise optimization, NETTENTION aggregates the two types of information through minimizing the difference between the representation distributions in the low-dimensional topology and attribute spaces. The joint inference is encapsulated in a generative adversarial training process, yielding better generalization performance and robustness. The learned distributions consider both locality-preserving and global reconstruction constraints which can be inferred from the learning of the adversarially regularized autoencoders. Additionally, a multi-head self-attention module is developed to explicitly model the attribute interactions. Extensive experiments on benchmark datasets have verified the effectiveness of the proposed NETTENTION model on a variety of tasks, including vertex classification and link prediction.**

***Index Terms*—network embedding, attributed network, deep embedding, generative adversarial networks, self-attention**

I. INTRODUCTION

Low-dimensional network representations of vertices have proved to be extremely useful as a feature extraction mechanism for a wide variety of network prediction and analysis tasks such as vertex classification and link prediction. The basic idea of network embedding is to use dimension reduction techniques to learn compact vertex embeddings while preserving the neighborhood information. Prior methods, such as DeepWalk [1], node2vec [2] and SDNE [3], have focused primarily on preserving network topology structure while the attributes associated with each vertex have largely been ignored. In real-world scenarios, however, apart from network topology information, vertices are often associated with a rich set of attributes, e.g. user profiles in social networks, article's topic information in citation network. These attributes not only provide the proximity information in the attribute space, but also guide the formation of networks [4], [5]. Integrating the informative attributes with network topology can benefit network analysis tasks.

Recent work on attributed network embedding shows that jointly learning network representations with network topology information and vertex attributes enhances the performance on various tasks. Existing methods include matrix factorization based shallow models that enforce the homophily property between attributes and network topology in the representation space [5], [6], as well as neural network based deep models that explore the highly non-linear property in the embedding space and aggregate attributes/features from vertices' local neighborhoods [7]–[10].

Despite the recent progresses of attributed network embedding, existing approaches face two major open challenges. 1) Previous methods [6]–[10] seek to match the representations in topology space and attribute space for each individual vertex by assuming that the samples from the two spaces are drawn uniformly. The assumption, however, can hardly be guaranteed. On one hand, to capture the network topology information, most network embedding approaches employ network sampling techniques to derive vertex sequences as training datasets. However, the sampling strategy suffers from the data sparsity problem since the total number of vertex sequences is usually very large in real networks and it is often intractable to enumerate all [1]–[3]. On the other hand, in real-world problems, vertex attribute data are sparse, noisy and often incomplete due to measurement errors and ad hoc data collection techniques [3], [9]. Consequently, discrepancy between topology space and attribute space inevitably exists for individual vertex. As such, it is not reasonable to force the learned low-dimensional representations of each vertex in the two spaces to be in complete consensus. 2) An effective embedding approach on attributed network relies on high-order combinatorial attributes, which capture the interactions among vertex attributes. Modeling different orders of attribute interactions explicitly with good interpretability is nontrivial and has yet to be explored by existing embedding models.

To address the aforementioned challenges, we propose

*Corresponding authors.

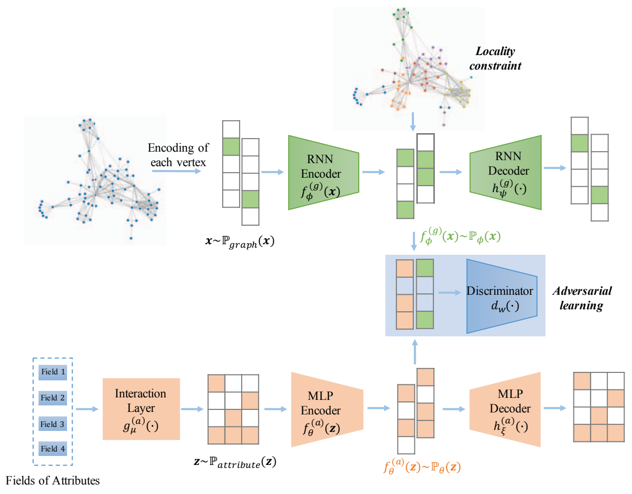

Locality constraint
Encoding of each vertex
RNN Encoder $f_\phi^{(g)}(\boldsymbol{x})$
RNN Decoder $h_\psi^{(g)}(\cdot)$
$\boldsymbol{x} \sim \mathbb{P}_{graph}(\boldsymbol{x})$
$f_\phi^{(g)}(\boldsymbol{x}) \sim \mathbb{P}_\phi(\boldsymbol{x})$
Discriminator $d_w(\cdot)$
Adversarial learning
Field 1
Field 2
Field 3
Field 4
Fields of Attributes
Interaction Layer $g_\mu^{(a)}(\cdot)$
$\boldsymbol{z} \sim \mathbb{P}_{attribute}(\boldsymbol{z})$
MLP Encoder $f_\theta^{(a)}(\boldsymbol{z})$
$f_\theta^{(a)}(\boldsymbol{z}) \sim \mathbb{P}_\theta(\boldsymbol{z})$
MLP Decoder $h_\xi^{(a)}(\cdot)$

Fig. 1: Illustration of NETTENTION, the deep network embedding architecture through adversarial learning. The encoded walks extracted from the network (denoted as $\boldsymbol{x}$) and attributes (denoted as $\boldsymbol{z}$) pass through the corresponding encoders to learn the distributions over topology space and attribute space. The joint optimization of these two distributions is encapsulated in a generative adversarial training process. The interactions of attributes among different fields are modeled using a self-attention mechanism.

NETTENTION, an adversarially regularized self-attentive embedding learning approach that can efficiently learn vertex embeddings on attributed networks via adversarial training. These vertex embeddings well capture the network structure by jointly considering both locality-preserving and global reconstruction constraints in the network topology space and attribute space. Specifically, two deep autoencoding architectures are proposed to capture the underlying high nonlinearity in both network structure and attributes, as shown in Figure 1. NETTENTION employs a discrete recurrent neural network autoencoder (RNN autoencoder) to learn a continuous vertex representation distribution in the topology space with sampled sequences of vertices as inputs. In parallel, a multilayer perceptron autoencoder (MLP autoencoder) is adopted to learn the distribution in the attribute space. The interactions of attributes among different fields are modeled using a self-attention mechanism (denoted as interaction layer in Figure 1). Additionally, NETTENTION enforces the learned vertex representations to preserve the neighborhood proximity with the locality constraint derived from the network structure. In order to learn a coherent and complementary representation from the topological structure and vertex attributes of a network, we propose a novel strategy to minimize the difference between the representation distributions in the low-dimensional topology and attribute spaces, which encapsulates a join inference in a generative adversarial training process. Extensive experiments on benchmark datasets have verified the effectiveness of our proposed approach on a variety of tasks, including vertex classification, link prediction and network visualization. To summarize, the main contributions of this work are as follows:

- A novel deep self-attentive attributed network embedding model, NETTENTION, has been proposed to learn vertex representations through minimizing a penalized form of the Wasserstein distance between the distributions from topology space and attribute space.
- A self-attention layer has been designed to learn the interactions of attributes among different fields, a.k.a. cross features, which enhances network analysis performance.
- Due to the intrinsic sparsity and incompleteness in topological structure and network attributes, NETTENTION proposes to use a generative adversarial network based regularizer to pull the embedding distributions in the two spaces together, so as to learn a consensus embedding. Thus, NETTENTION obtains better generalization performance and robustness.
- Extensive experiments have been conducted on tasks of link prediction, vertex classification and visualization using real-world information networks. Experimental results demonstrate the effectiveness and efficiency of NETTENTION.

The rest of this paper is organized as follows. In Section II, we review the preliminary knowledge including general network embedding models, optimal transport theory and attention mechanism. In Section III, we describe the NETTENTION model to learn low dimensional representations though adversarial learning. In Section IV, we demonstrate the performance of NETTENTION by adapting this joint learning framework on tasks of vertex classification, link prediction and visualization. In Section V, we compare the NETTENTION framework to other network embedding algorithms and discuss several related works. Finally, in Section VI we conclude this study and mention several directions for future work.

II. PRELIMINARIES

A. Network Embedding

Network embedding approaches seek to learn representations that encode structural information of the network. These approaches learn a mapping that embeds vertices as points in a low-dimensional space. Given the encoded vertex set $\{\boldsymbol{x}^{(1)}, ..., \boldsymbol{x}^{(n)}\}$, finding an embedding $f_\phi(\boldsymbol{x}^{(i)})$ of each $\boldsymbol{x}^{(i)}$ can be formalized as an optimization problem [11],

$$\min_{\phi,\psi} \sum_{1 \le i < j \le n} \mathcal{L}(f_\phi(\boldsymbol{x}^{(i)}), f_\phi(\boldsymbol{x}^{(j)}), \varphi_{ij}) + \lambda \cdot \mathcal{R}(\psi; \boldsymbol{x}) \quad (1)$$

where $f_\phi(\boldsymbol{x}) \in \mathbb{R}^d$ is the embedding result for a given input $\boldsymbol{x}$, $\mathcal{L}(\cdot)$ is the loss function between a pair of inputs, φ_{ij} is the weight between $\boldsymbol{x}^{(i)}$ and $\boldsymbol{x}^{(j)}$, $\mathcal{R}(\cdot)$ serves as a regularizer such as autoencoder [3]. In this paper, we consider the Laplacian Eigenmaps (LE) that enables the embedding to preserve the locality property of network structure. Formally, the embedding can be obtained by minimizing the following objective function

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

B. Generative Adversarial Networks

The generative adversarial networks (GANs) [12] build an adversarial training platform for two players, namely *generator* $g_\theta(\cdot)$ and *discriminator* $d_w(\cdot)$ to play a minimax game. Here w and θ are model parameters.

