Expert Systems With Applications 117 (2019) 300–311

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

# A Structure-Enriched Neural Network for network embedding

Lisheng Qiao[a], Hongke Zhao[a], Xiaohui Huang[b], Kai Li[a], Enhong Chen[a,*]

[a] Anhui Province Key Laboratory of Big Data Analysis and Application, University of Science and Technology of China, China
[b] School of Computer Science and Technology, University of Science and Technology of China, China

ARTICLE INFO

*Article history:*
Received 8 May 2018
Revised 23 August 2018
Accepted 18 September 2018
Available online 19 September 2018

*Keywords:*
Network embedding
Direction adjustment
Attention mechanism

ABSTRACT

Recent years have witnessed the importance of network embedding in many fields, as well as increased attention in academia. Although a number of algorithms have been proposed in this area, most existing models which only utilize the structure topology information of networks often suffer performance losses because of their insufficiency with regard to selecting structure similar patterns, handling noise data, and/or capturing non-linear or high-order structure information. To address these challenges, in this paper, we present a novel Structure-Enriched Neural Network (SENN) for network embedding. Specifically, SENN can not only capture the complex structure similar patterns observed in networks by introducing direction adjustment parameters of the transition probability, but also introduce a stacked denoise autoencoder to perform the dimension reduction for each order matrix independently. Therefore, SENN can preserve more useful structure information and make the embeddings more robust. Moreover, SENN can effectively integrate the multi-order structure information by the combining layer with attention mechanism. Finally, to compare with other state-of-the-art methods, we conduct extensive experiments with both synthetic and real-world datasets on various tasks (e.g.,node classification, visualization). The experimental results clearly demonstrate the effectiveness of our proposed model for network embedding.

## 1. Introduction

Network embedding is an important task of learning low-dimensional representations in many fields, e.g., vectors of nodes in networks, to capture and preserve the network structure (Wang, Cui, & Zhu, 2016). Representing networks in low-dimensional space could widely benefit some network-oriented application and research, e.g., influence maximization (Tang, Sun, Wang, & Yang, 2009), node classification (Bhagat, Cormode, & Muthukrishnan, 2011), community discovery (Parthasarathy, Ruan, & Satuluri, 2011), and recommendation (Chorowski, Bahdanau, Cho, & Bengio, 2014). Due to the importance and fundamental nature of network embedding, increasing researchers have paid attention to this research problem and participated in extensive efforts in recent years.

Among the most existing methods of network embedding which only utilize the structure topology information, those mapping a network to a low-dimensional vector space are the most effective and widely-used ones, such as DeepWalk (Perozzi, Al-Rfou, & Skiena, 2014), node2vec (Grover & Leskovec, 2016), GraRep (Cao, Lu, & Xu, 2015), and DNGR (Cao, Lu, & Xu, 2016). In particular, DeepWalk combines the random walk and skip-gram model (Mikolov, Sutskever, Chen, Corrado, & Dean, 2013) to learn low-dimensional network embeddings. This method mainly preserves the second-order proximity, and it is empirically effective in multi-label classification tasks. The node2vec traverses the neighborhood of one node in a network, and can efficiently explore diverse neighborhoods by introducing direction control parameters to a rand walk procedure. To exploit the high-order structure information, GraRep linearly maps different-order-structure information into different low-dimensional vector spaces. This method can obtain the final embedding of a node by briefly concatenating different order embeddings of network. Furthermore, DNGR proposes a random surfing model to directly capture the network structure information, and it adopts deep learning technique to map the multi-order combined structure information to a low-dimensional space. Thus, the high order proximity and non-linear structure information can be captured. However, these existing models usually handle the network embedding from one view of diversifying structure similar patterns, exploiting high-order proximity, or capturing non-linear structure information rather than a holistic manner. Moreover, some of them may also suffer performance losses because of their insufficiency in selecting neighborhood patterns or handling noise data. Indeed, it is necessary to utilize the full ef-

* Corresponding author.
*E-mail addresses:* lsqiaoa@mail.ustc.edu.cn (L. Qiao), zhhk@mail.ustc.edu.cn (H. Zhao), huangxia@mail.ustc.edu.cn (X. Huang), marvin77@mail.ustc.edu.cn (K. Li), cheneh@ustc.edu.cn (E. Chen).

https://doi.org/10.1016/j.eswa.2018.09.040
0957-4174/© 2018 Elsevier Ltd. All rights reserved.

**Table 1**
Symbols.

| Notations | Descriptions |
|---|---|
| $d$ | the dimension of embedding |
| $N$ | the number of nodes in network $G$ |
| $K$ | the total number of orders |
| $T$ | the number of categories in the classification task |
| $\mathcal{V} = \{v_i\}_{i=1}^{n}$ | the input network node data |
| $S = \{S_1, \cdots, S_n\}$ | the adjacency matrix for the network |
| $U^k$ | the $k$-order adjusted transfer probability matrix |
| $M^k$ | the $k$-order adjusted optimization matrix |
| $X^k$ | the positive $k$-order adjusted optimization matrix |
| $Y^k$ | the $k$-order latent network embedding matrix |
| $\boldsymbol{W}_e$, $\boldsymbol{W}_d$ | the weight parameters of the denoise autoencoder |
| $\boldsymbol{b}_e$, $\boldsymbol{b}_d$ | the biased parameters of the denoise autoencoder |
| $\theta = \{\boldsymbol{W}_e, \boldsymbol{W}_d, \boldsymbol{b}_e, \boldsymbol{b}_d\}$ | the unified parameter of the denoise autoencoder |

fective features of different views in learning network embedding (Airoldi, Bai, & Carley, 2011).

Therefore, in this paper, we design a novel Structure-Enriched Neural Network (SENN) for network embedding task. Specifically, SENN captures the different adjusted $k$-order relation information of a network from a probabilistic perspective, by directly manipulating different order transition matrices defined over the network without involving the slow and complex sampling process. Moreover, SENN adopts a stacked denoise autoencoder to perform the dimension reduction for each order matrix independently. In this manner, SENN can preserve more useful non-linear structure information and work robustly for noise data. Moreover, SENN can combine the multi-order structure information by the combining layer effectively with attention mechanism. In experiments, to compare with other state-of-the-art methods, we conduct extensive tests with synthetic and real-world datasets from various aspects, i.e., a multi-label classification task, adjusted case study, visualization and parameter sensitivity. The experimental results clearly demonstrate the effectiveness of our proposed model for network embedding learning.

We believe that this is an explorative work for conducting network embedding with rich multi-order structure information from a holistic perspective, e.g., introducing the direction adjustment parameters of transition probability into different order transition matrices, and using the stacked denoise autoencoder to perform the dimension reduction for different order matrices independently. Moreover, we also employ the approach with attention mechanism to combine the multi-order information.

The main body of this paper is organized as follows. Section 2 formalizes the research problem, and Section 3 introduces the related work of this study. Then, Section 4 elaborates the model we proposed. Section 5 evaluates the effectiveness of our model in four tasks. Finally, we conclude this study in Section 6.

## 2. Problem definition

In this section, we give the problem definition of network embedding in our study. Table 1 lists the main mathematical notations used in this paper. Specifically, we use lowercase letters (e.g., $x$ and $y$) to denote scalars, and boldface lowercase letters (e.g., $\boldsymbol{x}$ and $\boldsymbol{y}$) to denote vectors. We use calligraphic letters to represent sets (e.g.,$\mathcal{V}$ and $\mathcal{E}$ ). The notation $|\cdot|$ denotes the total number of samples of a given set. Let $G = (\mathcal{V}, \mathcal{E})$ denote a network, where $\mathcal{V} = \{v_1, \cdots, v_N\}$ represents $N$ nodes, and $\mathcal{E} = \{e_{i,j}\}_{i,j=1}^{N}$ represents the set of edges. Each $e_{i,j}$ is associated with a weight $S_{i,j} \geq 0$ (we only consider non-negative links in this paper). The network embedding aims to learn a low-dimensional vector $\boldsymbol{v}_{v_i} \in R^d \ \ (d \ll |\mathcal{V}|)$ for each node $v_i$ in the network, where the nodes close to each other in the network are assumed to be similar in the embedding

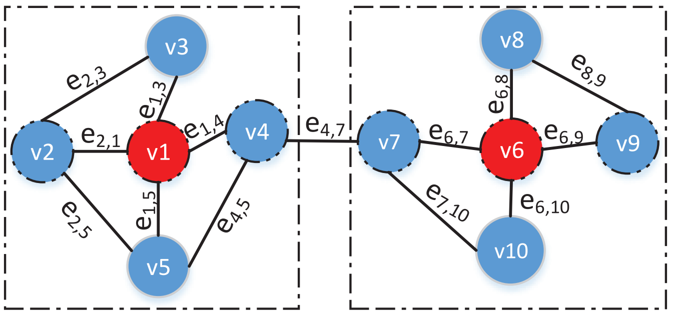


**Fig. 1.** The structure information of the network.

space. To this goal, from a holistic perspective, the main structure information that can be utilized are: (1) the similar patterns (e.g., the kinds of the nodes of different colors are shown in Fig. 1) of structure information, (2) the multi-order structure information (e.g., $v_4$ is the 2nd-order structure information of $v_2$ shown in Fig. 1), (3) the complex non-linear relation information. Then, the problem can be defined:

**Definition 1.** (Network embedding) Given a network $G = (\mathcal{V}, \mathcal{E})$, network embedding aims to represent each node $v \in \mathcal{V}$ in a low-dimensional space $R^d$, and the embeddings should explicitly preserve the following information: (1) the similar patterns of structure information, (2) the multi-order structure information, and (3) the non-linear structure information, such that nodes with similar network structure should be similarly represented.

