A Graph Auto-Encoder for Attributed Network Embedding

Keting Cen, Huawei Shen, Jinhua Gao, Qi Cao, Bingbing Xu, Xueqi Cheng
CAS Key Laboratory of Network Data Science and Technology,
Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China
University of Chinese Academy of Sciences, Beijing, China
{cenketing,shenhuawei,gaojinhua,caoqi,xubingbing,cxq}@ict.ac.cn

ABSTRACT

Attributed network embedding aims to learn low-dimensional node representations from both network structure and node attributes. Existing methods can be categorized into two groups: (1) the first group learns two separated node representations from network structure and node attribute respectively and concatenating them together; (2) the other group obtains node representations by translating node attributes into network structure or vice versa. However, both groups have their drawbacks. The first group neglects the correlation between these two types of information, while the second group assumes strong dependence between network structure and node attributes. In this paper, we address attributed network embedding from a novel perspective, i.e., learning representation of a target node via modeling its attributed local subgraph. To achieve this goal, we propose a novel graph auto-encoder framework, namely GraphAE. For a target node, GraphAE first aggregates the attribute information from its attributed local subgrah, obtaining its low-dimensional representation. Next, GraphAE diffuses its representation to nodes in its local subgraph to reconstruct their attribute information. Our proposed perspective transfroms the problem of learning node representations into the problem of modeling the context information manifested in both network structure and node attributes, thus having high capacity to learn good node representations for attributed network. Extensive experimental results on real-world datasets demonstrate that the proposed framework outperforms the state-of-the-art network approaches at the tasks of link prediction and node classification.

CCS CONCEPTS

• **Information systems** → **Social networks**.

KEYWORDS

Attributed Network Embedding, Graph Auto-encoder, Attributed local subgraph

ACM Reference Format:
Keting Cen, Huawei Shen, Jinhua Gao, Qi Cao, Bingbing Xu, Xueqi Cheng. 2018. A Graph Auto-Encoder for Attributed Network Embedding. In *Proceedings of Woodstock '18: ACM Symposium on Neural Gaze Detection (Woodstock '18)*. ACM, New York, NY, USA, Article 4, 11 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Networks are ubiquitous in nature and society, including social networks, information networks, biological networks and various technological networks. The complex structure of networks poses big challenge for data mining tasks dealing with networks. To combat this challenge, researchers resort to network embedding, i.e., learning low-dimensional representation for each node to capture and preserve the network structure [9, 15, 32]. With the learned representations of nodes, many downstream mining and prediction tasks on networks, e.g., node classification and link prediction, can be easily addressed using standard machine learning tools.

Many network embedding methods, unsupervised or supervised, have been proposed and successfully applied to node classification and link prediction [4, 9, 29, 36]. These methods learn representations of nodes leveraging structural proximity or structural similarity among nodes. However, in many real world networks, nodes are usually associated with rich attributes, e.g., content of articles in citation network [25] and user profile in social networks [33]. This motivates researchers to study the problem of attributed network embedding.

Attributed network embedding aims to learn a sole low-dimensional representation for each node by simultaneously considering the information manifested in both network structure and the node attributes [19, 20, 26]. Existing methods for attributed network embedding mainly fall into two paradigms. Methods in the first paradigm learn separated representations for each node according to network structure and node attributes respectively, and then concatenate them into a single representation [12, 26]. Methods in the other paradigm attempt to directly obtain a single representation for each node by translating node attributes into network structure or vice versa [27, 42]. However, methods in both paradigms have their drawbacks. Methods in the first paradigm neglect the correlation between these two types of information, while the second paradigm assumes strong dependence between node attributes and network structure. Thus we are still lack of an effective method for attributed network embedding.

In this paper, we propose a novel perspective to address attributed network embedding. Unlike previous methods, we attempt to learn node representations by modeling the attributed local subgraph of each node. A node's attributed local subgraph is defined as the subgraph centered at the target node together with associated node attributes. This perspective transfroms the problem of learning node representations into the problem of modeling the context information manifested in both network structure and node attributes. Motivated by this perspective, we propose a novel graph

Woodstock '18, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-1-4503-9999-9/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

auto-encoder framework, namely GraphAE, for attributed network embedding. GraphAE consists of graph encoder and graph decoder. In graph encoder, target node aggregates the attributes diffused from nodes in its local subgraph to generate its own representation, while in graph decoder, each node diffuses its representation to nodes in its local subgraph to help reconstruct their attribute information. Our proposed framework generates a node's representation by capturing both the structural information and attribute information manifested in its attributed local subgraph, having high capacity to learn good node representation for attributed network.

To evaluate the performance of proposed GraphAE framework, we conduct extensive experiments at two downstream tasks, i.e., node classification and link prediction. Experimental results on real-world datasets demonstrate that our proposed method outperforms the state-of-the-art network embedding approaches at both tasks.

2 RELATED WORK

Our proposed framework works in an encoder-decoder manner to learn better embeddings for attributed networks. To simultaneously capture the network structures and node attributes information manifested in local attributed subgraph, graph convolutional network is adopted in both encoder layer and decoder layer. In this section, we provide a brief introduction of related works in network embedding and graph convolutional network.

2.1 Network Embedding

Network embedding technology, which aims to learn low-dimensional embedding for nodes in network, actually evolves from the dimension reduction algorithm [9]. Some early works first leverage feature similarity to build an affinity graph, and then treat eigenvectors as network representations, such as LLE [35] and Isomap [37]. Recently, more network embedding methods leveraging the structural proximity or structural similarity among nodes have been proposed. Structural proximity based methods try to preserve different orders of proximities among nodes when learning node embeddings, varying from first-order proximity [28], second order proximity [36] to high order proximity [4, 29, 41]. Structure similarity based approaches [11, 17, 34] take into account structural roles of nodes, restricting nodes with similar structural roles to possess similar representations. Moreover, some deep models [5, 40] have been proposed to account for more complex structural properties.

However, besides structural properties, nodes in real world networks are usually associated with rich labels and attributes, which facilitates the problem of attributed network embedding [25, 26, 33, 42, 43]. Some approaches [19, 20, 38] simply take the label information into consideration, while others utilize more detailed attributes information. TADW [42] proposes to obtain node embeddings by decomposing the adjacency matrix, with the attribute matrix being fixed as a factor. DANE [12] leverages two separated auto-encoders to learn structural representations and attributed representations of nodes respectively and concatenates these two as final representations with consistent and complementary regularization in hidden layer.

Different from the above-mentioned algorithms that learn embeddings on homogeneous networks, some works [6, 21] also investigate network embedding on heterogeneous networks that have different types of nodes and links.