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

The *generator* $g_\theta(\cdot)$ tries to map the noise to the input space as closely as the true data, while the *discriminator* $d_w(\boldsymbol{x})$ represents the probability that $\boldsymbol{x}$ comes from the data rather than the noise. It aims to distinguish *real* data distribution $\mathbb{P}_{data}(\boldsymbol{x})$ and *fake* sample distribution $\mathbb{P}_g(\boldsymbol{z})$, e.g. $\mathbf{z} \sim \mathcal{N}(0, \mathbf{1})$. The Jensen-Shannon divergence is originally used by GANs which is known to suffer from training unstability. To overcome this problem, GANsWasserstein GANs [13] use the Earth-Mover (Wasserstein-1) distance, and solve the problem

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

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

III. Approach

We consider the attributed network embedding problem as finding a transformation that maps the structural input data to lower-dimensional representations. The resultant representations can be used in the downstream tasks, such as link prediction and vertex classification.

A. Problem Definition

Let $G(\mathcal{V}, \mathcal{E}, \boldsymbol{Z})$ denote an attributed network with n vertices ($n = |\mathcal{V}|$). $\mathcal{E}$ is the edge set and $\boldsymbol{Z} \in \mathbb{R}^{n\times d_0^{(a)}}$ is the attribute matrix where $\boldsymbol{z} = \boldsymbol{Z}_{i,:} \in \mathbb{R}^{d_0^{(a)}}$ denotes the attribute vector of the i^{th} vertex. $\boldsymbol{X} \in \mathbb{R}^{n\times d_0^{(g)}}$ represents the encoded vertex vectors through a lookup table or one-hot encoding. And $\boldsymbol{x} = \boldsymbol{X}_{i,:} \in \mathbb{R}^{d_0^{(g)}}$ denotes the vector representation of the i^{th} vertex. A random walk generator is utilized to obtain truncated random walks (i.e. sequences of vertices) rooted from each vertex $v \in \mathcal{V}$ in $G(\mathcal{V}, \mathcal{E}, \boldsymbol{Z})$. A walk is sampled randomly from the neighbors of the last visited vertex until the preset maximum length is reached.

Definition 1 (Network Homophily). *Given a network $G(\mathcal{V}, \mathcal{E}, \boldsymbol{Z})$, vertices of similar attributes are likely to be close to each other (i.e. connected by edges) than dissimilar ones. That is, the low-dimensional vertex representations of $\{\boldsymbol{X}_{i,:}\}_{i=1}^n$ are drawn from a distribution similar to that of the representations of the attributes $\{\boldsymbol{Z}_{i,:}\}_{i=1}^n$.*

Inspired by the principle of homophily [4], we assume that the formation of network is highly correlated with vertex attributes, and leveraging vertex attribute information can improve network embedding performance. Therefore, in this paper, we learn a low-dimensional vertex embedding based on the network topology $G(\cdot)$ and the attribute matrix $\boldsymbol{Z}$, such that the learned representations can preserve the proximity existing in both the network topology space and the attribute space. The key point is to view both network structure and attribute information as the latent factors to drive the formation of the network. Formally, let $f_\phi^{(g)}(\boldsymbol{x})$ and $f_\theta^{(a)}(\boldsymbol{z})$ denote the learned representations of network topological structure and attribute information, respectively. A mapping $f\{\boldsymbol{X}, \boldsymbol{Z}\} \to \boldsymbol{M}$ is going to be learned by minimizing the disagreement $\mathcal{L}(f_\phi^{(g)}(\boldsymbol{X}), f_\theta^{(a)}(\boldsymbol{Z}))$ between the learned topology space and structure space. Here, $\boldsymbol{M} \in \mathbb{R}^{n\times d}$ is the resultant representation matrix. Each row of $\boldsymbol{M}$ can be viewed as a vertex feature vector.

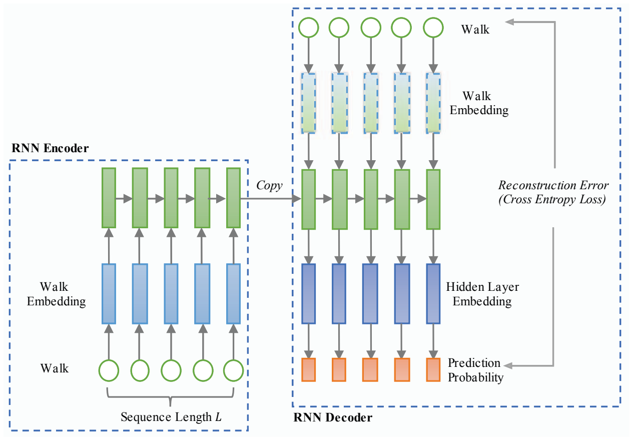


Fig. 2: Illustration of the RNN autoencoder with walks extracted from network as input

B. The Nettention Model

In this section, we propose Nettention, a self-attentive deep attributed network embedding model through adversarial learning. As illustrated in Figure 1, two deep autoencoding architectures are proposed to capture the underlying high non-linearity in both network structure and vertex attribute spaces. Autoencoders are popularly used for dimension reduction, providing informative low dimensional representations of input data. In Nettention, a discrete RNN autoencoder is employed, as depicted in Figure 2, to learn a continuous vertex representation distribution in the topology space with sampled sequences of vertices as inputs. The RNN autoencoder can be trained individually by minimizing the negative log-likelihood of reconstruction, which is indicated by cross entropy loss in the implementation

$$\mathcal{L}_{\text{AE}}^{(g)}(\phi, \psi; \boldsymbol{x}) = -\mathbb{E}_{\boldsymbol{x}\sim\mathbb{P}_{graph}(\boldsymbol{x})}[\text{dist}(\boldsymbol{x}, h_\psi^{(g)}(f_\phi^{(g)}(\boldsymbol{x})))] \tag{5}$$

where $\text{dist}(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{x}\log\boldsymbol{y} + (\mathbf{1} - \boldsymbol{x})\log(\mathbf{1} - \boldsymbol{y})$. Here $\boldsymbol{x}$ is a sampled batch from training data. $f_\phi^{(g)}(\boldsymbol{x})$ is the embedded latent representation of $\boldsymbol{x}$. ϕ and ψ are parameters of the encoder and decoder functions in the network topology space, respectively. Similarly, in the attribute space, an MLP autoencoder is adopted to learn the distribution. Therefore we have,

$$\mathcal{L}_{\text{AE}}^{(a)}(\theta, \xi; \boldsymbol{z}) = -\mathbb{E}_{\boldsymbol{z} \sim \mathbb{P}_{attribute}(\boldsymbol{z})}[\text{dist}(\boldsymbol{z}, h_{\xi}^{(a)}(f_{\theta}^{(a)}(\boldsymbol{z})))] \quad (6)$$

where θ and ξ are parameters of the encoder and decoder functions in the attribute space, respectively.

In the training iteration of the RNN autoencoder, not only the encoder and decoder are updated, the locality-preserving loss is also jointly minimized,

$$\mathcal{L}_{\text{LE}}(\phi; \boldsymbol{x}) = \sum_{1 \leq i < j \leq n} \| f_{\phi}^{(g)}(\boldsymbol{X}_{i,:}) - f_{\phi}^{(g)}(\boldsymbol{X}_{j,:}) \|^2 \varphi_{ij} \quad (7)$$

where $f_{\phi}^{(g)}(\boldsymbol{x}) \in \mathbb{R}^d$ is the embedding result for a given input $\boldsymbol{x}$. φ_{ij} is the weight between vertex i and j.