## 3. Related work

In this section, we introduce the related work of this study.

### 3.1. Network embedding

The mainstream methods of learning network embedding that only use the structure topology information can be generally divided into three categories, i.e., matrix factorization based approaches, random walk based methods, and deep learning techniques.

**Matrix factorization based approaches**, Bullinaria and Levy (2012) utilize the global statistics of various matrix representations of the network, especially the Laplacian and adjacency matrices. From the perspective of linear algebra, these methods could be considered as dimension reduction techniques. There have been several linear (e.g., Laplacian Eigenmaps) and non-linear (e.g., IsoMap) methods proposed (Belkin & Niyogi, 2002; Tenenbaum, De Silva, & Langford, 2000). However, these methods have some weak points in computational and statistical performance. For the aspect of computational efficiency, eigen decomposition of a data matrix is computationally expensive if the quality of the solution cannot be guaranteed by approximations. In terms of characteristics, the objective of matrix factorization is generally to represent the nodes with high first-order proximity closely, which means it sometimes may not preserve the necessary high-order proximity information of a network. Furthermore, though these methods are optimized for objectives, they may also not adapt well to the diverse structure similar patterns observed in networks (Grover & Leskovec, 2016), because some prior information (such as homophily and structure equivalence) of the network structure may not be utilized well. In addition, it is difficult for these models to capture the highly non-linear relations in networks effectively (Wang et al., 2016).

**Random walk based methods**, whose ideas mainly come from NLP fields, often employ a slide window or introduce probabilistic sampling algorithms to capture the neighborhood nodes of one node. Specifically, DeepWalk (Perozzi et al., 2014), node2vec (Grover & Leskovec, 2016) and metapath2vec (Dong, Chawla, & Swami, 2017) are the representative ones in this category. In general, the first two methods transformed the network structure into several linear node sequences by using a truncated or adjusted random walk procedure, while the last one transformed the network structure into several probabilistic meta-paths by using meta-path-based random walks. Then they generated network embeddings by using a skip-gram (or heterogeneous skip-gram) model or stochastic gradient descent algorithm. These methods perform well on some specific tasks such as multi-label classification task and prediction task. However, despite some statistically significant sampling strategies having been adopted, they are insufficient in fully capturing the necessary information since they use separate local context windows.

**Deep learning techniques** perform well in capturing the non-linear relationships in complex network. In this category, two representative algorithms are SDNE (Wang et al., 2016) and DNGR (Cao et al., 2016). Specifically, SDNE was proposed as a semi-supervised deep model to jointly optimize the first-order and second-order proximity to preserve the network structure information. DNGR adopted a new sampling strategy, i.e., a random surfing model, to capture the structure information of a network directly, and then it obtained the low-dimensional embedding using a deep learning method. Although these models are effective (often more accurate than the above two types of methods) on certain tasks, there is still room on utilizing the information. For example, neither SDNE and DNGR are designed for adjusting the structure similar pattern information or utilizing the high-order information.

### *3.2. Deep neural network*

Deep neural networks have excellent discrimination and interpretation in low-dimensional and non-linear embedded spaces. In actual networks, such as semantic networks, there are not only linear relations, but also various complex and non-linear relations (Bayer & Riccardi, 2016). Therefore, the deep network architecture has been widely used in learning tasks (Bengio et al., 2009; Hinton & Salakhutdinov, 2006), and using the deep network architecture to establish discriminative embeddings (Bengio, Courville, & Vincent, 2013), has achieved good results. Meanwhile, the layer-by-layer training can help to find better local solutions to non-convex target objects (Bengio, Lamblin, Popovici, & Larochelle, 2007). Moreover, the improvement of computing hardware make it possible for rapid application of deep methods, such as the greedy layer-by-layer training strategy (Bengio et al., 2007; Dean et al., 2012).

Recently, there have been several deep learning works for network embedding learning. Inspired by related research results (Bengio et al., 2013; Cao et al., 2016; Gutmann & Hyvärinen, 2012; Tian, Gao, Cui, Chen, & Liu, 2014; Vincent, Larochelle, Bengio, & Manzagol, 2008; Wang et al., 2016), we take the denoise autoencoder as the basic component in our model, and combine them in a stacked manner. At the same time, we adopt the greedy layer-by-layer training (Vincent et al., 2008) to learn the embeddings, so that the network embeddings obtained can capture the complex non-linear relations of the network.

### *3.3. Attention mechanism*

attention mechanism is a processing method based on the human visual attention mechanism. The essence of the human visual attention mechanism is to focus on a specific area of the picture according to the “high resolution”, perceive the peripheral area of the image with “low resolution”, and then adjust the focusing mode continuously. Attention based models have been applied to various tasks, including image classification (Mnih, Heess, Graves et al., 2014), machine translation (Bahdanau, Cho, & Bengio, 2014) and speech recognition (Chorowski et al., 2014). Our work relates to the attention based model, which aims to infer the importance of different latent embeddings, and makes the learning algorithm focus on the informative parts. By taking advantage of attention mechanism, our model can focus on the information that is more important to the embedding.

## 4. Methodology

The overall architecture of the proposed Structure-Enriched Neural Network (SENN) is shown in Fig. 2, which is a novel framework defined for network-oriented tasks, such as node classification or node clustering. The architecture mainly consists of four parts. Specifically, SENN first introduces the $k$-order adjusted optimization matrix $M^k$ which is used to capture network structure information. In this subsection, we introduce the control parameters to adjust the transition probability and propose the concept of the adjusted transition probability matrix, which can select and adjust the structure similar patterns of the network flexibly. Because the motive of SENN is to use the necessary information to better accomplish the embedding, it is necessary to introduce adjustment parameters to select the necessary information to better complete the task flexibly. Secondly, to improve the sparsity and consistency of embeddings, SENN calculates the positive $k$-order adjusted optimization matrix $X^k$. After that, SENN utilizes the stacked denoise autoencoder to reduce the dimension for each $X^k$, and obtain the corresponding embedding $Y^k$, where $k \in \{1, 2, \dots, K\}$. The different $k$-order embeddings reflect the different aspects and levels of the structure characteristics of the networks due to the diversity of structure similar patterns (e.g., the network structure homogeneity, structure equivalence, size of different sub-graphs and other factors). Then, SENN combines the multi-order information by the combining layer and obtains the final output. Because SENN can select and adjust the structure similar patterns of the network flexibly, and can optimize the weights corresponding to different $k$-orders in the combining layer, our model can focus on the information that is more important to the embedding task. In accordance with the flow of SENN, we will detail each part in the following subsections.

### *4.1. Adjusted optimization matrix* $\mathrm{M}^k$

In this subsection, to capture the similar pattern of structure information, we first put forward the concept of the adjusted transfer probability matrix $U^k$ which can learn representations based on the notes' roles in networks and/or communities that they belong to by choosing an appropriate notion of a neighborhood, and infer the theoretical loss function $L_k$ of network embeddings based on $U^k$. Then, we optimize the loss function $L_k$, and get the associated adjusted optimization matrix $M^k$ which can capture the similar pattern of structure information guaranteed by the designed strategy.

#### *4.1.1. The adjusted transfer probability matrix* $\mathrm{U}^k$.

To selectively capture the similar pattern of structure information, it is intuitive to solve this problem from the perspective of the transfer probability between the nodes of network. Here, we formally propose our adjusted method to capture the similar pattern structure information.