2.2 Graph Convolutional Network

Graph convolutional neural networks (GCNN) generalize CNN on non-Euclidean domains [2] have shown great success in variant of tasks. Existing GCNNs can be roughly categorized into two kinds, i.e., spectral GCNNs [3, 10] and spatial GCNNs [2]. Spectral GCNNs define the convolution on spectral domain, which first transform the signal into spectral domain and apply filters on it [3]. Spatial GCNNs view graph convolution as "patch operator", which construct a new feature vector using its neighborhood's information. GCN introduced by Kipf et al. [24] uses $k = 1$ order Chebyshev polynomials to approximate the filter in GCNN [3]. Under these circumstances, convolution operator equals to the weighted sum of neighboring nodes' features and weights are defined by normalized edge weights. As the weights in GCN only determined by network structure, velickovic et al. [39] propose Graph Attention Network(GAT) to learn the weights by structure masked self-attention.

The majority of these methods do not scale to large graph or are designed for whole graph, GraphSAGE [16] learns a function that generates embeddings by sampling and aggregating features from a node's local neighborhood which could learn inductive node embedding for large graph. Recently, some methods optimize the sampling strategy [7, 8] so that gcn can be better applied to large-scale networks.

3 METHOD

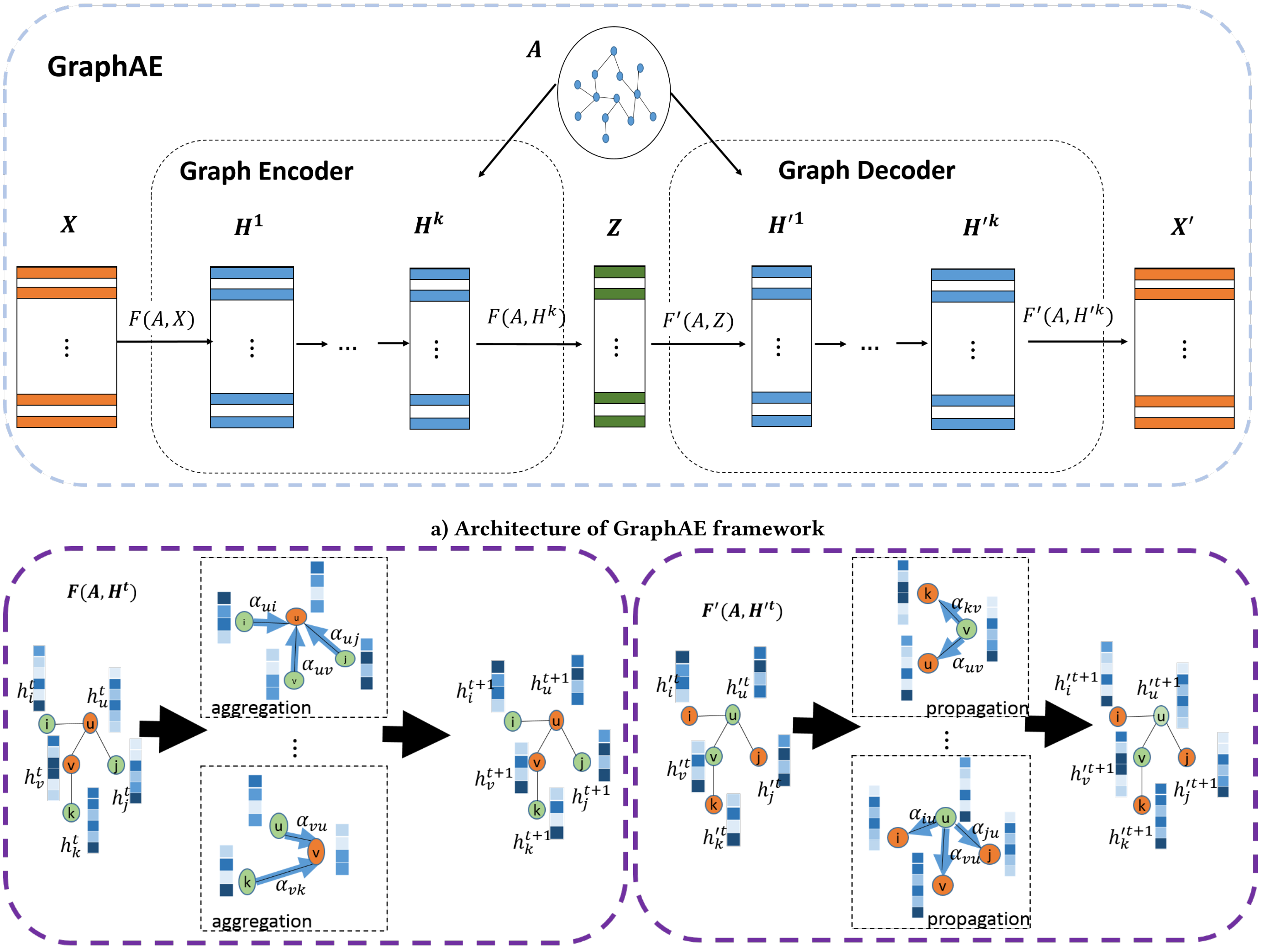
In this section, we first give definition of the notations used in this paper and then introduce the architecture and detailed implementation of our proposed graph auto-encoder framework.

3.1 Definition

We define a network as $G = \{V, E\}$, where $V = \{v_i\}_{i=1}^{n}$ denotes the node set with size n, and $E \subseteq V \times V$ denotes the edge set. The network is represented by an adjacency matrix A, where $A_{ij} = 1$ if $(v_i, v_j) \in E$ and $A_{ij} = 0$ otherwise. Attributes of nodes in the network are represented by an attribute matrix $X \in \mathbb{R}^{n \times d}$, where d is the dimension of node attributes. X_i is the i-th row of X, and represents the attribute vector of node v_i. Attributed network embedding aims to learn low-dimensional representations $Z \in \mathbb{R}^{n \times k}$ from adjacency matrix A and attribute matrix X, such that the learned representations can preserve both network structure and node attributes.