In order to minimize the discrepancies between attribute distribution and network topology distribution, we propose 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 from data [14]. 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 [15]. To be more specific, in NETTENTION, we introduce a discriminator in the latent space which tries to separate the generated vectors from the encoder network $f_{\phi}^{(g)}(\cdot)$ with network topology and the encoder network $f_{\theta}^{(a)}(\cdot)$ with attributes. Let $f_{\phi}^{(g)}(\boldsymbol{x}) \sim \mathbb{P}_{\phi}(\boldsymbol{x})$ denotes a sample drawn from the distribution of the network topology space $\mathbb{P}_{\phi}(\boldsymbol{x})$, and $f_{\theta}^{(a)}(\boldsymbol{z}) \sim \mathbb{P}_{\theta}(\boldsymbol{z})$ denotes a sample drawn from the distribution of the attribute space $\mathbb{P}_{\theta}(\boldsymbol{z})$. 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 [13],

$$W(\mathbb{P}_{\phi}(\boldsymbol{x}), \mathbb{P}_{\theta}(\boldsymbol{z})) = \sup_{\|d(\cdot)\|_{L \leq 1}} \mathbb{E}_{\boldsymbol{y} \sim \mathbb{P}_{\phi}(\boldsymbol{x})}[d(\boldsymbol{y})] - \mathbb{E}_{\boldsymbol{y} \sim \mathbb{P}_{\theta}(\boldsymbol{z})}[d(\boldsymbol{y})] \quad (8)$$

where $\|d(\cdot)\|_{L \leq 1}$ 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

$$\begin{aligned} W(\mathbb{P}_{\phi}(\boldsymbol{x}), \mathbb{P}_{\theta}(\boldsymbol{z})) \propto \max_{w \in \mathcal{W}} & \mathop{\mathbb{E}}_{\boldsymbol{x} \sim \mathbb{P}_{graph}(\boldsymbol{x})}[d_w(f_{\phi}^{(g)}(\boldsymbol{x}))] \\ & - \mathop{\mathbb{E}}_{\boldsymbol{z} \sim \mathbb{P}_{attribute}(\boldsymbol{z})}[d_w(f_{\theta}^{(a)}(\boldsymbol{z}))] \end{aligned} \quad (9)$$

In our adversarial setup, we use the parameterized encoders $f_{\phi}^{(g)}(\boldsymbol{x})$ and $f_{\theta}^{(a)}(\boldsymbol{z})$ as generators, and the training of generator and discriminator are separated. As for the generators, the cost function is defined by,

$$\begin{aligned} \mathcal{L}_{\text{GEN}}(\theta, \phi; \boldsymbol{x}, \boldsymbol{z}) = & \mathbb{E}_{\boldsymbol{x} \sim \mathbb{P}_{graph}(\boldsymbol{x})}[d_w(f_{\phi}^{(g)}(\boldsymbol{x}))] \\ & - \mathbb{E}_{\boldsymbol{z} \sim \mathbb{P}_{attribute}(\boldsymbol{z})}[d_w(f_{\theta}^{(a)}(\boldsymbol{z}))] \end{aligned} \quad (10)$$

Similarly, and the cost function of discriminator is defined by,

$$\begin{aligned} \mathcal{L}_{\text{DIS}}(w; \boldsymbol{x}, \boldsymbol{z}) = & - \mathbb{E}_{\boldsymbol{x} \sim \mathbb{P}_{graph}(\boldsymbol{x})}[d_w(f_{\phi}^{(g)}(\mathbf{x}))] \\ & + \mathbb{E}_{\boldsymbol{z} \sim \mathbb{P}_{attribute}(\boldsymbol{z})}[d_w(f_{\theta}^{(a)}(\boldsymbol{z}))] \end{aligned} \quad (11)$$

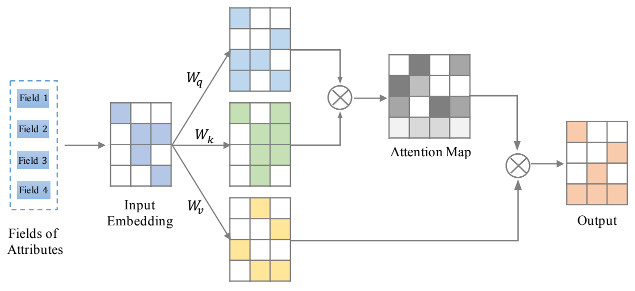


Fig. 3: Illustration of attribute interaction layer with a single self-attention head

NETTENTION learns smooth representations by jointly minimizing the reconstruction errors of autoencoders with network topological structure and vertex attributes as inputs, and the locality-preserving loss within an adversarial training process. Specifically, we consider solving the joint optimization problem with objective function

$$\begin{aligned} \mathcal{L}(\phi, \theta, \psi, \xi, w) = & \mathcal{L}_{\text{AE}}^{(g)}(\phi, \psi; \boldsymbol{x}) + \mathcal{L}_{\text{AE}}^{(a)}(\theta, \xi; \boldsymbol{z}) \\ & + \lambda_1 \mathcal{L}_{\text{LE}}(\phi; \boldsymbol{x}) + \lambda_2 W(\mathbb{P}_{\phi}(\boldsymbol{x}), \mathbb{P}_{\theta}(\boldsymbol{z})) \end{aligned} \quad (12)$$

In order to learn the interactions among vertex attributes, a.k.a cross features, an interaction layer has been introduced in the attribute space (as shown in Figure 1). In this layer, self-attention mechanism has been adopted to map the attributes of different fields, such as "author profiles" and "paper keywords", with weighted sum to the output by computing its similarity against different attribute fields, as illustrated in Figure 3. Formally, let p denote the total number of fields in the input attributes. For each field $\{\boldsymbol{z}_0^{(i)}\}_{i=1}^{p}$, a linear mapping function $\Phi(\cdot)$ is used to map the field to a low-dimensional dense vector $\Phi(\boldsymbol{z}_0^{(i)}) \in \mathbb{R}^{d_1^{(a)}}$. By applying the mapping function on all fields, the output of one instance i would be a concatenation of multiple embedding vectors denoted by $\boldsymbol{Z}_{0_{i,:}} \in \mathbb{R}^{p \times d_1^{(a)}}$,

$$\boldsymbol{Z}_{0_{i,:}} = \Phi([\boldsymbol{z}_0^{(1)}, \boldsymbol{z}_0^{(2)}, ..., \boldsymbol{z}_0^{(p)}]) \quad (13)$$

In this paper, we employ the scaled dot-product attention [16] to compute the outputs with attention weights. This self-attention mechanism consists of three parts: the *queries* $\boldsymbol{Q}$, the *keys* $\boldsymbol{K}$ and the *values* $\boldsymbol{V}$. All of these parts are derived from the same embedding $\boldsymbol{Z}_0$ with the *ReLU* activation, such that $\boldsymbol{Q} = ReLU(\boldsymbol{Z}_0\boldsymbol{W}_q)$, $\boldsymbol{K} = ReLU(\boldsymbol{Z}_0\boldsymbol{W}_k)$, and $\boldsymbol{V} = ReLU(\boldsymbol{Z}_0\boldsymbol{W}_v)$, where $\boldsymbol{W}_q, \boldsymbol{W}_k, \boldsymbol{W}_v \in \mathbb{R}^{d_1^{(a)} \times d_1^{(a)}}$ are parameters to be learned. We then compute the attention map using the *queries* and *keys* as shown in Figure 3. Each entry of the attention map represents the interaction intensity of attributes between two different fields. The output of the self-attention module is computed together with the *values*,

$$\boldsymbol{Z} = \sigma\left(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{d_1^{(a)}}}\right)\boldsymbol{V} \quad (14)$$

where $\sigma(\cdot)$ is the Softmax function. Additionally, an attribute field may also involved in the interaction in different representation subspaces. We use multi-headed attention to achieve this since it allows the model to jointly attend to information from different subspaces. The final output from the interaction layer is defined as follows,

$$\boldsymbol{Z} = \sigma\left(\frac{\boldsymbol{Q}_1\boldsymbol{K}_1^T}{\sqrt{d_1^{(a)}}}\right)\boldsymbol{V}_1 \oplus \sigma\left(\frac{\boldsymbol{Q}_2\boldsymbol{K}_2^T}{\sqrt{d_1^{(a)}}}\right)\boldsymbol{V}_2 \oplus ... \oplus \sigma\left(\frac{\boldsymbol{Q}_h\boldsymbol{K}_h^T}{\sqrt{d_1^{(a)}}}\right)\boldsymbol{V}_h \oplus \boldsymbol{Z}_0 \tag{15}$$

where $\oplus$ denotes concatenation operation, h is the number of heads and $\boldsymbol{Z} \in \mathbb{R}^{n \times d_0^{(a)}}$, $d_0^{(a)} = (h+1) \times p \times d_1^{(a)}$.