First, to capture the transfer information from one node to another, we assume that the transition probability from node $v_i$ to node $v_j$ is proportional to $S_{ij}$, which is an element in the adjacency

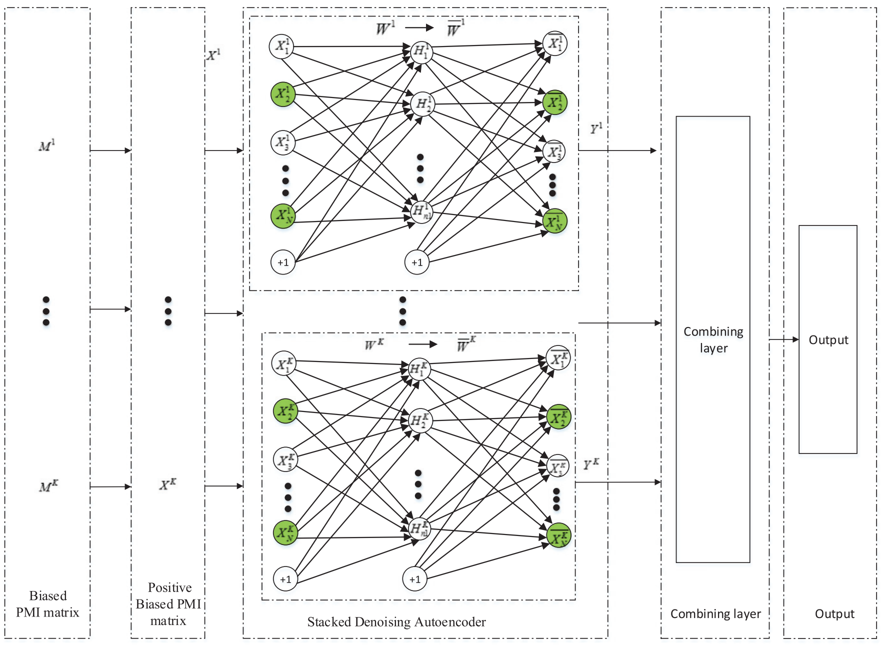

$M^1$
$M^K$
$X^1$
$X^K$
$W^1 \longrightarrow \overline{W}^1$
$W^K \longrightarrow \overline{W}^K$
$Y^1$
$Y^K$
+1
Biased PMI matrix
Positive Biased PMI matrix
Stacked Denoising Autoencoder
Combining layer
Combining layer
Output
Output

**Fig. 2.** The Structure-Enriched Neural Network (SENN).

matrix $S$. Then the (1-order) transition probability matrix can be defined as follows:

$$A = D^{-1}S,$$

where $A_{ij}$ is the transition probability from $v_i$ to $v_j$, $S$ is the adjacency matrix of the network, and $D$ is the degree matrix of the network, which is a diagonal matrix. The relationship between $D$ and $S$ is as follows:

$$D = \begin{cases} \sum_{h\in\mathcal{H}} S_{v_i v_h}, & \text{if } \mathrm{v_i} = \mathrm{v_j} \\ 0, & \text{if } v_i \neq v_j \end{cases},$$

where $\mathcal{H}$ represents the neighbor nodes of node $v_i$, and $v_j \in \mathcal{H}$ .

To get the needed structure similar patterns observed in the network selectively, we introduce two control parameters in our formalization to adjust the transfer probability between the nodes of the network. That is to say, we explore the diverse neighborhoods of one node by introducing direction control parameters to make the structure similar pattern reflected by the embeddings of nodes be the structural similar pattern we need. This idea is inspired by the adjusted walk in Grover and Leskovec (2016). Specifically, we design the direction control function (here we call it as the adjusted control function for simplicity) as follows:

$$\alpha_{pq}\left(v_i, v_j\right) = \begin{cases} \frac{1}{p} & \text{if } d_{v_i v_j} = 0 \\ 1 & \text{if } d_{v_i v_j} = 1 \\ \frac{1}{q} & \text{if } d_{v_i v_j} \geq 2 \end{cases},$$

where $d_{v_i v_j}$ is the distance between node $v_i$ and node $v_j$, and $p$ and $q$ are the hyper-parameters to control the likelihood of immediately transferring from one node to another.

To combine the transition probability information with the adjusted information, we make $\pi_{v_i,v_j} = \alpha_{pq}\left(v_i, v_j\right) \cdot A_{v_i v_j}$, and then:

$$p\left(v_j|v_i\right) = \begin{cases} \frac{\pi_{v_i,v_j}}{Z} & \text{if}\left(\mathrm{v_i}, \mathrm{v_j}\right) \in \mathcal{E} \\ 0 & \text{otherwise} \end{cases},$$

where $\mathcal{E}$ is the set of edges, and $Z$ is the normalization constant. We borrow the idea of the probability that any neuron $(i, j)$ is reasonable in Hang and Stemberg (2012), and give a penalty when the nodes with the needed structure similar pattern are mapped far away in the latent embedding space. Differently, the information we introduce is the structural equivalence related information, while the former used is multiplication.

Finally, we suppose that one node is $v_i$, another node is $v_j$, and the original transfer matrix is $A$. Let $B = \left[\alpha_{pq}\left(v_i, v_j\right)\right]$, where $i, j \in \{1, 2, \ldots N\}$ . Then, there is an adjusted transfer probability matrix $(B \cdot A)$ (which can be denoted as $U$) that is manipulated by the control function $\alpha_{pq}(v_i, v_j)$, where $\cdot$ denotes the pointwise multiplication. At the same time, we use $p_k(v_j|v_i)$ to represent the $k$-order transition probability from $v_i$ to $v_j$. Thus, we introduce the following $k$-order adjusted transition probability matrix as:

$$U^k = B \cdot A^k .$$

According to the method in Cao et al. (2015), we can find that $U^k_{v_i,v_j}$ is exactly equal to the adjusted transfer probability of the $k$-order transition from node $v_i$ to node $v_j$, that is,

$$p_k\left(v_j|v_i\right) = U^k_{v_i,v_j}, \quad (1)$$

where $U^k_{v_i,v_j}$ is the element of the row corresponding to $v_i$ and column corresponding to $v_j$ in matrix $U^k$.

### 4.1.2. *Loss function* $\mathrm{L_k}$ *based on* $\mathrm{U^k}$.

In this part, let us study the loss function of network embedding when the input is the adjusted transfer probability matrix under the unsupervised condition. Without loss of generality, we consider the case of the $k$-order adjusted transfer probability matrix $U^k$. Given a network $G$, we consider all paths that include a $k$-order sampling path from observed nodes to other nodes in the network, the motive of the loss function is to maximize the probability of a pair of nodes from a same group, and minimize the probability between the other nodes that are not in the same groups.

Here we denote the collection that is composed of pairs between observed nodes and other nodes and has the $k$-order sampling path between the pairs as $\mathcal{D}_k$, and the number of the pair $(v_i, v_j)$ that appears in $\mathcal{D}_k$ as $\#\left(v_i, v_j\right)_k$ . If the network is directed, the $\#\left(v_i, v_j\right)_k$ notation can be directional, that is, the number of the pair ($v_i$ to $v_j$) that appears in $\mathcal{D}_k$ . Similarly, $\#(v_i)_k$ and $\#(v_j)_k$ respectively denote the numbers of $v_i$ and $v_j$ that appear in $\mathcal{D}_k$ .

Specifically, we refer to SGNS (Gutmann & Hyvärinen, 2012) to define our loss function, which is similar to the discussion in Levy and Goldberg (2014) and Cao et al. (2015). First, we introduce the loss function based on the $k$-order adjusted transfer probability matrix $U^k$:

$$L_k = \sum_{v_i \in \mathcal{V}} L_k(v_i),$$

where,

$$L_k(v_i) = \left( \sum_{v_j \in \mathcal{V}} \#(v_i, v_j)_k \log\sigma\left(\boldsymbol{v}_{v_i} \cdot \boldsymbol{v}_{v_j}\right) \right. \\ \left. + \lambda \cdot \#(v_i)_k E_{v_{jn} \sim P_k(\mathcal{D}_k)} \left[\log\sigma\left(-\boldsymbol{v}_{v_i} \cdot \boldsymbol{v}_{v_{jn}}\right)\right] \right), \tag{2}$$

$\sigma(\cdot)$ is a sigmoid function, which is $\sigma(x) = (1+e^{-x})^{-1}$, $\lambda$ is the number of “negative” samples, $v_{jn}$ is the negative sample, and $P_k(\mathcal{D}_k)$ is the distribution of negative samples, which can be assumed to be an empirical unigram distribution, i.e., $P_k(v_{jn}) = \frac{\#(v_{jn})}{|\mathcal{D}_k|}$, where $v_{jn}$ follows the distribution $P_k(\mathcal{D}_k)$. The expression of the expected term is as follows:

$$E_{v_{jn} \sim P_k(\mathcal{D}_k)} \left[\log\sigma\left(-\boldsymbol{v}_{v_i} \cdot \boldsymbol{v}_{v_{jn}}\right)\right] = \sum_{v_{jn} \in \mathcal{D}_k} p_k(v_{jn}) \log\sigma\left(-\boldsymbol{v}_{v_i} \cdot \boldsymbol{v}_{v_{jn}}\right)$$

$$= p_k(v_j) \log\sigma\left(-\boldsymbol{v}_{v_i} \cdot \boldsymbol{v}_{v_j}\right) + \sum_{v_{jn} \in \mathcal{D}_k \setminus \{v_j\}} p_k(v_{jn}) \log\sigma\left(-\boldsymbol{v}_{v_i} \cdot \boldsymbol{v}_{v_{jn}}\right).$$

From the above two equations, the local loss function of the specific $(v_i, v_j)$ pairs can be defined approximately as:

$$L_k(v_i, v_j) = \#(v_i, v_j)_k \log\sigma\left(\boldsymbol{v}_{v_i} \cdot \boldsymbol{v}_{v_j}\right) \\ + \lambda \cdot \#(v_i)_k \cdot \frac{\#(v_j)_k}{|\mathcal{D}_k|} \log\sigma\left(-\boldsymbol{v}_{v_i} \cdot \boldsymbol{v}_{v_j}\right).$$

### 4.1.3. *Optimization of the* k*-order loss function.*

In this part, we detail how to optimize the $k$-order loss function obtained from the above definition.