3.2 Graph Auto-Encoder Framework

In this paper, we propose a graph auto-encoder framework for attributed network embedding. The framework consists of two main parts, i.e., graph encoder and graph decoder. Graph encoder generates hidden representation Z with attribute matrix X serving as input, while graph decoder tries to reconstruct attribute matrix X from hidden representation Z. Both graph encoder and graph decoder charactirize the diffusion of attribute information over


GraphAE
A
Graph Encoder
Graph Decoder
X
H^1
H^k
Z
H'^1
H'^k
X'
F(A, X)
F(A, H^k)
F'(A, Z)
F'(A, H'^k)
F(A, H^t)
h_i^t, h_u^t, h_v^t, h_j^t, h_k^t
α_ui, α_uj, α_uv, α_vu, α_vk
aggregation
aggregation
h_i^{t+1}, h_u^{t+1}, h_v^{t+1}, h_j^{t+1}, h_k^{t+1}
F'(A, H'^t)
h_i'^t, h_u'^t, h_v'^t, h_j'^t, h_k'^t
α_kv, α_uv, α_iu, α_ju, α_vu
propagation
propagation
h_i'^{t+1}, h_u'^{t+1}, h_v'^{t+1}, h_j'^{t+1}, h_k'^{t+1}
i
u
v
j
k

a) Architecture of GraphAE framework

b) Single layer of Graph Encoder

c) Single layer of Graph Decoder

Figure 1: Visual illustration of the GraphAE framework. a) is the architecture of GraphAE. GraphAE uses graph decoder reconstruct the node attributes X from hidden representation Z which is generated by graph encoder with network structure A and node attributes X as input. H^t represents the output of t-th layer in graph encoder while H'^t is the output of t-th layer in graph decoder. b) shows the process of a single layer of graph encoder with nodes u and v as targets. c) shows the process of a single layer of graph decoder. Nodes u and v propagate their representations to help neighboring nodes reconstruct their attributes.

network A. The whole architecture of the framework is shown in Figure 1a).

A single layer of graph encoder and graph decoder are shown in Figure 1b) and Figure 1c) respectively. Graph encoder learns a sole embedding for the target node by aggregating the attribute information from nodes in its attributed local subgraph. In graph decoder, each node tries to propagate its representation to nodes in its local subgraph to help them reconstruct their attribute information. The network embedding learned by such encoder-decoder framework naturally captures both the structural information and the attribute information manifested in each node's attributed local subgraph, thus better serving applications on attributed networks.

3.2.1 Graph Encoder. In this section, we introduce the detailed implementation of graph encoder. Graph encoder consists of a stack of single encoder layers, each of which aggregates the attribute information from the neigboring nodes of a target node. By stacking multiple encoder layers, graph encoder is able to aggregate the attribute information from the multi-hop ego-network of the target node, which is taken as the target node's attributed local subgraph in this paper.

A single encode layer can be formalized as follows:

$$h_i^{t+1} = \sigma((\sum_{j \in N_i} \alpha_{ij} W h_j^t)), \tag{1}$$

where $h_i^t \in \mathbb{R}^f$ is the hidden representation of node i in the t-th layer and f denotes its dimension. N_i is the set of node i's neighbors including node i itself in our experiments. α_{ij} is the aggregation weight and measures how important node j is to node i, and σ represents the nonlinear activation function. The linear transformation parameterized by a weight matrix $W \in \mathbb{R}^{f' \times f}$ is applied on every node to extract effective high-level features from inputs. The key of designing a single encoder layer lies in the definition of aggregation weight α_{ij}, which is implemented via attention mechanism in this paper.

Attention based aggregation mechanism. We adopt shared attention mechanism to effectively measure the aggregation weight between two given nodes with node attributes serving as input. We use the same attention mechanism as in GAT [39]. This mechanism is parameterized by a weight vector $\vec{a} \in \mathbb{R}^{2f'}$($f'$ is the dimension of input vector), following with a nonlinearity activation. For each node i, we only compute α_{ij} for node $j \in \mathcal{N}_i$, where $\mathcal{N}_i$ is neighbors of node i [39]. The attention mechanism can be expressed as:

$$\alpha_{ij} = \frac{\exp(\sigma'(\vec{a}^{\mathrm{T}}[W\vec{h}_i || W\vec{h}_j]))}{\sum_{k \in N_i} \exp(\sigma'(\vec{a}^{\mathrm{T}}[W\vec{h}_i || W\vec{h}_k]))}, \tag{2}$$

where $\cdot^{\mathrm{T}}$ denotes matrix transposition and || represents concatenation operation. In our experiments, we adopt LeakyReLU (with negative input slope $\alpha = 0.2$) [39] as nonlinearity activation σ'. We also employ multi-head attention to stabilize the learning process of self-attention and capture multiple types of relationships between nodes. In our experiments, we concatenate the different representations learned by different heads in hidden layer, and average them on the final layer of the graph encoder.

Attention based aggregation mechanism can capture both the structural proximity and the attribute proximity between pairs of nodes, allowing better modeling for the attributed local subgraph of a target node.

3.2.2 Graph Decoder. Similar to graph encoder, graph decoder consists of multiple single decoder layers. The decoder layer in conventional auto-encoder framework decompresses the hidden representations and makes the output closely match the input data, which regularizes the hidden representation containing rich information about raw input. In graph encoder, a node obtains its represetation by aggregating the attribute information from its local subgraph, which makes it neccessary to propagate its representation to nodes in its local subgraph to help reconstruct their attribute information. In fact, all nodes aggregating from their neighbors are identical to all nodes propagating to their neighbors from the global view of the whole network, which allows graph decoder to adopt the same architecture as graph encoder. Taking figure 1c) as example, node v propagates its representation to neighbors u, k with attention weight α_{kv} and α_{uv}. But from the view of node k, this operator is equal to aggregate representation from node v with attention weight α_{kv}. This motivates us to adopt graph attention layer to build graph decode layer $F'(A, X)$. In our experiments we stack the same number of graph attention layer as graph encoder, and the hidden units are symmetric to graph encoder. The overall architecture for graph decoder is shown right side of figure 1a) and a single layer of graph decoder is shown in Figure 1c).

3.3 Loss function

We directly measure the Euclidean distance between the reconstructed attribute matrix X' (Output of graph decoder) and the original input attributes X as loss function, which is formalized as follows:

$$L_c = ||X - X'||_F \tag{3}$$

We add L2 regularization on parameters and the overall loss function of our model can be formalized as follows:

$$L = L_c + \lambda L_r \tag{4}$$

where λ is the hyper-parameter used to control the weight of L2 regularization of parameters. All the parameters in our framework are trained by minimize L using gradient descent.