Theorem 1. *Let $\mathbb{P}_\phi(\boldsymbol{x})$ be any distribution. Let $\mathbb{P}_\theta(\boldsymbol{z})$ be the distribution of $f_\theta^{(a)}(\boldsymbol{z})$ where $\mathbf{z}$ is a sample drawn from distribution $\mathbb{P}_{attribute}(\boldsymbol{z})$ and $f_\theta^{(a)}(\cdot)$ is a function satisfying the local Lipschitz constants $\mathbb{E}_{\boldsymbol{z}\sim\mathbb{P}_g(\boldsymbol{z})}[L(\theta,\boldsymbol{z})] < +\infty$. Then we have*

$$\begin{aligned}
\nabla_\theta\mathcal{L} &= -\nabla_\theta\mathbb{E}_{\boldsymbol{x}\sim\mathbb{P}_{attribute}(\boldsymbol{x})}[dist(\boldsymbol{x}, h_\xi^{(a)}(f_\theta^{(a)}(\boldsymbol{x})))] \\
&\quad -\lambda_2\nabla_\theta\mathbb{E}_{\boldsymbol{z}\sim\mathbb{P}_{attribute}(\boldsymbol{z})}[d_w(f_\theta^{(a)}(\boldsymbol{z}))] && (16)\\
\nabla_w\mathcal{L} &= -\lambda_2\nabla_w\mathbb{E}_{\boldsymbol{x}\sim\mathbb{P}_{graph}(\boldsymbol{x})}[d_w(f_\phi^{(g)}(\boldsymbol{x}))] \\
&\quad +\lambda_2\nabla_w\mathbb{E}_{\boldsymbol{z}\sim\mathbb{P}_{attribute}(\boldsymbol{z})}[d_w(f_\theta^{(a)}(\boldsymbol{z}))] && (17)\\
\nabla_\phi\mathcal{L} &= \lambda_1\nabla_\phi\sum_{1\le i<j\le n}\|f_\phi^{(g)}(\boldsymbol{X}_{i:}) - f_\phi^{(g)}(\boldsymbol{X}_{j:})\|^2\varphi_{ij} \\
&\quad -\nabla_\phi\mathbb{E}_{\boldsymbol{x}\sim\mathbb{P}_{graph}(\boldsymbol{x})}[dist(\boldsymbol{x}, h_\psi^{(g)}(f_\phi^{(g)}(\boldsymbol{x})))] \\
&\quad +\lambda_2\nabla_\phi\mathbb{E}_{\boldsymbol{x}\sim\mathbb{P}_{graph}(\boldsymbol{x})}[d_w(f_\phi^{(g)}(\boldsymbol{x}))] && (18)\\
\nabla_\psi\mathcal{L} &= -\nabla_\psi\mathbb{E}_{\boldsymbol{x}\sim\mathbb{P}_{graph}(\boldsymbol{x})}[dist(\boldsymbol{x}, h_\psi^{(g)}(f_\phi^{(g)}(\boldsymbol{x})))] && (19)\\
\nabla_\xi\mathcal{L} &= -\nabla_\xi\mathbb{E}_{\boldsymbol{x}\sim\mathbb{P}_{attribute}(\boldsymbol{x})}[dist(\boldsymbol{x}, h_\xi^{(a)}(f_\theta^{(a)}(\boldsymbol{x})))] && (20)
\end{aligned}$$

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

$$\begin{aligned}
V(\tilde{d},\theta) &= \mathbb{E}_{\boldsymbol{y}\sim\mathbb{P}_\phi(\boldsymbol{x})}[\tilde{d}(\boldsymbol{y})] - \mathbb{E}_{\boldsymbol{y}\sim\mathbb{P}_\theta(\boldsymbol{z})}[\tilde{d}(\boldsymbol{y})] \\
&= \mathbb{E}_{\boldsymbol{y}\sim\mathbb{P}_\phi(\boldsymbol{x})}[\tilde{d}(\boldsymbol{y})] - \mathbb{E}_{\boldsymbol{z}\sim\mathbb{P}_{attribute}(\boldsymbol{z})}[\tilde{d}(f_\theta^{(a)}(\boldsymbol{z}))] \quad (21)
\end{aligned}$$

where $\tilde{d}$ lies in $\mathcal{D} = \{\tilde{d} : \mathcal{X} \to \mathbb{R}, \tilde{d}$ is continuous and bounded, $\|\tilde{d}\| \le 1\}$. Since $\mathcal{X}$ is compact, we know by the Kantorovich-Rubinstein duality [13] that there exists a $d \in \mathcal{D}$ that attains the value

$$W(\mathbb{P}_\phi(\boldsymbol{x}), \mathbb{P}_\theta(\boldsymbol{z})) = \sup_{\tilde{d}\in\mathcal{D}} V(\tilde{d},\theta) = V(d,\theta) \tag{22}$$

and $D^*(\theta) = \{d \in \mathcal{D} : V(d,\theta) = W(\mathbb{P}_\phi(\boldsymbol{x}), \mathbb{P}_\theta(\boldsymbol{z}))\}$ is non-empty. According to the envelope theorem [17], we have

$$\nabla_\theta W(\mathbb{P}_\phi(\boldsymbol{x}), \mathbb{P}_\theta(\boldsymbol{z})) = \nabla_\theta V(d,\theta) \tag{23}$$

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

$$\begin{aligned}
\nabla_\theta W(\cdot) &= \nabla_\theta V(d,\theta) \\
&= \nabla_\theta\mathbb{E}_{\boldsymbol{y}\sim\mathbb{P}_\phi(\boldsymbol{x})}[d(\boldsymbol{y})] - \mathbb{E}_{\boldsymbol{z}\sim\mathbb{P}_{attribute}(\boldsymbol{z})}[d(f_\theta^{(a)}(\boldsymbol{z}))] \\
&= -\nabla_\theta\mathbb{E}_{\mathbf{z}\sim\mathbb{P}_{attribute}(\boldsymbol{z})}[d_w(f_\theta^{(a)}(\mathbf{z}))] \quad (24)
\end{aligned}$$

By adding the loss term from the autoencoder from attribute space, we have,

$$\begin{aligned}
\nabla_\theta\mathcal{L} &= -\nabla_\theta\mathbb{E}_{\boldsymbol{x}\sim\mathbb{P}_{attribute}(\boldsymbol{x})}[\mathrm{dist}(\boldsymbol{x}, h_\xi^{(a)}(f_\theta^{(a)}(\boldsymbol{x})))] \\
&\quad -\lambda_2\nabla_\theta\mathbb{E}_{\mathbf{z}\sim\mathbb{P}_{attribute}(\boldsymbol{z})}[d_w(f_\theta^{(a)}(\mathbf{z}))] \quad (25)
\end{aligned}$$

Eq.(17)-(20) are straightforward applications of the derivative definition. □

We now have all the derivatives needed. This joint architecture requires dedicated training objective for each part. To train the model, we use block coordinate descent to alternate between optimizing different parts of the model: (1) for RNN autoencoder reconstruction error in the network topology space $\mathcal{L}_{\mathrm{AE}}^{(g)}(\phi,\psi;\boldsymbol{x})$ and locality-preserving loss $\mathcal{L}_{\mathrm{LE}}(\phi;\boldsymbol{x})$, updating ϕ and ψ; (2) for MLP autoencoder reconstruction error in the attribute space $\mathcal{L}_{\mathrm{AE}}^{(a)}(\theta,\xi;\boldsymbol{z})$, updating θ and ξ, the interaction layer is optimized as an end-to-end model; (3) for the discriminator in the adversarial training process, updating w; (4) for RNN encoder and MLP encoder, updating ϕ and θ. The pseudocode of the full approach is given in Algorithm 1.

Algorithm 1 NETTENTION Model Training

```
Require: the walks generated from input graph, network attributes
    from different fields, maximum training epoch $n_{epoch}$, the number
    of discriminator training iteration $n_D$.
1:  for $epoch = 0; epoch < n_{epoch}$ do
2:      (1) Network topology space encoding
3:      Sample $\{\boldsymbol{x}^{(i)}\}_{i=1}^B \sim \mathbb{P}_{graph}(\boldsymbol{x})$ from the walks
4:      Compute latent representation $f_\phi^{(g)}(\boldsymbol{x}^{(i)})$
5:      Compute reconstruction output $h_\psi^{(g)}(f_\phi(\boldsymbol{x}^{(i)}))$
6:      Compute $\mathcal{L}_{\mathrm{AE}}^{(g)}(\phi,\psi;\boldsymbol{x})$ and $\mathcal{L}_{\mathrm{LE}}(\phi;\boldsymbol{x})$ using Eq. (5), (7)
7:      Back propagate loss and update $\phi$ and $\psi$ using Eq. (18), (19)
8:
9:      (2) Network attribute space encoding
10:     Compute interaction layer output $\boldsymbol{Z}$ using Eq. (15)
11:     Sample $\{\boldsymbol{z}^{(i)}\}_{i=1}^B \sim \mathbb{P}_{attribute}(\boldsymbol{z})$ from the attribute vectors
12:     Compute latent representation $f_\theta^{(a)}(\boldsymbol{z}^{(i)})$
13:     Compute reconstruction output $h_\xi^{(a)}(f_\theta^{(a)}(\boldsymbol{z}^{(i)}))$
14:     Compute $\mathcal{L}_{\mathrm{AE}}^{(a)}(\theta,\xi;\boldsymbol{z})$ using Eq. (6)
15:     Back propagate loss and update $\theta$ and $\xi$ using Eq. (16), (20)
16:
17:     (3) Discriminator training
18:     for $n = 0, n < n_D$ do
19:         Sample $\{\boldsymbol{x}^{(i)}\}_{i=1}^B \sim \mathbb{P}_{graph}(\boldsymbol{x})$ from the walks
20:         Sample $\{\boldsymbol{z}^{(i)}\}_{i=1}^B \sim \mathbb{P}_{attribute}(\boldsymbol{z})$ from the attribute vectors
21:         Compute representations $f_\phi^{(g)}(\boldsymbol{x}^{(i)})$ and $f_\theta^{(a)}(\boldsymbol{z}^{(i)})$
22:         Compute $\mathcal{L}_{\mathrm{DIS}}(w;\boldsymbol{x},\boldsymbol{z})$ using Eq.(11)
23:         Back propagate loss and update $w$ using Eq.(17)
24:     end for
25:
26:     (4) RNN/MLP encoders adversarial training
27:     Sample $\{\boldsymbol{x}^{(i)}\}_{i=1}^B \sim \mathbb{P}_{graph}(\boldsymbol{x})$ from the walks
28:     Sample $\{\boldsymbol{z}^{(i)}\}_{i=1}^B \sim \mathbb{P}_{attribute}(\boldsymbol{z})$ from the attribute vectors
29:     Compute representations $f_\phi^{(g)}(\boldsymbol{x}^{(i)})$ and $f_\theta^{(a)}(\boldsymbol{z}^{(i)})$
30:     Compute $\mathcal{L}_{\mathrm{GEN}}(\theta,\phi;\boldsymbol{x},\boldsymbol{z})$ using Eq.(10)
31:     Backpropagate loss and update $\theta$ and $\phi$ using Eq.(16), (18)
32: end for
```