Let $t = \boldsymbol{v}_{v_i} \cdot \boldsymbol{v}_{v_j}$, the partial derivative of $t$ for $L_k(v_i, v_j)$ is as follows:

$$\frac{\partial L_k}{\partial t} = \#(v_i, v_j)_k \sigma(-t) - \lambda \#(v_i)_k \frac{\#(v_j)_k}{|\mathcal{D}_k|} \sigma(t).$$

Let $\frac{\partial L_k}{\partial t} = 0$, then:

$$\#(v_i, v_j)_k \sigma(-t) - \lambda \#(v_i)_k \frac{\#(v_j)_k}{|\mathcal{D}_k|} \sigma(t) = 0.$$

Let $H = \frac{\#(v_i, v_j)_k}{\#(v_i)_k \cdot \#(v_j)_k}$, then:

$$e^{2t} - \left(\frac{|\mathcal{D}_k| H}{\lambda} - 1\right) e^t - \frac{|\mathcal{D}_k| H}{\lambda} = 0.$$

Let $z = e^t$, the above equation becomes the quadratic equation of $z$, and there are two solutions as follows: $z = -1$ (according to the definition of $z$, illegal), and:

$$z = \frac{|\mathcal{D}_k| H}{\lambda} = \frac{|\mathcal{D}_k|}{\lambda} \frac{\#(v_i, v_j)_k}{\#(v_i)_k \cdot \#(v_j)_k}.$$

Substituting $z$ with $e^t$, and $t$ with $\boldsymbol{v}_{v_i} \cdot \boldsymbol{v}_{v_j}$, has:

$$\boldsymbol{v}_{v_i} \cdot \boldsymbol{v}_{v_j} = \log\left(\frac{|\mathcal{D}_k|}{\lambda} \frac{\#(v_i, v_j)_k}{\#(v_i)_k \cdot \#(v_j)_k}\right) = \log\left(\frac{1}{\lambda} \frac{p_k(v_i, v_j)}{p_k(v_i) \cdot p_k(v_j)}\right). \tag{3}$$

The following equation can be obtained from Bayes formula:

$$p_k(v_i, v_j) = p_k(v_j | v_i) p_k(v_i). \tag{4}$$

Substituting Eqs. (4) and (1) into Eq. (3) yields:

$$\boldsymbol{v}_{v_i} \cdot \boldsymbol{v}_{v_j} = \log\left(\frac{1}{\lambda} \frac{p_k(v_j | v_i)}{p_k(v_j)}\right) = \log\left(\frac{1}{\lambda} \frac{U^k_{v_i, v_j}}{p_k(v_j)}\right), \tag{5}$$

where $\lambda$ is the number of “negative” samples, which is found to be useful for small training datasets when in the range 5–20, and effective for large datasets when in range 2–5 (Mikolov et al., 2013). In our study, we empirically set it to 5. $p_k(v_j)$ is the prior probability of $v_j$ in $\mathcal{D}_k$.

We can have the adjusted optimization matrix $M^k$ as follows:

$$M^k_{v_i, v_j} = \boldsymbol{Y}_{v_i, \cdot} \cdot \boldsymbol{C}^T_{\cdot, v_j} = \log\left(\frac{1}{\lambda} \frac{U^k_{v_i, v_j}}{p_k(v_j)}\right), \tag{6}$$

where $Y$ is the embedding matrix of the network, whose each row is the embedding of the node (Mikolov et al., 2013), and $C$ is the context matrix of the network, whose each row is the embedding of the context node.

From Eqs. (5) and (6), we could conclude that we need to factorize the matrix $M$ into two matrices $Y$ and $C$. Thus, the problem of optimizing the loss function of the $k$-order adjusted transfer probability matrix $U^k$ becomes the problem of factorization of the adjusted optimization matrix $M^k$.

## 4.2. *Positive adjusted optimization matrix*

To improve the sparsity and consistency of embeddings (Levy & Goldberg, 2014), all the negative values of $M^k$ can be replaced by 0, and then a positive $k$-order optimization matrix $X^k$ is obtained, which is:

$$X^k_{i,j} = max(M^k_{i,j}, 0). \tag{7}$$

Now, we can see that the $k$-order input structure information needed to map from the high-dimensional space to the low-dimensional space is the matrix $X^k$, and $X^k$ can be calculated according to Eq. (7) so as to facilitate the subsequent processing of the optimization step.

In summary, we know that the optimization problem with $k$-order loss function is equivalent to the decomposition of the positive adjusted optimization matrix $X^k$. At present, there are a variety of matrix factor decomposition techniques, such as the truncated SVD method (Levy & Goldberg, 2014) , etc. However, they do not have the ability to capture the complex non-linear relationships. At the same time, the popular singular value decomposition method and the autoencoder have some similarities in terms of optimization (Tian et al., 2014), and the other related research results (Bengio et al., 2013; Vincent et al., 2008; Wang et al., 2016) illustrate that the autoencoder can map the data in high-dimensional space to the low-dimensional space in non-linear form, retain more non-linear structure relations in networks, and obtain better results in experiments. Moreover, the works (Cao et al., 2016; Gutmann & Hyvärinen, 2012) illustrate and demonstrate the advantages of the denoise autoencoder in the aspects of anti-noise and optimization of dimension reduction.

Therefore, to better preserve the rich structure information, and solve the problem of noise, we choose the stacked denoise autoencoder to optimize the loss function of our study.

## 4.3. *Stacked denoise autoencoder*

The stacked denoise autoencoder (Vincent et al., 2008) is a popular model in deep learning, that can generate compressed and low-dimensional vectors from the primitive high-dimensional vectors, because it layer-wise pre-trains the weights of the deep network to learn more robust embeddings. Indeed, the stacked denoise autoencoder introduces random noise at the visual layer of the network (i.e., the input layer of data), with the damaged input data to reconstruct the original data (that is, no damage to the data), so the parameters trained in this manner are more robust. Specifically, for each input sample vector $\boldsymbol{x}$, the partial position of the vector is randomly set to 0 with a certain probability, and the other process is similar to that in standard autoencoders.

The training process is described with a single denoise autoencoder as follows: Let $\boldsymbol{X}_i$ correspond to the input data, $\bar{\boldsymbol{X}}_i$ correspond to the input data containing noise, $\boldsymbol{H}_i$ correspond to the latent embedding learned by the hidden layer, and $\theta = \{\boldsymbol{W}_e, \boldsymbol{d}_e, \boldsymbol{W}_d, \boldsymbol{d}_d\}$ represent the parameters that need to be learned in the encode and decode phases. Then, to learn the non-linear embedding of $Y$ that can best reconstruct the original data $X$, we have:

$$\begin{aligned}\hat{\theta} = \underset{\theta}{\operatorname{argmin}}\, L_\theta\left(X, \bar{X}\right) &= \underset{\theta}{\operatorname{argmin}} \sum_{i=1}^{N} L_\theta\left(\boldsymbol{X}_i, \overline{\boldsymbol{X}_i}\right) \\ &= \underset{\theta}{\operatorname{argmin}} \sum_{i=1}^{N} L_\theta\left(\boldsymbol{X}_i, g\left(s\left(\overline{\boldsymbol{X}_i}\right)\right)\right),\end{aligned} \tag{8}$$

where $L_\theta\left(\boldsymbol{X}_i, \overline{\boldsymbol{X}_i}\right)$ is the distance function that measures the reconstruction error. We use the Euclidean distance in our study. The $s(\cdot)$ and $g(\cdot)$ are the non-linear mapping functions in the encode and decode phases, respectively. After the denoise greedy layer-by-layer training (Bengio et al., 2007; Vincent et al., 2008), the parameters $\boldsymbol{W}_e$, $\boldsymbol{d}_e$ can be obtained, and then the embeddings of all nodes can be obtained.