Algorithm 1 mini-batch version of GraphAE

```
Require: adjacency matrix $A$, node attributes $\{X_v, \forall v \in B\}$, neighbor sampling functions, $\mathcal{N}_k, \forall k \in 1, ..., K$, attention mechanism, $attention_k, \forall k \in 1, ..., K$
Ensure: output representation of $z_v$ and reconstructed attributes $r_v$ for all $v \in \mathcal{B}$
$B^K \leftarrow B$
for $k = K...1$ do
    $B^{k-1} \leftarrow B^k$
    for $u \in B^k$ do
        $B^{k-1} \leftarrow B^{k-1} \cup N_k(u)$
    end for
end for
$h_u^0 \leftarrow x_u, \forall u \in B^0$
for $k = 1...K$ do
    for $u \in B^k$ do
        $a^k_{N_k(u)} \leftarrow attention_k(W^k h_v^{k-1}, \forall v \in N_k(u))$
        $h_u^k = \sigma(W^k(\sum_{j \in N_k(u)} \alpha_{uj} h_j^{t-1}))$
        $z_u \leftarrow h_u^k$
    end for
end for
$h_u'^0 \leftarrow z_u, \forall u \in B^K$
for $k = 1...K$ do
    for $u \in B^k$ do
        $a^k_{N_k(u)} \leftarrow attention_k(W'^k h_v'^{k-1}, \forall v \in N_k(u))$
        $h_u'^k = \sigma(W'^k(\sum_{j \in N_k(u)} \alpha_{uj} h_j'^{t-1}))$
        $r_u \leftarrow h_u^k$
    end for
end for
return $r_u, z_u, \forall u \in B$
```

3.4 Scalable to large graph

Graph attention layer takes the whole graph (adjacency matrix A and node attributes matrix X) as input, and both memory and time cost for computing are related to the size of graph N. As a result, this layer can not directly be applied to large-scale networks. Mini-batch is used to avoid inputting the whole graph, but it is still computing costly as aggregation operator depends on lots of neighbors. Specifically, "K-order neighbors" are attached to the input in order to learn the embedding of the given node when we stack k layers of graph attention networks as graph encoder. Because in real-world network, neighbors will soon cover the whole network with increase of order k. In order to reduce the complexity of the computing (reduce the number of neighbors), we randomly sample a fix number of neighbors for each node to update its representation in each epoch. All nodes needed in computing are sampled at first, the number of which is unrelated to the size of graph n, thus our model is scalable to large graph. The process is shown in Algorithm 1.

Line 1-7 is the process of sampling all nodes that used in computing. B^{k-1} is the set of nodes that are used in layer k. We adapt the same sample strategy as GraphSAGE [16] and $N_k(u)$ is a uniform sampling function at layer k, i.e, for each node randomly sample a number of its neighbor nodes by uniform distribution. Furthermore, we sample nodes with replacement in cases where the sample size is larger than the nodes' degree. Line 8-13 is the process of computing representations for all nodes giving input set B, while line 15-21 is the process of reconstructing node attributes from hidden representations.

Different from the algorithm that mentioned above, the attention weight and the aggregation operator are only applied on the subset of its neighbors that appear in set B^{k-1}. From the Algorithm 1, "k-order proximity" of target nodes are still kept and the number of nodes that used to learn the representation for target nodes is reduced to an acceptable scale.

4 EXPERIMENTS

Table 1: Statistics of Datasets

Datasets	Nodes	Edges	Classes	Features
Cora	2,708	5,429	7	1,433
Citeseer	3,327	4,732	6	3,703
Wiki	2405	17981	1 7	4973
Pubmed	19,717	44,338	3	500

We evaluate our proposed model on real-world datasets at two commonly adopted tasks, i.e., link prediction and node classification. Link prediction checks the ability of nodes' representations to reconstruct network structure and predict future links, while node classification verifies whether node embeddings learned by our model are effective for downstream tasks. Moreover, we also provide detailed analysis for the performance of our proposed model.

4.1 Dataset

We conduct experiments on four real-world datasets, i.e., Cora [12, 27, 42], Citeseer [12, 27, 42], Wiki [12, 27, 42] and Pubmed [12]. Cora, Citeseer and Pubmed are three citation networks where nodes are articles and edges indicate citations between articles. In these three datasets, citation relationships are viewed as undirected edges for simplicity. Attributes associated with nodes are extracted from the title and the abstract of each article and are represented as sparse bag-of-word vectors. Stop words and low-frequency words are removed in preprocessing. Wiki dataset is a web page network, where nodes represent web pages and edges are hyper links among web pages. Text information on the web pages is processed in a similar way as in the other three datasets to extract the attributes. Each node in the four datasets only has one label, indicating which class the node belongs to. Statistics of these datasets, including number of nodes (Nodes), number of edges (Edges), number of categories (Classes) and the dimension of attributes (Features), are summarized in Table 1.

4.2 Experiments Set-up

4.2.1 Model set-up. In experiments, the number of layers in graph encoder is set to be 2. The dimensions of hidden representations in two encoder layers are set to be 128 and 64 respectively. The number of attention heads is set to be $K = 8$ for the first encode layer, and $K = 1$ for the second layer. We stack two decode layers for graph decoder. The first decode layer has 128 hidden units with $K = 8$ attention heads. The dimension of the output of second decoder layer is set to be the dimension of input attributes and the second decoder layer has $K = 1$ attention head. We also add dropout (dropout probability= 0.5) and L2 regularization ($\lambda = 5e{-}4$) to prevent overfitting, and train our models using Adam with a learning rate of 0.001. Weights are all initialized by glorot [13] that brings substantially faster convergence.

All these models are implemented in tensorflow [1], a widely used deep learning tool. We optimize all the hyper-parameters using a validation set.

4.2.2 Baselines. We compare our model with the following baselines at both link prediction and node classification tasks. All the baselines fall into three categories, namely "Attributes-only", "Structure-only" and "Attributes+Structure". Models in "Attributes-only" group leverage node attributes information only to extract node representations, from which we select SVD and auto-encoder as our baselines. "Strucuture-only" models consider structure information only, i.e., preserving structural proximity in embedding space, while ignoring attribute information. In this group we choose Deepwalk and SDNE as our baselines. Methods in "Attribute+Structure" group capture both nodes attributes and structure proximity simultaneously and we consider several recent state-of-the-art algorithms as our baselines. A detailed description of our baselines is illustrated as follows:

- **Singular Value Decomposition (SVD)**: SVD [14] is a linear model that can extract node representations by decomposing the node attribute matrix. The largest 200 eigen values are kept when performing SVD.
- **Auto-encoder (AE)**: AE [18] is the conventional auto-encoder model with nodes attributes as input only. The number of hidden units is set the same as the GraphAE.
- **DeepWalk (DW)**: DW [32] learns embedding using structural information only. DeepWalk learns the node embedding from a collection of random walks using skip-gram with hierarchical softmax. As for the parameters, the number of random walks is 10, the number of vertex per walk $\gamma = 80$, window size $t = 10$ and embedding dimension $k = 128$.