With this generative adversarial training, the latent space of RNN autoencoder provides an optimal embedding of the vertices with the guided information from the attribute space. Notably, the usage of RNN autoencoder in the topology space takes the vertex order information of the sampled walks into consideration, which is well-suited for vertex representation learning. After the training of NETTENTION, we obtain vertex representations $f_\phi^{(g)}(\boldsymbol{x})$ of the network by passing the input walks through the encoder function.

Theoretical Analysis. We now show the connection between the optimization over the NETTENTION objective function $\mathcal{L}(\phi, \theta, \psi, \xi, w)$ and the optimization of the autoencoder reconstruction errors defined with optimal transport cost [18], [19]. From the analysis we will see that, the adversarial training process in NETTENTION is equivalent to optimize the optimal transport cost between the input and output distributions of RNN/MLP autoencoders, which enforcing the latent embeddings in topology space and attribute space to follow the same prior distribution.

Let $\boldsymbol{x} \sim \mathbb{P}_{graph}$ denote the samples drawn from the input, $\boldsymbol{y} \sim \mathbb{P}_{prior}$ denote the samples drawn from a prior distribution in the latent space, $f_\phi^{(g)}$ and $h_\psi^{(g)}$ denote the encoding and decoding functions, and $\mathbb{P}_\phi$ and $\mathbb{P}_\psi$ denote the corresponding distributions, respectively.

Theorem 2. *For $\mathbb{P}_\psi$ defined above with deterministic $\mathbb{P}_\psi(\boldsymbol{y}|\boldsymbol{x})$ and any function $h_\psi^{(g)} : \mathcal{Y} \mapsto \mathcal{X}$,*

$$\begin{aligned} W_c(\mathbb{P}_{graph}, \mathbb{P}_\psi) & \inf_{\Gamma \in \mathcal{P}(\boldsymbol{x} \sim \mathbb{P}_{graph}, \boldsymbol{y} \sim \mathbb{P}_{prior})} \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y}) \sim \Gamma}[c(\boldsymbol{x}, h_\psi^{(g)}(\boldsymbol{y}))] \\ & \inf_{\mathbb{P}_\phi : \mathbb{P}_\phi = \mathbb{P}_{prior}} \mathbb{E}_{\boldsymbol{x} \sim \mathbb{P}_{graph}} \mathbb{E}_{\mathbb{P}_\phi(\boldsymbol{y}|\boldsymbol{x})}[c(\boldsymbol{x}, h_\psi^{(g)}(\boldsymbol{y}))], \end{aligned} \tag{26}$$

where $\mathbb{P}_\phi$ is the marginal distribution of $\boldsymbol{y}$ when $\boldsymbol{x} \sim \mathbb{P}_{graph}$ and $\boldsymbol{y} \sim \mathbb{P}_\phi(\boldsymbol{y}|\boldsymbol{x})$ [19].

Based on Theorem 2, the objective function of autoencoder with optimal transport cost (Wasserstein autoencoder) is defined by [20],

$$\mathcal{L}_{\mathrm{AE}}^{(g)} = \inf_{\mathbb{P}_\phi(\boldsymbol{y}|\boldsymbol{x}) \in \mathcal{F}} \mathbb{E}_{\boldsymbol{x} \sim \mathbb{P}_{graph}} \mathbb{E}_{\mathbb{P}_\phi(\boldsymbol{y}|\boldsymbol{x})}[c(\boldsymbol{x}, h_\psi^{(g)}(\boldsymbol{y}))] + \lambda \cdot \mathcal{D}(\mathbb{P}_\phi, \mathbb{P}_{prior}) \tag{27}$$

where $\mathcal{F}$ is any nonparametric set of probabilistic encoders, $\mathcal{D}$ is an arbitrary divergence between $\mathbb{P}_\phi$ and $\mathbb{P}_{prior}$, $\lambda > 0$ is a hyper parameter. Similarly, we can define the Wasserstein autoencoder in the attribute space,

$$\mathcal{L}_{\mathrm{AE}}^{(a)} = \inf_{\mathbb{P}_\theta(\boldsymbol{y}|\boldsymbol{z}) \in \mathcal{F}} \mathbb{E}_{\boldsymbol{z} \sim \mathbb{P}_{attribute}} \mathbb{E}_{\mathbb{P}_\theta(\boldsymbol{y}|\boldsymbol{z})} \left[c(\boldsymbol{z}, h_\xi^{(a)}(\boldsymbol{y}))\right] + \lambda \cdot \mathcal{D}(\mathbb{P}_\theta, \mathbb{P}_{prior}) \tag{28}$$

Theorem 3. *The optimization over the* NETTENTION *objective function $\mathcal{L}(\phi, \theta, \psi, \xi, w)$ is equivalent to minimize the locality-preserving loss and the Wasserstein autoencoders from the network topology space and attribute space with same prior distribution $\mathbb{P}_{prior}$.*

Proof. Given $\mathrm{dist}(\boldsymbol{x}, \boldsymbol{y}) = c(\boldsymbol{x}, \boldsymbol{y})$, evidently the first term in $\mathcal{L}_{\mathrm{AE}}^{(g)}$ and $\mathcal{L}_{\mathrm{AE}}^{(a)}$ are equal to the autoencoder losses defined in Eq. (5) and Eq. (6). That is,

$$\inf_{\mathbb{P}_\phi(\boldsymbol{y}|\boldsymbol{x}) \in \mathcal{F}} \mathbb{E}_{\boldsymbol{x} \sim \mathbb{P}_{graph}} \mathbb{E}_{\mathbb{P}_\phi(\boldsymbol{y}|\boldsymbol{x})}[c(\boldsymbol{x}, h_\psi^{(g)}(\boldsymbol{y}))] = \mathcal{L}_{\mathrm{AE}}^{(g)}(\phi, \psi; \boldsymbol{x}) \tag{29}$$

$$\inf_{\mathbb{P}_\theta(\boldsymbol{y}|\boldsymbol{z}) \in \mathcal{F}} \mathbb{E}_{\boldsymbol{z} \sim \mathbb{P}_{attribute}} \mathbb{E}_{\mathbb{P}_\theta(\boldsymbol{y}|\boldsymbol{z})} \left[c(\boldsymbol{z}, h_\xi^{(a)}(\boldsymbol{y}))\right] = \mathcal{L}_{\mathrm{AE}}^{(a)}(\theta, \xi; \boldsymbol{z}) \tag{30}$$

Additionally, we have two Wasserstein autoencoders following the same prior distribution $\mathbb{P}_{prior}$, thus,

$$\mathcal{L}_{\mathrm{AE}}^{(g)} + \mathcal{L}_{\mathrm{AE}}^{(a)} = \mathcal{L}_{\mathrm{AE}}^{(g)}(\phi, \psi; \boldsymbol{x}) + \mathcal{L}_{\mathrm{AE}}^{(a)}(\theta, \xi; \boldsymbol{z}) + \lambda \cdot \mathcal{D}(\mathbb{P}_\phi, \mathbb{P}_\theta) \tag{31}$$