### 4.4. Combining layer

The main function of the combining layer is to integrate the multiple latent embeddings of each node obtained in previous parts to get the final robust embedding. At present, the main approaches of integration include: concatenation and weighted sum. In the approach of weighted sum, the weights could be automatically determined by labeled training data through attention mechanism. In order to maintain the generality and flexibility of our overall framework, we adopt these two methods in experiments. Here, we briefly introduce the approach of weighted sum. The main idea of the approach of weighted sum is to filter the redundant information and retain the effective information. At the same time, attention mechanism has the characteristic well satisfying this demand. By utilizing it in the training of the model proposed, to calculate the combined weights of the latent embeddings of each node obtained by the previous modules, the combining layer can adjust the flow of fusion information in our study (Chung, Gulcehre, Cho, & Bengio, 2014; Zhang, Chen, Liu, Liu, & Lv, 2017).

Specifically, we apply the method with attention mechanism to calculate the weights of the latent embeddings in different spaces. Because the approach of weighted sum is related to the specific task, here, we introduce the case of node classification, and the other cases are similar. The relevant parameters and weights are optimized by the learning algorithm proposed in Section 4.5. In this manner, we define:

$R_j^k = \sigma\left(\boldsymbol{G}_{jk} \cdot Y^k + b_{jgk}\right)$,

where $R_j^k$ is the weight of the $k$-order latent embedding calculated by the method with attention mechanism in the case of category $j$, and $Y^k$ is the output of the stacked denoise autoencoder corresponding to the $k$-order input matrix. $\boldsymbol{G}_{jk}$ and $b_{jgk}$ represent the weights and biases of the $k$th part in the case of category $j$, respectively. $\sigma(.)$ is the sigmoid function.

Then, we combine all the $k$-order latent embedding matrices together to represent the nodes of the network and calculate the output in the given case by the following equation:

$P(label_j|X_i) = \sigma\left(M_j \cdot \Sigma_{k=0}^{K} (R_j^k \cdot Y^k) + b_{jm}\right)$,

where $M_j$ and $b_{jm}$ represent the weights and biases of the $\sigma$ function in the case of category $j$ respectively, $R_j^k$ is the scalar weight of the $k$-order latent embedding calculated by the method with attention mechanism in the case of category $j$, $Y^k$ is the output of the stacked denoise autoencoder corresponding to the $k$-order input matrix.

By concatenating the multiple latent embeddings of each node obtained in previous parts, or by the approach of weighted sum with attention mechanism, we could get the robust and discriminable latent embedding with more effective information to some degree.

Thus, we have detailed the four components of SENN, i.e., the adjusted optimization matrix $M^k$, positive adjusted $k$-order optimization matrix, stacked denoise autoencoder, and the combining layer. In the next subsection, we introduce the algorithm for training SENN.

### 4.5. Training algorithm for SENN

We introduce the training algorithm for SENN with regard to the following aspects: the loss function of optimization, model initialization and update, and complexity analysis.

#### 4.5.1. Loss function of optimization

The loss function of training the entire architecture of SENN is mainly composed of two parts, the loss function of the deep networks and the loss function of the combining layer. For convenience, we unify the two parts in an equation, but we must note that the training of the two parts is carried out separately. The equation of the unified expression is:

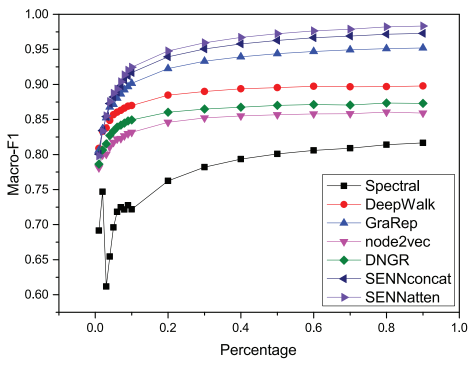
$$\begin{aligned}L = &\sum_{k=1}^{K} \underset{\theta_k}{\operatorname{argmin}} \sum_{i=1}^{N} L_{\theta_k}\left(\boldsymbol{X}_i, g(s(\boldsymbol{X}_i))\right) \\ &- \frac{1}{N}\sum_{j=1}^{T}\sum_{i=1}^{N}\left(\begin{array}{l} label_j \cdot \log(P(label_j|\boldsymbol{X}_i)) \\ +\left(1 - label_j\right) \cdot \left(1 - \log\left(P\left(label_j|\boldsymbol{X}_i\right)\right)\right)\end{array}\right),\end{aligned} \tag{9}$$

where the first term corresponds to the loss of deep networks and the second term corresponds to the loss of the combining layer. $N$ is the total number of input samples, $T$ is the total number of output categories, $i$ represents the $i$th instance, $j$ represents the $j$th class, and $P(labe_j|\boldsymbol{X}_i)$ denotes the probability of the corresponding category $j$ under the condition of input $\boldsymbol{X}_i$.

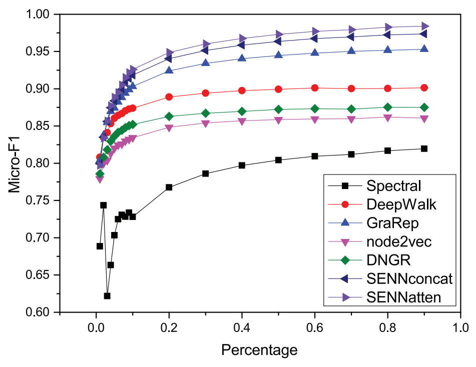
#### 4.5.2. Model initialization and update

The parameters to be trained are $\left\{\boldsymbol{M}_j^k, b_{jmk}, \boldsymbol{G}_j^k, b_{jgk}\right\}$, where $j$ represents the $j$th label in the node classification task and $k$ represents the $k$-order. When we initialize the model, we set the values of the weight vectors $\boldsymbol{M}_j^k$ and $\boldsymbol{G}_j^k$ to the interval $[-\sqrt{6/(n_{in} + n_{out})}, \sqrt{6/(n_{in} + n_{out})}]$ randomly, which is the same as Orr and Müller (2003), and set the biases $b_{jmk}$ and $b_{jgk}$ to zero.

After the parameter initialization, we use the BP algorithm to train the model, that is, the loss function is minimized by the stochastic gradient descent (SGD) algorithm (Bottou, 2010; Zhang, 2004). Specifically, we use a “mini-batch” to speed up the training process, set the batch size between 10–50, and set the initial learning rate between 5–25. Considering different characteristics of the datasets, their specific batch sizes and learning rates may be different. In addition, to avoid overfitting, the learning rate should be updated dynamically after a certain number of iterations. Here, we halve the learning rate until it reaches the user-specified minimum threshold after a certain quantity (e.g., 100) of batch data.

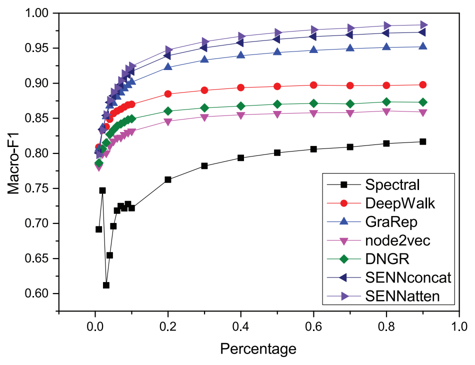


(a) Macro-F1

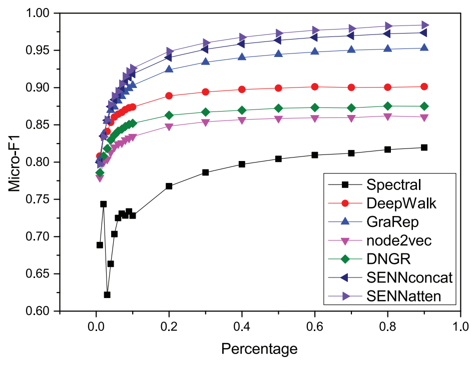


(b) Micro-F1

**Fig. 3.** Performance results on BlogCatalog.



(a) Macro-F1



(b) Micro-F1

**Fig. 4.** Performance results on FLICKR.

### *4.5.3. Complexity analysis*

We consider the time complexity of the main processes (i.e., computing $A^k$, calculating $B$, obtaining $M^k$, training the stacked donise autoencoder, and training the combining layer) in SENN. With adoption of the fast procedure introduced by Le Gall (2014), the procedure of computing $A^k$ has $O((k-1)N^{2.3728639})$ time complexity. The complexity of obtaining $B$ is $O(nnz(A)c^2)$, where $nnz(\cdot)$ indicates the number of non-zero elements and $c$ is the average degree of the network. The complexity of calculating $M^k$ is $O(nnz(A)^2)$. The training complexity of the stacked donise autoencoder is $O(Ncd(l-2)I_{denoise})$, where $c$ is the average degree of the network, $l$ is the number of layers of the stacked donise autoencoder, and $I_{denoise}$ is the number of iterations. The training complexity of the attention mechanism part is $O(NdI_{am})$, where $I_{am}$ is the number of iterations. In general, $N$ is far larger than other parameters. Therefore, the overall time complexity of our model is $O((K-1)N^{2.3728639})$ .