Table 2: Result of link prediction

Categories		Cora		Wiki		Citeseer		Pubmed	
	method	AUC	AP	AUC	AP	AUC	AP	AUC	AP
Attributes-only	SVD	79.1	82.46	83.7	87.98	85.82	88.76	85.98	87.6
	AE	79.14	79.40	77.26	82.08	81.39	82.08	85.15	84.82
Structure-only	DW	80.53	82.79	81.27	82.39	73.22	76.21	76.88	74.73
	SDNE	77.87	81.95	81.68	82.66	74.46	78.91	77.91	75.29
Attributes+Structure	DW+SVD	81.06	83.13	89.57	91.1	73.92	76.72	76.92	74.76
	TADW	93.01	93.95	92.19	93.1	94.51	95.67	94.71	95.01
	DANE	88.19	89.56	91.01	92.45	84.93	84.68	87.76	89.69
	STNE	89.9	88.77	88.87	88.75	93.51	94.61	92.77	91.65
	GAE	91.47	92.37	91.81	92.91	90.52	91.59	95.93	**96.25**
	VGAE	91.7	92.64	91.17	92.49	90.96	92.97	94.05	94.42
Our Model	GraphAE	**96.70**	**97.46**	**94.54**	**95.13**	**96.99**	**96.92**	**96.91**	96.10
	Improvement	3.96%	3.73%	2.55%	2.18%	2.62%	1.30%	1.02%	-0.15%

- **SDNE**: SDNE [40] is a deep model that capture both first-order and second-order proximity of nodes in embedding with only structure information being considered. The structure of hidden units in SDNE is set the same as GraphAE, and the hyper-parameters of α, β and ν are tuned by using grid search on the validation set.
- **DW+SVD**: DW+SVD concatenates representations leaned by DeepWalk and SVD.
- **TADW**: TADW [42] is an approach that utilizes both network structure and text information to learn embedding. We set the dimension of representations to be 160 and the coefficient of regularization term to be 0.2.
- **GAE/VGAE**: GAE and VGAE [23] are weakening variant of the model, as they replace attention based aggregation with edge based aggregation and remove graph decoder only aiming to reconstruct network structure. Hyper-parameters are set the same as in their paper. We train the model for a maximum of 200 iterations using Adam [22] with a learning rate of 0.01.
- **DANE**: DANE [12] use two independent auto-encoder to model attributes and structure information respectively with serveral regularizations in hidden representation. We use the same architecture and hyper-parameters of DANE as in [12].
- **STNE**: STNE [27] is a sequence translation model that translate attributes associate with nodes into their identities with structure information encoded in random walk path. For all the datasets, we generate 10 random walks that start at each node, and the length of the walks is set to 10. For Cora Citeseer and Wiki which are used in [27], we use the same architecture of model and hyper-parameters as in [27]. The neural networks have 9 layers for Pubmed with dropout probability $p = 0.2$.

4.3 Link Prediction

In this section, we evaluate the ability of learned embeddings to reconstruct network and predict future connections via link prediction. We generate the dataset as many other works do [15, 23, 40]. We split the edges in the network according to the ratio of 85%, 5% and 10% as positive instances for training, validation and testing, respectively. For test set, we add some negative samples by randomly sampling some unconnected node pairs and the ratio of positive samples and negative samples is kept as 1:1. After having obtained the embeddings for each node, we get link probability by calculating inner product of the embeddings on test data. We adopt the area under the ROC-curve (AUC) and average precision from prediction scores (AP) implemented by sklearn [31] as the evaluation metrics. We report the AUC and AP measures of baseline models and ours in Table 1, we bold the best results and underline the next best results. We summarize the following observations and analyses:

- "Attributed-only", especially SVD, achieves comparable or better results than "Structure-only" methods in all datasets. The reason is that all these datasets are assortative networks, nodes with similar attributes are more likely to connect with each other.
- We also observe that "Attributes+Structure" methods that incorporate both node attributes and network structure information improves the link prediction performance. TADW, GAE and VGAE get better results than other "Attributes+Structure" methods. The superiority may result from the fact that they treat reconstruct adjacency matrix as their objects and directly optimize them, which is highly related to the link prediction task.
- Our GraphAE model achieves relatively significant improvements in AUC and AP over the baselines in all the four datasets, as shown in the Table 2. Our model incorporates node attributes and network structure in a way and captures high-order proximity in graph encoder, thus achieving better result.

4.4 Node Classification

In this section, we conduct experiments on node classification to demonstrate the effectiveness of the learned embeddings for downstream tasks. All nodes attributes and edges are observed when learning embeddings. After learning the embedding for each node, a logistic regression classifier (LR) with L2 regularization are used

Table 3: Node classification result of Cora

	% Labeled Nodes									
	10%		**20%**		**30%**		**40%**		**50%**	
method	Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1
SVD	47.44	34.84	60.86	54.60	66.70	62.43	68.54	65.20	69.55	66.64
AE	68.00	64.17	71.38	68.11	71.30	68.22	72.36	69.68	73.26	69.66
DW	76.53	75.26	78.58	77.03	79.86	77.95	80.5	78.26	81.33	79.19
SDNE	76.84	75.13	79.64	77.63	79.98	78.26	81.31	78.73	82.122	79.72
DW+SVD	76.653	74.91	80.37	79.08	81.63	80.30	83.36	81.92	83.97	82.44
TADW	77.92	75.34	82.04	80.657	83.5	82.3	83.89	82.63	84.58	82.91
DANE	78.01	76.09	80.39	78.76	81.86	80.05	82.15	80.5	82.5	80.58
STNE	81.83	80.35	84.31	82.34	84.75	82.84	86.27	84.59	86.55	84.53
GAE	80.39	79.3	81.31	80.32	82.22	81.84	82.27	80.8	82.35	81.39
VGAE	83.02	81.23	83.29	82.36	84.28	83.96	84.61	84.35	85.96	84.56
GraphAE	**84.7**	**83.73**	**85.74**	**85.21**	**86.5**	**85.77**	**86.77**	**85.84**	**87.37**	**86.29**
Improvement	2.02%	3.07%	1.69%	3.46%	2.06%	2.15%	0.58%	1.47%	0.94%	2.04%

Table 4: Node classification result of Wiki

	% Labeled Nodes									
	10%		**20%**		**30%**		**40%**		**50%**	
method	Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1
SVD	65.32	49.60	72.45	58.08	75.59	61.47	76.51	63.14	77.61	65.92
AE	53.99	41.28	61.69	52.90	65.00	54.73	66.39	57.97	68.08	59.25
DW	57.95	42.28	62.71	48.74	65.34	52.77	65.97	54.64	67.15	55.70
SDNE	53.99	43.479	56.79	46.504	59.84	50.31	60.84	50.757	61.981	52.03
DW+SVD	64.24	48.39	71.31	56.20	75.14	60.01	75.56	61.34	76.97	64.01
TADW	67.64	50.10	73.04	58.14	76.17	63.01	78.03	64.66	79.00	66.92
DANE	72.98	59.97	75.00	63.58	77.26	64.14	77.52	66.52	78.30	67.84
STNE	71.31	56.10	74.74	64.04	77.73	**70.62**	79.21	70.04	80.05	69.47
GAE	68.82	49.75	70.79	53.28	70.600	54.24	72.14	57.02	72.48	55.95
VGAE	65.54	46.77	66.73	49.30	68.82	51.61	69.43	50.64	68.99	51.49
GraphAE	74.82	**63.83**	**77.65**	**67.81**	**78.44**	69.19	**79.62**	**70.67**	**80.38**	**72.34**
Improvement	2.52%	6.43%	3.53%	5.88%	0.91%	-2.02%	0.51%	0.90%	0.41%	4.13%