$\mathcal{D}(\cdot)$ is defined with *earth mover distance*, then we have,

$$\begin{aligned} \mathcal{L} &= \mathcal{L}_{\mathrm{AE}}^{(g)}(\phi, \psi; \boldsymbol{x}) + \mathcal{L}_{\mathrm{AE}}^{(a)}(\theta, \xi; \boldsymbol{z}) + \lambda_1 \mathcal{L}_{\mathrm{LE}}(\phi; \boldsymbol{x}) + \lambda_2 W(\mathbb{P}_\phi(\boldsymbol{x}), \mathbb{P}_\theta(\boldsymbol{z})) \\ &= \mathcal{L}_{\mathrm{AE}}^{(g)}(\phi, \psi; \boldsymbol{x}) + \mathcal{L}_{\mathrm{AE}}^{(a)}(\theta, \xi; \boldsymbol{z}) + \lambda_1 \mathcal{L}_{\mathrm{LE}}(\phi; \boldsymbol{x}) + \lambda_2 \cdot \mathcal{D}(\mathbb{P}_\phi, \mathbb{P}_\theta) \\ &= \mathcal{L}_{\mathrm{AE}}^{(g)} + \mathcal{L}_{\mathrm{AE}}^{(a)} + \lambda_1 \mathcal{L}_{\mathrm{LE}}(\phi; \boldsymbol{x}) \end{aligned}$$

□

Computational Cost. Given a network $G(\mathcal{V}, \mathcal{E}, \boldsymbol{Z})$, where $|\mathcal{V}| = n, |\mathcal{E}| = m$, according to the definition in Eq.(2), the overall complexity of Laplacian Eigenmaps embedding is $\mathcal{O}(n^2)$. In the implementation, we only consider the vertex pairs $(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)})$ that have edges between them, thus the size of the sampled pairs reduced to $\mathcal{O}(m)$. The computational complexity of learning autoencoders is proportional to the number of parameters in each iteration. Therefore, the learning computational complexity for RNN/MLP autoencoders is $\mathcal{O}(n_{epoch} \times (|\phi| + |\theta|))$. Similarly, the computational complexity for discriminator is $\mathcal{O}(n_{epoch} \times (n_D \times |w|))$.

IV. EVALUATION

A. Datasets and Baselines

To verify the performance of the proposed NETTENTION model, we conduct experiments on a variety of networks summarized in Table I.

- The Cora and CiteSeer datasets consist of scientific publications classified into different classes. Each publication in the dataset is described by a 0/1-valued word vector indicating the absence/presence of the corresponding word from the dictionary.
- Wikipedia (Wiki) [21] is a directed word network. Vertex labels represent the Part-of-Speech (POS) tags inferred using the Stanford POS-Tagger [22].
- The PubMed Diabetes dataset consists of scientific publications from PubMed database pertaining to diabetes classified into one of three classes. Each publication in the dataset is described by a TF/IDF weighted word vector from a dictionary which consists of 500 unique words.

We compare our proposed model with baselines in three categories: (1) baselines only using network topology features, such as DeepWalk [23], node2vec [21] and SDNE [3]. (2) baselines only using network attributes, including XG-Boost [24] and Logistic Regression (LR). And (3) baselines

TABLE I: Statistics of the real-world network datasets

Dataset	$\lvert V \rvert$	$\lvert E \rvert$	**#Attribute**	**#label**
Citeseer	3,312	4,660	3,703	6
Cora	2,708	5,278	1,433	7
Wiki	2,405	12,761	4,973	17
PubMed	19,717	44,338	500	3

combining network topology and attributes to learn final representations, including TADW [5], GAE and VGAE [7], AANE [6], SAGE [8], DANE [9] and G2G [10]. Detailed descriptions of baselines can be found as follows,

Network topology based baselines:

- DeepWalk [23]: DeepWalk is a skip-gram [25] based model which learns the graph embedding with truncated random walks.
- node2vec [21]: 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) [3]: 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.

Attribute based baselines:

- XGBoost [24]: XGBoost provides a parallel tree boosting (also known as GBDT, GBM) that solve many data science problems in a fast and accurate way. We use XGBoost for the vertex classification task.
- Logistic Regression (LR): Logistic regression (or logit regression) is estimating the parameters of a logistic model; it is a form of binomial regression.

Attributed network embedding baselines:

- TADW [5]: The text-associated DeepWalk (TADW) incorporates text features of vertices into network representation learning under the framework of matrix factorization.
- GAE and VGAE [7]: VGAE/GAE is a framework for unsupervised learning on graph-structured data based on the (variational) auto-encoder.
- AANE [6]: Accelerated attributed network embedding (AANE) learns an effective unified embedding representation by incorporating node attribute proximity into network embedding.
- SAGE [8]: SAGE is a general inductive framework that learns a function to generate embeddings by sampling and aggregating features from a node's local neighborhood.
- DANE [9]: DANE is a deep attributed network embedding approach, which can capture the high nonlinearity and preserve various proximities in both topological structure and node attributes.
- G2G [10]: G2G embeds each node as a Gaussian distribution, allowing us to capture uncertainty about the representation.

After obtaining the features from network topology, we create another versions of the above network embedding algorithms in category (1) which concatenated the attributes as the final vertex representations. These variants are denoted as DeepWalk*, node2vec* and SDNE*. A variant of the NETTENTION model *without* self-attention module is denoted as NETTENTION*.

B. Experimental Settings

For fair comparison, 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 20 with walk length 30, which is the same as the random walk generation step of NETTENTION. The window size of DeepWalk and node2vec is optimized to 10. For the rest baselines, we utilize the default parameter setting as described in each paper. The gradient clipping is performed in every training iteration to avoid the gradient explosion in NETTENTION. LSTM is adopted for RNN encoder and decoder with stochastic gradient descent as the optimizer in training. In the interaction layer with self-attention, the embedding dimension $d_1^{(a)}$ for each attribute field is set to 16, and the number of heads h is set to 4. A 6-layer MLP autoencoder is used in learning attribute representations, and a 3-layer MLP is used in discriminator. The hyper parameters λ_1 and λ_2 are set 1 in the experiments. The MLP autoencoder and discriminator are optimized using Adam [26]. The evaluation of different algorithms is based on the vertex classification and link prediction accuracy using the learned embeddings, as illustrated in the subsequent sections.

C. Classification

In this section, we use vertex features as input to a one-vs-rest logistic regression to train the classifiers. To make a comprehensive evaluation, we randomly select 10%, 30%, 50% of total number of vertices as the training set and use the remaining vertices as the test set. We report Micro-F1 (Mi-F1) and Macro-F1 (Ma-F1) as evaluation metrics. Each result is averaged over 10 trials, as shown in Table II.

It is evident from the figure that NETTENTION outperforms the state-of-the-art embedding algorithms on the vertex classification task on all four datasets. Among all baselines, G2G and NETTENTION are very comparable with small training percentage. However, as the training percentage goes from 10% to 50%, the performance of NETTENTION gets better and beats all baselines by a large margin. Comparing the performance of topology-based baselines, attribute-based baselines and attributed network embedding baselines, we observe that vertex attribute information plays an important role in the classification task which cannot be ignored, and better accuracy will be achieved if both network topology and attributes are considered. Notably, the constrained version of our method NETTENTION* that does not consider the attribute interactions is still able to outperform most of the competing approaches.

TABLE II: Classification performance comparison

Methods	Citeceer						Cora					
	`10% Training`		`30% Training`		`50% Training`		`10% Training`		`30% Training`		`50% Training`	
	Mi-F1	Ma-F1	Mi-F1	Ma-F1	Mi-F1	Ma-F1	Mi-F1	Ma-F1	Mi-F1	Ma-F1	Mi-F1	Ma-F1
DeepWalk	0.5046	0.4681	0.5897	0.5316	0.6017	0.5536	0.7041	0.6920	0.7290	0.7064	0.7583	0.7226
SDNE	0.5183	0.4794	0.5910	0.5399	0.6164	0.5812	0.7087	0.6943	0.7319	0.7069	0.7483	0.7173
node2vec	0.5319	0.4917	0.6159	0.5775	0.6400	0.6035	0.7230	0.7094	0.7394	0.7112	0.7662	0.7415
XGBoost	0.6391	0.5777	0.7031	0.6494	0.7120	0.6697	0.7545	0.7585	0.8100	0.7984	0.8400	0.8192
LR	0.6461	0.5931	0.6951	0.6531	0.7190	0.6724	0.7590	0.7566	0.8048	0.8019	0.8421	0.8242
DeepWalk*	0.6548	0.6017	0.6950	0.6524	0.7296	0.6797	0.7628	0.7381	0.7982	0.7906	0.8339	0.8127
SDNE*	0.6583	0.6166	0.7070	0.6601	0.7312	0.6887	0.7333	0.7095	0.8011	0.8026	0.8390	0.8223
node2vec*	0.6662	0.6126	0.7068	0.6585	0.7453	0.6932	0.7526	0.7309	0.8208	0.8191	0.8218	0.8053
TADW	0.6134	0.5516	0.6644	0.5897	0.6600	0.5985	0.7538	0.7211	0.7942	0.7697	0.8270	0.8102
AANE	0.5979	0.5489	0.6783	0.6208	0.7075	0.6640	0.7301	0.7226	0.8034	0.7822	0.8003	0.8054
GAE	0.6103	0.5560	0.6530	0.5843	0.6662	0.5962	0.7200	0.7199	0.8015	0.7807	0.8077	0.8000
VGAE	0.6116	0.5777	0.6458	0.5987	0.6470	0.5870	0.7620	0.7495	0.8078	0.7829	0.8002	0.7935
SAGE	0.5487	0.5102	0.6376	0.6081	0.6629	0.6141	0.7530	0.7481	0.8083	0.8003	0.8222	0.7984
DANE	0.6509	0.6162	0.7271	0.6736	0.7463	0.6733	0.7872	0.7634	0.8250	0.8122	0.8442	0.8355
G2G	0.6887	0.6218	0.7169	0.6810	0.7276	0.6424	0.7858	0.7628	0.8299	0.7995	0.8324	0.8161
Nettention*	0.6914	0.6202	0.7305	0.6830	0.7441	0.6810	0.7939	0.7743	0.8306	0.8149	0.8496	0.8382
Nettention	**0.6956**	**0.6306**	**0.7428**	**0.6927**	**0.7563**	**0.6942**	**0.7971**	**0.7781**	**0.8479**	**0.8209**	**0.8522**	**0.8461**