## 5. Experiments

To evaluate the performance of SENN, we construct extensive experiments in this section. Specifically, this section introduces the datasets, baselines, evaluation metrics, experimental settings, and results. We use four tasks to test in experiments. First, to verify the discriminative performance of our model, we conduct experiments on the multi-label node classification task. Second, to demonstrate the selection ability of the structure pattern of our model, we conduct an experiment on a network of characters of a novel named Huckleberry Finn (Knuth, 1993). Third, to demonstrate the performance of our model more intuitively, we report a visualization-task result. Finally, in order to evaluate the effectiveness under different parameter values, we conduct the experiments of parameter sensitivity.

### *5.1. Datasets*

We adopt the following public datasets combining the characteristics of the tasks in our experiments:

- **BlogCatalog**[1]: It is a real-world social network of bloggers listed on the BlogCatalog website, which has 10,312 nodes, 333,983 edges, and 39 different topic categories as labels presented by authors. It will be used in the multi-label classification task and parameter sensitivity task.
- **FLICKR**[2]: A real-world network of contacts between users of the photo sharing website (Tang & Liu, 2009). It has 80,513 nodes, 11,799,764 edges, and 195 different categories. It will be used in the multi-label classification.
- **YOUTUBE**[3]: This is another popular social network of online users. Each user is labeled by at least one category. It has 1, 138, 499 nodes, 5,980, 886 edges, and 47 categories. The categories can be used as the ground-truth of each node. This dataset is used in the multi-label classification task.
- **Wikipedia** (Mahoney, 2011): This is a word co-occurrence network of words appearing in first million bytes of the Wikipedia dump. Each node is labeled by at least one label. The labels represent the Part-of-Speech (POS) tags inferred using the Stanford POS-Tagger (Toutanova, Klein, Manning, & Singer, 2003).

[1] http://leitang.net/code/social-dimension/data/blogcatalog.mat
[2] http://leitang.net/code/social-dimension/data/flickr.mat
[3] http://leitang.net/code/social-dimension/data/youtube.mat

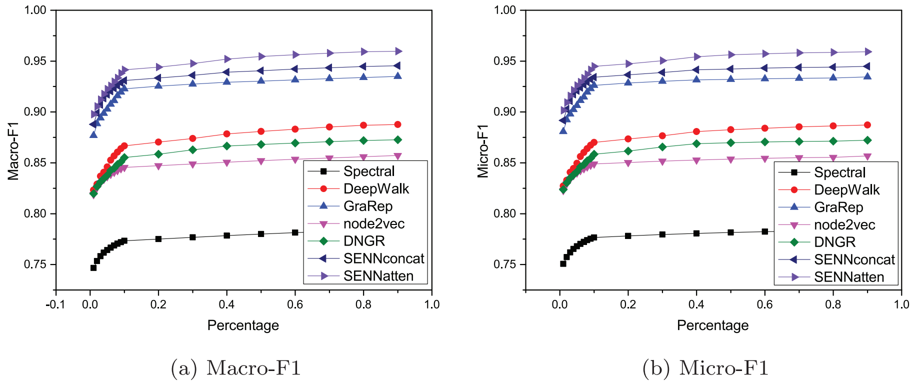


**Fig. 5.** Performance results on YOUTUBE.

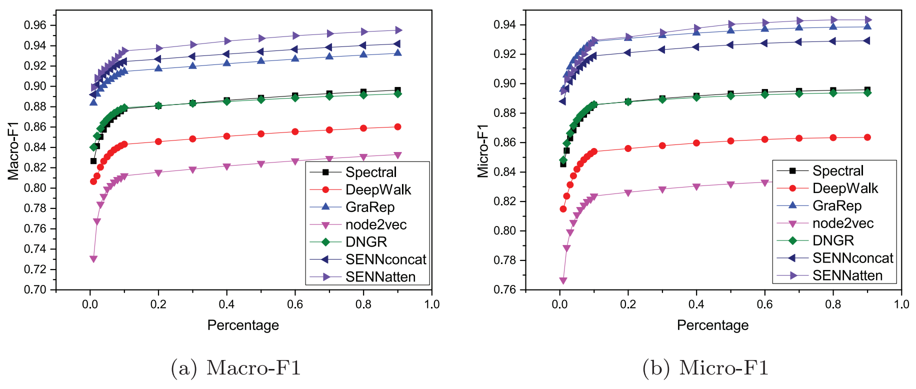


**Fig. 6.** Performance results on Wikipedia.

The network has 4777 nodes, 184,812 edges, and 40 different labels. This dataset is also used in the multi-label classification task.

- **Huck**[4]: A network in which nodes correspond to the roles in the novel Huckleberry Finn (Knuth, 1993), and edges represent the relationship between the co-appearing characters. It has 74 nodes and 301 edges. This dataset is used for the case study experiment.
- **20-NewsGroup**[5]: This dataset contains approximately 20,000 News Group (NG) documents and is split into 20 different groups according to categories. We represent every document as a vector of *tf-idf* scores of each word, and built the cosine similarity network based on the *tf-idf* scores. In our experiments, to carry out the visualization task, we construct one network of 3 different news groups, whose names are labeled as comp.graphics, rec.sport.baseball and talk.politics.gums. For convenience and without loss of generality, we extracted 300 samples from each of them randomly.

### 5.2. Baselines

We compare our model with other six start-of-the-art and widely-used network embedding methods which only utilize the structure topology information. The selected baselines are matrix factorization based approaches, such as Spectral clustering (Tang & Liu, 2011), and GraRep (Cao et al., 2015), random walk based methods, such as DeepWalk (Perozzi et al., 2014), LINE (Tang et al., 2015), node2vec (Grover & Leskovec, 2016), and deep learning techniques, such as DNGR (Cao et al., 2016). Moreover, we denote SENN*concat* which uses the concatenation approach in the combining layer to get the network embedding, and SENN*atten* which uses the weighted sum approach based on attention mechanism.

### 5.3. Evaluation metrics

In our experiment, we evaluate the average performances by the measures of Micro-F1 and Macro-F1 in the same manner as many other works (Cao et al., 2015; Tang & Liu, 2009). In detail, we denote $TP(L)$, $FP(L)$ and $FN(L)$ as the number of true positives, false positives and false negatives in the classifying instances that are predicted as $L$, respectively. Suppose $A$ is the overall label set. Micro-F1 and Macro-F1 are defined as follows:

$$Pr = \frac{\sum_{L\in A} TP(L)}{\sum_{L\in A}(TP(L)+FP(L))},$$

$$R = \frac{\sum_{L\in A} TP(L)}{\sum_{L\in A}(TP(L)+FN(L))},$$

$$Micro-F1 = \frac{2*Pr*R}{Pr+R},$$

$$Macro-F1 = \frac{\sum_{L\in A} F1(L)}{|A|},$$

where $F1(L)$ is the $F1$-measure for the label $L$.

### 5.4. Experimental settings

For DeepWalk, as suggested in Perozzi et al. (2014), we set the walk length $\eta = 4$, walks per node $\gamma = 80$, and window size $w = 10$ . For LINE, we set the number of negative samples $K = 5$, as used in Tang et al. (2015). For node2vec, similar to Grover and Leskovec (2016), we set $p, q \in \{0.25, 0.50, 1, 2, 4\}$ with a grid search as the default values. For DNGR, the neural network structures

[4] ftp://ftp.cs.stanford.edu/pub/sgb/sgb.tar.gz
[5] qwone.com/ jason/20Newsgroups/

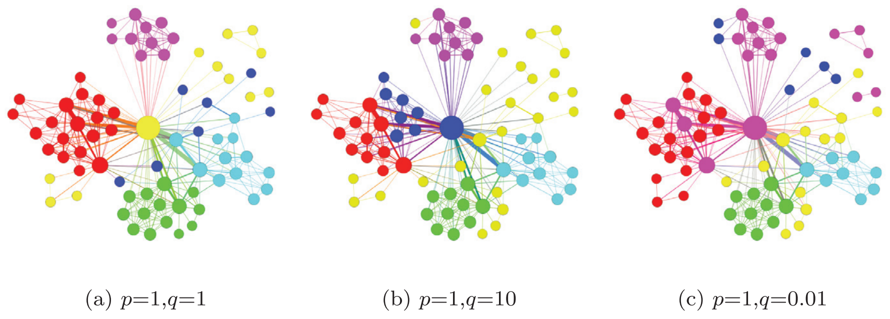


**Fig. 7.** Case study results for nodes clustering in network. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

**Table 2**
Neural network structures of DNGR.

| Dataset | #Nodes in each layer |
|---|---|
| Huck | 74-32-16 |
| NG | 900-512-256-128-64 |
| BlogCatalog | 10,312-4,096-2,048-1,024-512-256 |
| FLICKR | 80,513-32,768-16,384-8,192-4,096-2,048-1,024-512-256 |
| YOUTUBE | 22,693-8,192-2,048-1,024-512-256 |
| Wikipedia | 4,777-2,048-1,024-512-256 |

**Table 3**
Stacked denoise autoencoder in SENN.

| Dataset | #Nodes in each layer |
|---|---|
| Huck | 74-32-16 |
| NG | 900-256-64 |
| BlogCatalog | 10,312-2,048-256 |
| FLICKR | 80,513-16,384-2,048-256 |
| YOUTUBE | 22,693-2,048-256 |
| Wikipedia | 4,777-1,024-256 |

are shown in Table 2 according to Cao et al. (2016). For SENN, we set the parameters $p$, $q \in \{0.01, 1, 10\}$ with a grid search in the multi-label classification task. At the same time, we set the parameters of the stacked denoise autoencoder used in SENN, as shown in Table 3. Moreover, to solve the problem of imbalance of samples, we adopt the algorithm smote (Chawla, Bowyer, Hall, & Kegelmeyer, 2002) for resampling, which results in that the same algorithm with the same dataset has not the same scores comparing with some other papers.