Table 5: Node classification result of Citeseer

	% Labeled Nodes									
	10%		**20%**		**30%**		**40%**		**50%**	
method	Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1
SVD	61.14	53.19	66.40	58.56	66.95	59.73	68.05	61.31	69.39	62.83
AE	68.03	59.02	68.99	59.69	69.30	60.10	69.89	60.43	70.22	61.01
DW	50.15	46.18	54.01	49.51	56.24	51.03	56.32	51.25	56.83	51.90
SDNE	53.79	49.33	55.39	50.352	57.03	51.97	57.41	52.35	58.68	53.6
DW+SVD	52.84	48.77	57.99	53.51	61.37	56.25	64.13	58.98	67.29	62.21
TADW	67.02	60.89	70.08	64.84	71.76	66.78	72.19	67.13	72.92	68.22
DANE	63.64	59.83	67.24	61.83	68.69	64.96	72.21	68.30	72.30	67.78
STNE	66.37	61.67	71.45	66.72	73.20	68.84	73.96	70.18	74.58	70.84
GAE	60.52	52.94	60.12	53.13	60.59	52.27	61.11	52.87	62.00	53.38
VGAE	64.91	58.24	66.44	59.98	67.06	60.65	67.71	60.51	68.25	62.18
GraphAE	**72.61**	**67.76**	**74.18**	**69.18**	**74.33**	**70.06**	**75.51**	**71.47**	**76.04**	**71.76**
Improvement	8.34%	9.87%	3.82%	3.68%	1.54%	1.77%	2.09%	1.83%	1.95%	1.29%

Table 6: Node classification result of Pubmed

	% Labeled Nodes									
	10%		**20%**		**30%**		**40%**		**50%**	
method	Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1
SVD	64.93	50.21	74.41	68.98	78.05	75.47	80.36	79.20	81.12	80.31
AE	79.34	79.57	80.56	80.77	80.85	81.00	81.22	81.35	81.54	81.67
DW	79.87	78.31	80.65	79.18	80.99	79.60	81.20	79.80	81.35	79.86
SDNE	80.17	79.31	81.56	79.85	81.34	79.65	81.94	80.1	82.12	80.18
DW+SVD	79.86	78.39	81.35	80.00	82.09	80.88	82.63	81.49	83.00	81.93
TADW	82.86	82.75	83.59	83.46	83.83	83.71	84.74	84.61	84.78	84.64
DANE	84.28	**83.97**	85.14	84.87	85.44	85.17	85.91	85.54	**86.56**	86.35
STNE	83.16	82.32	83.73	82.94	84.23	83.43	84.50	83.64	84.87	84.05
GAE	83.91	83.27	83.96	83.36	84.38	83.77	84.34	83.72	84.33	83.72
VGAE	82.52	81.94	82.63	82.01	83.04	82.45	82.99	82.48	83.17	82.66
GraphAE	**84.71**	83.95	**85.49**	**85.23**	**85.83**	**85.58**	**86.36**	**86.12**	86.46	**86.37**
Improvement	0.51%	-0.02%	0.41%	0.42%	0.45%	0.48%	0.52%	0.67%	-0.11%	0.02%

to classify the nodes into different labels. We use LR package provided by sklearn [31] with default parameters. We random sample a certain number of nodes with labels as training data and the rest as test. We repeat the experiments 10 times and report the mean result. To conduct a comprehensive evaluation, we vary the percentage of labeled nodes in training from 10% to 50%. Furthermore, we employ Macro-F1 and Micro-F1 as the metrics to evaluate the classification result. All hyper-parameters used for learning embedding are set the same as previous link prediction experiment in section 4.3. The classification results are shown in Table 3, 4, 5, 6 respectively, and the best results are boldfaced while the next best are underlined. From these results, we have the following observations and analysis:

- "Structure-only" methods outperform "Attributes-only" methods in Cora, get comparable result in Pubmed, while perform worse on the other two datasets. Characteristics of datasets are the main reason to explain this phenomenon. Documents in Wiki network contribute more on the classification of nodes as the hyperlink relationship is loose. Web pages belonging to different categories still have a high probability to have hyperlinks. Documents in both Citeseer and Wiki have more words than Cora, so "Attributes-only" methods perform better. Cora network has high edge density and less words in documents thus "Structure-only" methods get better result in this dataset.
- Well-designed attributed network embedding methods (TADW, DANE, STNE) perform better than both "Attributes-only" methods and "Structure-only" methods, because these two kinds of information describe different aspects of the same node and provide complementary information. Unfortunately, simply concatenate these two kinds of information may not improve the performance, as "SVD+DW" gets worse node classification result than SVD in Citeseer and Wiki dataset. It demonstrates that simple concatenation is not sufficient to capture the interaction between these two types of information. GAE and VGAE get poor results in Wiki, which can be explained by the design of these two models. Wiki network has higher edge density than other networks and the aim of GAE and VGAE is to reconstruct the observed edges, thus may overfitting the observed edges and bringing in much noises.
- Our model outperforms all compared baselines in Cora, Citeseer and Pubmed. In Wiki network, our model achieves comparable result with STNE but still outperforms other baselines. Our model captures the attribute information and structure information in a unified way and adopts attention mechanism to enhance proximity modeling, thus better utilizing these two complementary type of information.
- Our model GraphAE outperforms all compared baselines when fewer labeled nodes are available in training. As we can see in the tables 2,3,4,5,6 the result of almost all the baselines (except GAE and VGAE) drop quickly when fewer labeled nodes are used in training. The reason is that these baselines do not well use neighboring nodes to preserve proximities in embedding space. By leveraging our graph encoder, our model gets smooth representations of nodes with their neighbors', thus obtaining better result especially when lacking of label information. Although edges in Wiki are not reliable, our model still get better result in Wiki data. This is because our model better leverage node attributes—learn more accurate aggregation weights through attention mechanism and preserve attributes proximity with graph decoder.

Table 7: Link prediction results of GraphAE and its variants

method	Datasets			
	Cora	**Wiki**	**Citeseer**	**Pubmed**
GraphAE	**96.70**	**94.54**	**96.99**	**96.91**
-decoder	94.44	93.66	93.63	95.33
-attention	87.08	93.88	87.47	93.56

4.5 Analysis of our model

To comprehensively analyze the performance of our proposed model, we provide detailed illustration on the difference and relation between our model and some other auto-encoder frameworks.