Methods	Wiki						PubMed					
	`10% Training`		`30% Training`		`50% Training`		`10% Training`		`30% Training`		`50% Training`	
	Mi-F1	Ma-F1	Mi-F1	Ma-F1	Mi-F1	Ma-F1	Mi-F1	Ma-F1	Mi-F1	Ma-F1	Mi-F1	Ma-F1
DeepWalk	0.5559	0.4437	0.6378	0.5254	0.6634	0.6011	0.7828	0.7344	0.7892	0.7753	0.7954	0.7837
SDNE	0.5769	0.4663	0.6479	0.5327	0.6658	0.5673	0.7668	0.7465	0.8097	0.7660	0.7936	0.7781
node2vec	0.5576	0.4158	0.6070	0.4805	0.6412	0.5109	0.7860	0.7422	0.7969	0.7645	0.8025	0.7888
XGBoost	0.6240	0.5947	0.6629	0.6003	0.6871	0.6235	0.8191	0.7805	0.8215	0.7943	0.8232	0.8088
LR	0.6291	0.5928	0.6811	0.6059	0.6998	0.6203	0.8238	0.7798	0.8339	0.8016	0.8403	0.8217
DeepWalk*	0.6119	0.5820	0.6775	0.6097	0.7038	0.6213	0.8001	0.7834	0.8146	0.7999	0.8177	0.7994
SDNE*	0.6206	0.5739	0.6859	0.5982	0.6889	0.6296	0.7958	0.7793	0.8163	0.8018	0.8101	0.8071
node2vec*	0.6252	0.5827	0.6825	0.6068	0.7026	0.6316	0.7924	0.7751	0.8069	0.7990	0.8020	0.7941
TADW	0.7157	0.6223	0.7505	0.6448	0.7754	0.6530	0.8392	0.8380	0.8521	0.8570	0.8565	0.8637
AANE	0.6247	0.5357	0.7033	0.5908	0.7275	0.6847	0.7898	0.7813	0.8150	0.8093	0.8179	0.8114
GAE	0.6192	0.5136	0.6507	0.5023	0.6625	0.5073	0.8243	0.8239	0.8325	0.8269	0.8369	0.8213
VGAE	0.6506	0.5143	0.6849	0.5647	0.7081	0.5818	0.8289	0.8313	0.8316	0.8196	0.8382	0.8211
SAGE	0.6181	0.5170	0.6770	0.5925	0.6893	0.5966	0.8096	0.8056	0.8220	0.8067	0.8259	0.8208
DANE	0.7354	0.6148	0.7500	0.6633	0.7729	0.6865	0.8316	0.8298	0.8433	0.8306	0.8418	0.8503
G2G	0.7231	0.6072	0.7514	0.6422	0.7714	0.6873	0.8306	0.8230	0.8402	0.8337	0.8514	0.8461
Nettention*	0.7274	0.6240	0.7701	0.6638	0.7879	0.6864	0.8210	0.8244	0.8534	0.8311	0.8623	0.8705
Nettention	**0.7362**	**0.6300**	**0.7827**	**0.6710**	**0.8021**	**0.6973**	**0.8336**	**0.8300**	**0.8626**	**0.8416**	**0.8697**	**0.8777**

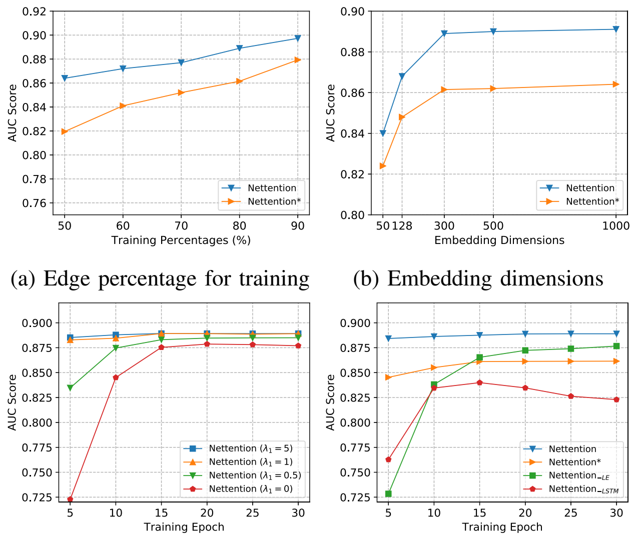
TABLE III: Link prediction performance comparison measured by AUC score

Methods	**Citeceer**	**Cora**	**Wiki**	**PubMed**
DeepWalk	0.7562	0.8024	0.7630	0.8137
SDNE	0.7605	0.8230	0.7888	0.8057
node2vec	0.7517	0.8191	0.7693	0.8224
XGBoost	0.8194	0.7833	0.8374	0.8352
LR	0.8176	0.7836	0.8421	0.8458
DeepWalk*	0.8230	0.8336	0.8206	0.8959
SDNE*	0.8255	0.8311	0.8234	0.8818
node2vec*	0.8318	0.8341	0.8402	0.8934
TADW	0.7557	0.7630	0.7905	0.8649
AANE	0.7808	0.7808	0.8077	0.8529
GAE	0.8077	0.8282	0.8363	0.8941
VGAE	0.8169	0.8212	0.8523	0.8984
SAGE	0.8132	0.8490	0.8375	0.9046
DANE	0.8435	0.8634	0.8860	0.9130
G2G	0.8586	0.8933	0.8798	0.9166
Nettention*	0.8615	0.9019	0.8941	0.9361
Nettention	**0.8890**	**0.9181**	**0.9072**	**0.9413**

D. Link Prediction

Link prediction is a common task to demonstrate the effectiveness of embedding models. The objective of link prediction task is to infer missing edges given a network with a certain fraction of edges removed. In the experiments, we randomly remove 20% of edges from the network, which serve as positive samples, and select an equal number of vertex pairs without edges between them as negative samples. With the vertex representation learned by network embedding algorithms, we obtain the link prediction ranking score from the ℓ_2 norm of two vertex vectors. We report the area under curve (AUC) score in link prediction task by convention. The results for different datasets are shown in Table III.

Obviously, NETTENTION outperforms the baseline algorithms across all datasets by a large margin, which is a strong sign that the learned embeddings are useful. In detail, NETTENTION achieves 3% to 19% improvement based in AUC score on these four datasets. Note that, the constrained



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

(c) Performance on different λ_1 (d) NETTENTION variants

Fig. 4: Model analysis on Citeceer dataset

version of our model NETTENTION* also outperforms all competitors. Another observation is that, the baselines based solely on attributes achieve surprisingly strong performance on some datasets, which is even better than some of the attributed network embedding methods, which is another strong sign to show the importance of vertex attributes. By comparing NETTENTION, AANE and DANE, which all seek to minimize the discrepancies between topology space and attribute space, we can see the effectiveness of generative adversarial regularization for improving the generalization performance in NETTENTION model, which shows NETTENTION can overcome the sparsity issue existing in attributed network embedding.

E. Model Analysis

In this section, we investigate the performance of the proposed model using different parameter set and model architectures on link prediction task. We study how the training set size, embedding dimension and locality-preserving constraint parameter λ_1 will affect the link prediction performance. Additionally, by changing the architecture of the NETTENTION model, we can investigate the importance of different components in NETTENTION.