## 5.5. Results

In this subsection, we report the results on each experimental task.

### 5.5.1. Task 1: multi-label node classification

In this part, we evaluate the effectiveness of our model on the multi-label classification task with the datasets BlogCatalog, FLICKR, YOUTUBE and Wikipedia. In addition, we remove the nodes which are not labeled in YOUTUBE. Following Sarwar, Karypis, Konstan, and Riedl (2002) and Tang and Liu (2009), we use the LIBLINEAR package (Fan, Chang, Hsieh, Wang, & Lin, 2008) to train one-vs.-rest logistic regression classifiers, and use them to deal with the other samples. We run this process 10 times and report the average Macro-F1 and Micro-F1 scores. For each time, we randomly sample 1% to 90% of the instances for training, and use the remaining instances for testing. In this experiment, we set the dimension of the embeddings as 256. For GraRep and SENN, we set the maximum order of the transition matrix as 5.

Figs. 3(a), 3(b), 4(a), 4 (b), 5(a), 5(b), and 6(a), 6(b) report the results of SENN and other methods. Overall, even with less than 10% of the instances used for training, SENN is significantly better than other methods. This indicates that different and complementary orders of rich structure information learned by SENN can be chosen and enhanced by each other effectively, such that the embedding for the classification task can be completed better.

### 5.5.2. Task 2: case study with the Huckleberry Finn network

In this part, aiming to empirically demonstrate that our adjusted model has the ability to select the wishing structure similar pattern information based on the settings of the adjusted parameters, we perform a case study on the Huck dataset. Firstly, we set $d$=16, the maximum order of the transition matrix to 5, and utilize our model to learn the final embedding for each node in the network. For the simplicity and effectiveness of this experiment, we use the concatenation approach in the combining layer of our model to get the network embedding. Then, the final embeddings are clustered using $k$-means. Finally, we visualize the character structure in two dimensions with nodes assigned colors based on their clusters.

Fig. 7(a) shows the case when we set $p$=1, and $q$=1. Let us pay attention to the regions colored the same. It can be seen that the embeddings obtained by our model have rich structure equivalence and homogeneous information. For example, the yellow colored nodes are categorized into one group, not only because they are close and interact with each other frequently, but also because different parts have similar equivalence structure from the major sub-plots of the novel.

Fig. 7 (b) shows the example in which we set $p$=1, and $q$=10. It can be seen that our model focuses more on the local homogeneous information, and the structure equivalence information is small. For instance, the blue nodes are mainly separated from the red ones, some of the yellow nodes are separated from the green ones, and so on. This result proves that our model has the ability to identify homogeneous information, and increase the proportion of homogeneous information in this setup.

To discover nodes that have similar structural roles, let us set $p$=1, and $q$=0.01. Theoretically, our model in this setup is more focused on similar structure equivalence information. In fact, our model obtains a more complementary assignment of nodes to groups that have similar structural roles in the network, as shown in Fig. 7 (c). For example, our model sets some node colors to pink, mainly because they have a similar role status. Other colors (such as red, sky blue, yellow and green) are similar. This result shows that, our model increases the proportion of structure equivalence information in the embeddings of nodes in this setup.

Through the case study experiments above, we can see that the embeddings obtained by our model not only have abundant structure information, but also can adjust and balance different types of structure similar information by hyper-parameters.

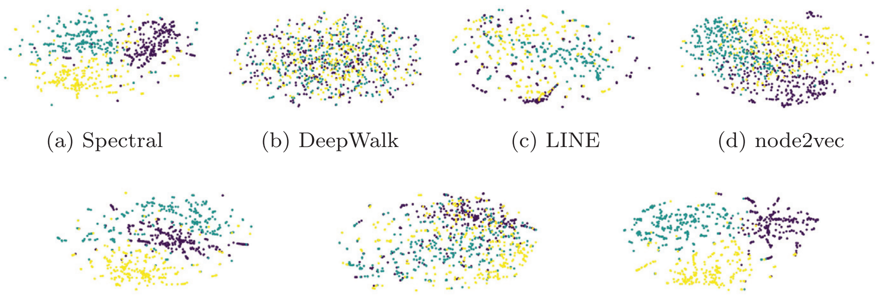
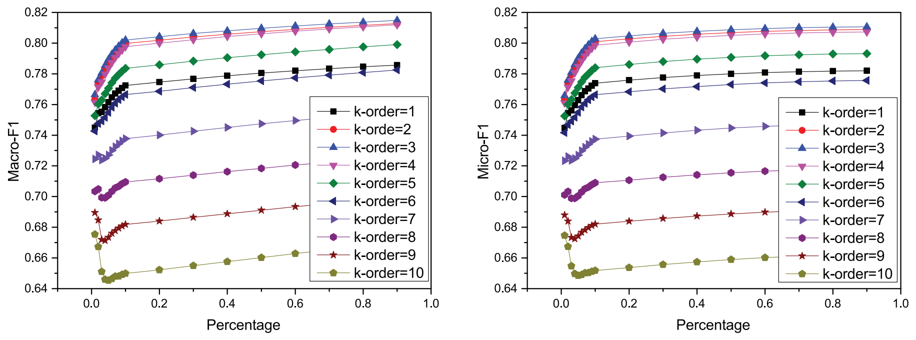

(a) Spectral (b) DeepWalk (c) LINE (d) node2vec
(e) GraRep (f) DNGR (g) SENNconcat

**Fig. 8.** Visualization results for 20-NewsGroup network.



(a) Macro-F1 (b) Micro-F1

**Fig. 9.** Performance over *k*-order on BlogCatalog.

### 5.5.3. *Task 3: visualization*

To demonstrate the discriminative power of SENN, we focus on visualizing the learned embeddings of the 20-NewsGroup network. For simplicity, we use the concatenation approach in the combining layer of our model to get the network embedding. We feed the network embeddings learned by different network embedding methods into the standard *t*-SNE tool (Maaten & Hinton, 2008), where each NewsGroup document is mapped as a two-dimensional vector, and the documents labeled as the same category have the same color. A good visualization result is that the points of the same color are close to each other. Results are shown in Fig. 8.

From Fig. 8, we can see that, the result of DeepWalk is almost random. It is because DeepWalk is only suitable for unweighted network, the embeddings obtained by it can not reflect the effect of the weights of the network. For LINE(1st), we use the source code provided by Tang et al. (2015), adopt the recommended parameters in Tang et al. (2015), and select the best results. The result of LINE (1st) is slightly better than that of DeepWalk, we regard this because LINE(1st) can utilize the weights in the network. For DNGR, the result is slightly better than that of LINE(1st), but many documents belonging to the same categories are not clustered together, it is believed that the embeddings obtained by this method can not preserve more discriminative structure information. For Spectral, node2vec and GraRep, the results are better, since the clusters of different categories are formed, although the boundaries of each group are not very clear. Obviously, from the visualization result of SENN, we observe that SENN performs best in the aspects of group separation and boundary clarity. The result shows that our model is effective.

### 5.5.4. *Task 4: parameter sensitivity*

In the examination of parameter effects in SENN, we assess the effect of each *k*-order, the maximal size *K*, and the dimension *d* in our model. For simplicity, we utilize the concatenation approach in the combining layer of our model.

Fig. 9 shows the Macro-F1 and Micro-F1 scores over different choices of *k*-order on the BlogCatalog dataset with the dimension $d$=128. In both Fig. 9 (a) and (b), the scores of the 3-order are the highest, followed by the scores of 2, 4, 5, 1, 6, 7, 8, 9 and 10-order, but when $k$ increases to more than 5, the learned *k*-order latent embeddigngs contain less effective information.

Fig. 10 shows the Macro-F1 and Micro-F1 scores over different choices of $K$ on the BlogCatalog dataset with dimension $d$=128. From both Fig. 10 (a) and (b), we can observe that the increase rate of the scores decreases gradually with the increase of $K$. When $K$ is greater than 5, the increase can be neglected. The results of Figs. 9 and 10 imply that different *k*-order matrices can learn complementary information.