Table 8: Node classification results of GraphAE and its variants

method	Datasets			
	Cora	**Wiki**	**Citeseer**	**Pubmed**
GraphAE	**83.73**	**63.83**	**67.76**	**83.95**
-decoder	84.04	48.9	62.05	82.03
-attention	78.22	63.16	65.08	82.84

Moreover, the effectiveness of different components of our model is also analyzed in this section.

4.5.1 Relation to some baselines. AE, SDNE, GAE, DANE are four auto-encoder based frameworks used for network embedding. Thus we analyse their relationships with GraphAE in this section. Autoencoder only leverages the attributes information, and SDNE uses only structure information when learning embeddings. In contrast, GraphAE model network structure and node attributes simultaneously. GraphAE outperforms them in both link prediction and node classification tasks, which demonstrates that both nodes attributes and network structure are essential information for attributed network embedding.

GAE and VGAE [23] are special variants of our model, as they replace attention based aggregation mechanism with edge based aggregation mechanism and take network structure reconstruction as their objectives. We find that GraphAE gets better result than GAE and VGAE in all downstream tasks across all the datasets. It demonstrates that graph decoder and attention based aggregation help our model to better capture essential information of inputs which is useful for downstream tasks. DANE leverages two separated auto-encoders to learn structural representations and attribute representations of nodes respectively and concatenate these two as final representations with consistent and complementary regularization in hidden layer. Our model also outperforms DANE in both downstream tasks, which demonstrates that GraphAE better captures the context information manifested in both node attributes and network structure.

Recently, Hu et al, proposed ARGA [30] which improve GAE by enforcing the latent representation to match a prior distribution via an adversarial training scheme. This idea can also be added to our model to further enhance the embeddings.

4.5.2 Influence of attention mechanism and graph decoder. In this section we compare GraphAE with its two variants to verify the effectiveness of attention based aggregation weighting mechanism and graph decoder. For the first purpose, we replace the attention based aggregation weighting mechanism with edge based aggregation weighting mechanism, where $\alpha_{ij} = \frac{e_{ij}}{\sum_{j \in N_i} e_{ij}}$, e_{ij} is the edge weight. We mark this baseline as "-attention". To inspect the effectiveness of graph decoder we add another variant of our model which removes graph decoder and adopts structure reconstruction based objective function. We name this model "-decoder", and the loss function is formalized as follows:

$$L = - \sum_{<u,v> \in E} \log \left(\sigma \left(\mathbf{z}_u^\top \mathbf{z}_v \right) \right) - Q \cdot \mathbb{E}_{v_n \sim P_n(v)} \log \left(\sigma \left(-\mathbf{z}_u^\top \mathbf{z}_{v_n} \right) \right), \tag{5}$$

where Q is the number of negative samples and we set $Q = 10$. $P_n(v)$ is the distribution to sample nodes from, uniform distribution is used in our experiments. Other hyper parameters, e.g., number of hidden units, are set the same as GraphAE for fair comparison. Experiment setup is the same as section 3, and limited by space we only display the AUC result for link prediction and macro-f1 with 10% train label for node classification. As shown in the tables 7 and 8, GraphAE performs consistently better than "-attention" in both link prediction and node classification. This phenomenon can be explained by following two reasons. First, edges in the four datasets do not contain rich information and only indicate whether two nodes are connected with each other. Thus GraphAE-attention aggregates all neighboring nodes without distinguishing importance of nodes, which limits the capacity of model. Second, attention based mechanism provides a more flexible way to capture the proximities in node attributes and network structure by reweighting the importance of neighbors.

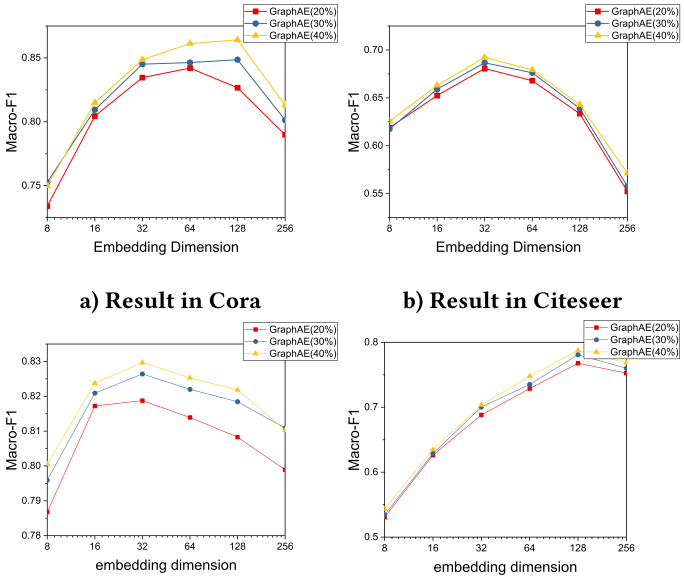
In all downstream tasks, we find that GraphAE outperforms "-decoder" especially when edges are not reliable, for example GraphAE outperforms "-decoder" with a huge margin in wiki. It mainly because structure reconstruction based loss overlaps with the function of graph encoder—model the proximity of nodes. It demonstrates that our graph decoder is better than structure reconstruction based loss which might over-emphasize local proximity.

4.5.3 Influence of dimension of hidden representation. Dimension of embedding is an important parameter, thus we examine how the different sizes of embedding affect the performance of downstream tasks. Due to space limitations, we only display the result of node classification on four datasets and we get similar result in link prediction task. We vary dimension of embedding from [8, 16, 32, 64, 128, 256], the number of units in first hidden layer is twice than the dimension of embedding. Other hyper-parameters are keep the same as mentioned in Section 4.

As shown in Figure2, the trend of the curves in two datasets is very similar-performance increase when dimension of embedding get larger at first and decrease when the size of embedding larger than a specific value. From the experimental results, we can find that our model is a bit sensitive to dimension. Fortunately, as the curves are "unimodal", it is easy to find a dimension gets good results in downstream tasks.

5 CONCLUSION

In this paper, we propose a novel graph auto-encoder framework, namely GraphAE, for attributed network embedding. GraphAE use a stack of graph convolutional networks to encode both network structure and node attributes into a sole low-dimensional representation for each node. In graph decoder, we use another stack of graph convolutional networks to reconstruct both network structure and node attributes from representations. Our model leverages the complex interaction between network structure and node attributes through feature diffusion, thus has high capacity to learn good node representations for attributed network. Experimental results show that our model consistently outperforms all the benchmark algorithms in two down-stream tasks. In the future, we will explore more kinds of powerful graph convolutional networks for the encoder layer of our framework.



a) Result in Cora

b) Result in Citeseer

c) Result in Pubmed

d) Result in Wiki

Figure 2: Macro-F1 result on different hidden dimension

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. Tensorflow: a system for large-scale machine learning.. In *OSDI*, Vol. 16. 265–283.