In Figure 4(a), the training percentage of edges in the Citeceer network changes from 50% to 90%. It can be seen that the performance of NETTENTION and NETTENTION* increases as the training percentage increases. Notably, with a small proportion of training edges, the gap of AUC scores between NETTENTION and NETTENTION* (that has no interaction layer) is larger, which demonstrates the generalization capability of the NETTENTION model with self-attention mechanism. In Figure 4(b), we study the effects of embedding dimension. The prediction performance gets saturated as the dimension increases from 50 to 1000. Since the embedding dimension is proportional to the model parameter size in NETTENTION, we choose to set the embedding size to 300 as a trade-off between the performance and efficiency of model training. We also investigate the impact of λ_1, the relative strength between locality-preserving constraint and autoencoder constraint. The higher the λ_1, the larger the gradient coming from the locality-preserving constraint. As observed from the Figure 4(c), a higher λ_1 not only helps the training converge fast, but also enhances the link prediction performance on the Citeceer network, indicating the importance of network neighborhood information.

We also include three variants of NETTENTION to demonstrate the importance of individual components, including NETTENTION*, NETTENTION$_{-LE}$ and NETTENTION$_{-LSTM}$. As described in the previous section, NETTENTION* removes the interaction layer in the NETTENTION model. Similarly, NETTENTION$_{-LE}$ removes the locality-preserving constraint, and we replace RNN autoencoder with MLP autoencoder in NETTENTION$_{-LSTM}$. It is evident from Figure 4(d) that RNN autoencoder, locality-preserving constraint, and self-attention mechanism play important roles in NETTENTION model. Overfitting becomes obvious in the training of NETTENTION$_{-LSTM}$.

V. RELATED WORK

Attributed Network Embedding. Recently, a wide variety of models have been proposed for attributed networks [5]–[10], which show that jointly learning network representations with network topology information and vertex attributes enhance the performance on various tasks. The text-associated DeepWalk [5] incorporates text features of vertices into network representation learning under the framework of matrix factorization. Similarly, an accelerated attributed network embedding learns an effective unified embedding representation by incorporating node attribute proximity into network embedding [6]. However, these matrix factorization based shallow models are not enough for the complicated attributed networks. More deep models have been proposed, including graph convolutional neural network models [7], deep autoencoder based models [9] and models that can handle inductive learning scenarios [8], [10], to learn the non-linearity in the representation space.

Generative Adversarial Networks. Generative adversarial networks (GANs) [12] have achieved great success in generating and learning the latent presentation of high-dimensional data such as images [27]. There have been several successful attempts [20], [28]–[30] of implementing GANs on discrete structures, such as text and discrete images. Using GANs to learn the representation of discrete contents like natural languages and network embedding remains a challenging problem due to the difficulty in back-propagation through discrete random variables [30]. Recent work on GANs such as GraphGAN [31], NetRA [32] and ANE [33] for discrete data is either though the use of discrete structures or the improved autoencoders.

Attention Mechanisms. Motivated by visual attention to different regions of an image or correlated words in one sentence, attention mechanisms become a fundamental part of deep neural network models that use to capture the global dependencies [34]–[36]. In particular, self-attention [16], also known as intra-attention, relates different positions of a single sequence by attending to all positions within the same sequence. It has achieved state-of-the-art results by solely using a self-attention model in machine translation [16]. Self-attention mechanism is also been widely used in various perceptual tasks such as image generation [37] and translation [38].

VI. Conclusion

In this paper, we propose NETTENTION, an adversarially regularized embedding learning approach that can efficiently learn vertex embeddings on attributed networks that well capture the network structure through minimizing a penalized form of the Wasserstein distance between the distributions learned from the topology space and the attribute space so as to learn a consensus and complementary representation. The joint inference is encapsulated in a generative adversarial training process, yielding better generalization performance and robustness. Additionally, a self-attention module is developed to explicitly model the attribute interactions. The effectiveness has been verified by extensive experiments including vertex classification and link prediction.

Acknowledgement

The work is partially supported by NSF DGE-1829071. We thank the anonymous reviewers for their careful reading and insightful comments on our manuscript.

References

[1] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 701–710, ACM, 2014.

[2] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 855–864, ACM, 2016.

[3] D. Wang, P. Cui, and W. Zhu, "Structural deep network embedding," in *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 1225–1234, ACM, 2016.

[4] M. McPherson, L. Smith-Lovin, and J. M. Cook, "Birds of a feather: Homophily in social networks," *Annual review of sociology*, vol. 27, no. 1, pp. 415–444, 2001.

[5] C. Yang, Z. Liu, D. Zhao, M. Sun, and E. Y. Chang, "Network representation learning with rich text information.," in *IJCAI*, pp. 2111–2117, 2015.

[6] X. Huang, J. Li, and X. Hu, "Accelerated attributed network embedding," in *Proceedings of the 2017 SIAM International Conference on Data Mining*, pp. 633–641, SIAM, 2017.

[7] T. N. Kipf and M. Welling, "Variational graph auto-encoders," *arXiv preprint arXiv:1611.07308*, 2016.

[8] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," *arXiv preprint arXiv:1706.02216*, 2017.

[9] H. Gao and H. Huang, "Deep attributed network embedding," in *IJCAI*, pp. 3364–3370, 2018.

[10] A. Bojchevski and S. Günnemann, "Deep gaussian embedding of graphs: Unsupervised inductive learning via ranking," in *ICLR*, 2018.

[11] W. Yu, G. Zeng, P. Luo, F. Zhuang, Q. He, and Z. Shi, "Embedding with autoencoder regularization," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 208–223, Springer, 2013.

[12] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *NIPS*, pp. 2672–2680, 2014.

[13] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein generative adversarial networks," in *ICML*, pp. 214–223, 2017.

[14] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*, vol. 1. MIT Press, 2016.

[15] A. Makhzani, J. Shlens, N. Jaitly, and I. Goodfellow, "Adversarial autoencoders," in *ICLR*, 2016.

[16] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, pp. 5998–6008, 2017.

[17] P. Milgrom and I. Segal, "Envelope theorems for arbitrary choice sets," *Econometrica*, vol. 70, no. 2, pp. 583–601, 2002.

[18] C. Villani, *Optimal transport: old and new*, vol. 338. Springer Science & Business Media, 2008.

[19] O. Bousquet, S. Gelly, I. Tolstikhin, C.-J. Simon-Gabriel, and B. Schoelkopf, "From optimal transport to generative modeling: the vegan cookbook," *arXiv preprint arXiv:1705.07642*, 2017.

[20] I. Tolstikhin, O. Bousquet, S. Gelly, and B. Schoelkopf, "Wasserstein auto-encoders," *arXiv preprint arXiv:1711.01558*, 2017.

[21] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *KDD*, pp. 855–864, ACM, 2016.

[22] K. Toutanova, D. Klein, C. D. Manning, and Y. Singer, "Feature-rich part-of-speech tagging with a cyclic dependency network," pp. 173–180, Association for Computational Linguistics, 2003.

[23] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *KDD*, pp. 701–710, ACM, 2014.

[24] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pp. 785–794, ACM, 2016.

[25] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *NIPS*, pp. 3111–3119, 2013.

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

[27] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," *arXiv preprint arXiv:1511.06434*, 2015.

[28] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville, "Improved training of wasserstein gans," *arXiv preprint arXiv:1704.00028*, 2017.

[29] S. Rajeswar, S. Subramanian, F. Dutil, C. Pal, and A. Courville, "Adversarial generation of natural language," *arXiv preprint arXiv:1705.10929*, 2017.

[30] Y. Kim, K. Zhang, A. M. Rush, Y. LeCun, *et al.*, "Adversarially regularized autoencoders for generating discrete structures," *arXiv preprint arXiv:1706.04223*, 2017.

[31] H. Wang, J. Wang, J. Wang, M. Zhao, W. Zhang, F. Zhang, X. Xie, and M. Guo, "Graphgan: Graph representation learning with generative adversarial nets," *AAAI*, 2018.

[32] W. Yu, C. Zheng, W. Cheng, C. C. Aggarwal, D. Song, B. Zong, H. Chen, and W. Wang, "Learning deep network representations with adversarially regularized autoencoders," in *KDD*, pp. 2663–2671, ACM, 2018.

[33] Q. Dai, Q. Li, J. Tang, and D. Wang, "Adversarial network embedding," *arXiv preprint arXiv:1711.07838*, 2017.

[34] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[35] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, vol. 1, no. 2, 2017.

[36] W. Song, C. Shi, Z. Xiao, Z. Duan, Y. Xu, M. Zhang, and J. Tang, "Autoint: Automatic feature interaction learning via self-attentive neural networks," *arXiv preprint arXiv:1810.11921*, 2018.

[37] H. Zhang, I. Goodfellow, D. Metaxas, and A. Odena, "Self-attention generative adversarial networks," *arXiv preprint arXiv:1805.08318*, 2018.

[38] N. Parmar, A. Vaswani, J. Uszkoreit, Ł. Kaiser, N. Shazeer, and A. Ku, "Image transformer," *arXiv preprint arXiv:1802.05751*, 2018.