[2] Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. 2017. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine* 34, 4 (2017), 18–42.

[3] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann Lecun. 2014. Spectral networks and locally connected networks on graphs. In *International Conference on Learning Representations (ICLR2014), CBLS, April 2014*.

[4] Shaosheng Cao, Wei Lu, and Qiongkai Xu. 2015. GraRep: Learning Graph Representations with Global Structural Information. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*. ACM, 891–900.

[5] Shaosheng Cao, Wei Lu, and Qiongkai Xu. 2016. Deep Neural Networks for Learning Graph Representations.. In *AAAI*. 1145–1152.

[6] Shiyu Chang, Wei Han, Jiliang Tang, Guo-Jun Qi, Charu C Aggarwal, and Thomas S Huang. 2015. Heterogeneous network embedding via deep architectures. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 119–128.

[7] Jie Chen, Tengfei Ma, and Cao Xiao. 2018. FastGCN: Fast Learning with Graph Convolutional Networks via Importance Sampling. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. https://openreview.net/forum?id=rytstxWAW

[8] Jianfei Chen, Jun Zhu, and Le Song. 2018. Stochastic Training of Graph Convolutional Networks with Variance Reduction. In *International Conference on Machine Learning*. 941–949.

[9] Peng Cui, Xiao Wang, Jian Pei, and Wenwu Zhu. 2018. A survey on network embedding. *IEEE Transactions on Knowledge and Data Engineering* (2018).

[10] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*. 3844–3852.

[11] Claire Donnat, Marinka Zitnik, David Hallac, and Jure Leskovec. 2018. Learning Structural Node Embeddings via Diffusion Wavelets. In *International ACM Conference on Knowledge Discovery and Data Mining (KDD)*, Vol. 24.

[12] Hongchang Gao and Heng Huang. 2018. Deep Attributed Network Embedding.. In *IJCAI*. 3364–3370.

[13] Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. 249–256.

[14] Gene H Golub and Christian Reinsch. 1970. Singular value decomposition and least squares solutions. *Numerische mathematik* 14, 5 (1970), 403–420.

[15] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 855–864.

[16] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*. 1024–1034.

[17] Keith Henderson, Brian Gallagher, Tina Eliassi-Rad, Hanghang Tong, Sugato Basu, Leman Akoglu, Danai Koutra, Christos Faloutsos, and Lei Li. 2012. RolX: Structural Role Extraction & Mining in Large Graphs. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 1231–1239.

[18] Geoffrey E Hinton and Ruslan R Salakhutdinov. 2006. Reducing the dimensionality of data with neural networks. *science* 313, 5786 (2006), 504–507.

[19] Xiao Huang, Jundong Li, and Xia Hu. 2017. Accelerated attributed network embedding. In *Proceedings of the 2017 SIAM International Conference on Data Mining*. SIAM, 633–641.

[20] Xiao Huang, Jundong Li, and Xia Hu. 2017. Label informed attributed network embedding. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*. ACM, 731–739.

[21] Zhipeng Huang and Nikos Mamoulis. 2017. Heterogeneous information network embedding for meta path based proximity. *arXiv preprint arXiv:1701.05291* (2017).

[22] Diederik P Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. (2015).

[23] Thomas N Kipf and Max Welling. 2016. Variational Graph Auto-Encoders. *NIPS Workshop on Bayesian Deep Learning* (2016).

[24] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *International Conference on Learning Representations (ICLR)*.

[25] Tuan MV Le and Hady W Lauw. 2014. Probabilistic latent document network embedding. In *2014 IEEE International Conference on Data Mining (ICDM)*. IEEE, 270–279.

[26] Lizi Liao, Xiangnan He, Hanwang Zhang, and Tat-Seng Chua. 2018. Attributed social network embedding. *IEEE Transactions on Knowledge and Data Engineering* (2018).

[27] Jie Liu, Zhicheng He, Lai Wei, and Yalou Huang. 2018. Content to node: Self-translation network embedding. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 1794–1802.

[28] Tong Man, Huawei Shen, Shenghua Liu, Xiaolong Jin, and Xueqi Cheng. 2016. Predict Anchor Links across Social Networks via an Embedding Approach.. In *IJCAI*, Vol. 16. 1823–1829.

[29] Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. 2016. Asymmetric transitivity preserving graph embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 1105–1114.

[30] Shirui Pan, Ruiqi Hu, Guodong Long, Jing Jiang, Lina Yao, and Chengqi Zhang. 2018. Adversarially regularized graph autoencoder for graph embedding. *arXiv preprint arXiv:1802.04407* (2018).

[31] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[32] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. DeepWalk: Online Learning of Social Representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 701–710.

[33] Guo-Jun Qi, Charu Aggarwal, Qi Tian, Heng Ji, and Thomas Huang. 2012. Exploring context and content links in social media: A latent space method. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 34, 5 (2012), 850–862.

[34] Leonardo FR Ribeiro, Pedro HP Saverese, and Daniel R Figueiredo. 2017. struc2vec: Learning node representations from structural identity. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 385–394.

[35] Sam T Roweis and Lawrence K Saul. 2000. Nonlinear dimensionality reduction by locally linear embedding. *science* 290, 5500 (2000), 2323–2326.

[36] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. LINE: Large-scale Information Network Embedding. In *Proceedings of the 24th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 1067–1077.

[37] Joshua B Tenenbaum, Vin De Silva, and John C Langford. 2000. A global geometric framework for nonlinear dimensionality reduction. *science* 290, 5500 (2000), 2319–2323.

[38] Cunchao Tu, Weicheng Zhang, Zhiyuan Liu, Maosong Sun, et al. 2016. Max-Margin DeepWalk: Discriminative Learning of Network Representation.. In *IJCAI*. 3889–3895.

[39] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. *International Conference on Learning Representations* (2018). https://openreview.net/forum?id=rJXMpikCZ accepted as poster.

[40] Daixin Wang, Peng Cui, and Wenwu Zhu. 2016. Structural deep network embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 1225–1234.

[41] Xiao Wang, Peng Cui, Jing Wang, Jian Pei, Wenwu Zhu, and Shiqiang Yang. 2017. Community Preserving Network Embedding.. In *AAAI*. 203–209.

[42] Cheng Yang, Zhiyuan Liu, Deli Zhao, Maosong Sun, and Edward Y Chang. 2015. Network representation learning with rich text information.. In *IJCAI*. 2111–2117.

[43] Shenghuo Zhu, Kai Yu, Yun Chi, and Yihong Gong. 2007. Combining content and link for classification using matrix factorization. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, 487–494.