Fig. 11 (a) and (b) show the Macro-F1 and Micro-F1 scores of each algorithm over different settings of dimension $d$ on the BlogCatalog dataset. From Fig. 11 (a) and (b), we can observe that SENN consistently outperforms other baselines when they learn embeddings with the same dimension. When we increase $d$ from 32 to larger values, the Macro-F1 and Micro-F1 scores of all algorithms increase with different rates. Interestingly, when $d$ is greater than 256, the growth rate of SENN, GraRep and node2vec becomes very small, and the other algorithms have similar small rate when $d$ is greater than 512. At the same time, with the increase of $d$, the rate of improvement is smaller. Nevertheless, our SENN is always better than other baselines across all different $d$ values.

We also conduct experiments with the BlogCatalog dataset in the case of $d = 256$ to analyze the effect of each part of SENN on the performance improvement. Through the experimental comparison, it is found that, after adopting the adjusted matrix strategy, our model beats the baselines by at least 1%, with the stacked denoise autoencoder, the effect can be increased by at least 1.5%, and with adoption of attention mechanism, the effect can be increased

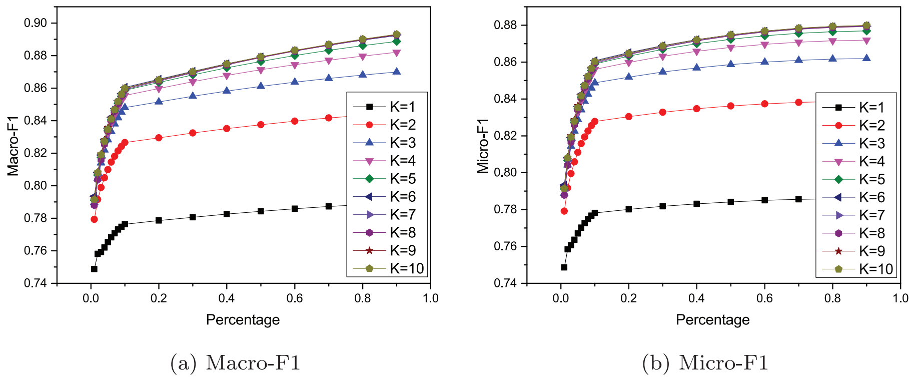

(a) Macro-F1 (b) Micro-F1

**Fig. 10.** Performance over order $K$ on BlogCatalog.

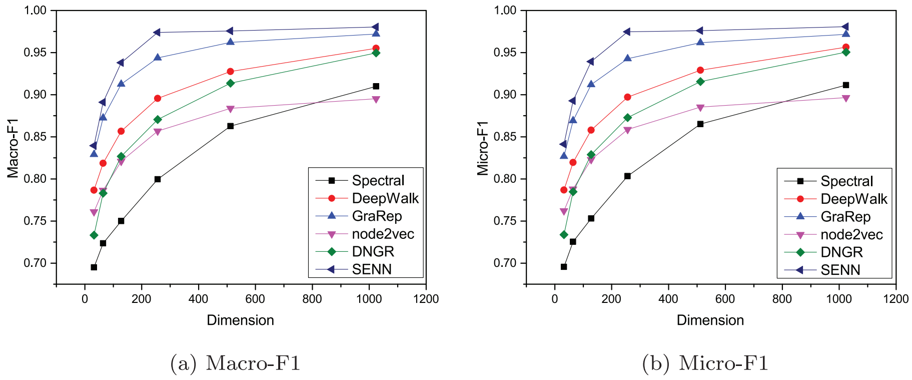

(a) Macro-F1 (b) Micro-F1

**Fig. 11.** Performance over dimension $d$ on BlogCatalog.

by at least 1%. These results clearly demonstrate the utility of each component in our proposed SENN model.

## 6. Conclusions

In this paper, we propose a structure-enriched network embedding model, i.e., SENN, for network embedding. Specifically, we present the concept of the adjusted transfer probability matrix to select the diverse structure similar patterns observed in networks, adopt the stacked denoise autoencoder to capture the highly non-linear structure information and make the learned network embeddings more robust. Meanwhile, we integrate the multi-order structure information by the combining layer. In experiments, we evaluate the generated embeddings with a few synthetic and real-world datasets. The results demonstrate the superior performance of our method compared with the state-of-the-art methods. In the future, we will combine the specific application scenarios to explore how to use more of other information to learn the network embedding.

## Acknowledgments

This research was partially supported by grants from the National Natural Science Foundation of China (Grants No. U1605251 and 61727809).

## References

Airoldi, E. M., Bai, X., & Carley, K. M. (2011). Network sampling and classification: An investigation of network model representations. *Decision Support Systems, 51*(3), 506–518.
Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. arXiv:1409.0473v1.
Bayer, A. O., & Riccardi, G. (2016). Semantic language models with deep neural networks. *Computer Speech & Language, 40*, 1–22.
Belkin, M., & Niyogi, P. (2002). Laplacian eigenmaps and spectral techniques for embedding and clustering. In *Advances in neural information processing systems* (pp. 585–591).
Bengio, Y., Courville, A., & Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 35*(8), 1798–1828.
Bengio, Y., Lamblin, P., Popovici, D., & Larochelle, H. (2007). Greedy layer-wise training of deep networks. In *Advances in neural information processing systems* (pp. 153–160).
Bengio, Y., et al. (2009). Learning deep architectures for ai. *Foundations and Trends® in Machine Learning, 2*(1), 1–127.
Bhagat, S., Cormode, G., & Muthukrishnan, S. (2011). Node classification in social networks. In *Social network data analytics* (pp. 115–148). Springer.
Bottou, L. (2010). Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010* (pp. 177–186). Springer.
Bullinaria, J. A., & Levy, J. P. (2012). Extracting semantic representations from word co-occurrence statistics: Stop-lists, stemming, and svd. *Behavior Research Methods, 44*(3), 890–907.
Cao, S., Lu, W., & Xu, Q. (2015). Grarep: Learning graph representations with global structural information. In *Proceedings of the 24th ACM international on conference on information and knowledge management* (pp. 891–900). ACM.
Cao, S., Lu, W., & Xu, Q. (2016). Deep neural networks for learning graph representations. In *AAAI* (pp. 1145–1152).
Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). Smote: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research, 16*, 321–357.
Chorowski, J., Bahdanau, D., Cho, K., & Bengio, Y. (2014). End-to-end continuous speech recognition using attention-based recurrent nn: First results. arXiv:1412.1602.
Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv:1412.3555v1.
Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Mao, M., et al. (2012). Large scale distributed deep networks. In *Advances in neural information processing systems* (pp. 1223–1231).
Dong, Y., Chawla, N. V., & Swami, A. (2017). metapath2vec: Scalable representation learning for heterogeneous networks. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 135–144). ACM.
Fan, R.-E., Chang, K.-W., Hsieh, C.-J., Wang, X.-R., & Lin, C.-J. (2008). Liblinear: A library for large linear classification. *Journal of Machine Learning Research, 9*(Aug), 1871–1874.
Grover, A., & Leskovec, J. (2016). 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.
Gutmann, M. U., & Hyvärinen, A. (2012). Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics. *Journal of Machine Learning Research, 13*(Feb), 307–361.
Hang, T. T. P., & Stemberg, M. J. E. (2012). Aligning protein-protein interaction networks using random neural networks.
Hinton, G. E., & Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *Science, 313*(5786), 504–507.
Knuth, D. E. (1993). *The stanford graphbase: A platform for combinatorial computing*: 37. Addison-Wesley Reading.
Le Gall, F. (2014). Powers of tensors and fast matrix multiplication. In *Proceedings of the 39th international symposium on symbolic and algebraic computation* (pp. 296–303). ACM.
Levy, O., & Goldberg, Y. (2014). Neural word embedding as implicit matrix factorization. In *Advances in neural information processing systems* (pp. 2177–2185).
Maaten, L. v. d., & Hinton, G. (2008). Visualizing data using t-sne. *Journal of Machine Learning Research, 9*(Nov), 2579–2605.
Mahoney, M. (2011). Large text compression benchmark. URL: http://www.mattmahoney.net/text/text.html,.
Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems* (pp. 3111–3119).
Mnih, V., Heess, N., Graves, A., et al. (2014). Recurrent models of visual attention. In *Advances in neural information processing systems* (pp. 2204–2212).
Orr, G. B., & Müller, K.-R. (2003). *Neural networks: Tricks of the trade*. Springer.
Parthasarathy, S., Ruan, Y., & Satuluri, V. (2011). Community discovery in social networks: Applications, methods and emerging trends. In *Social network data analytics* (pp. 79–113). Springer.
Perozzi, B., Al-Rfou, R., & Skiena, S. (2014). 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.
Sarwar, B., Karypis, G., Konstan, J., & Riedl, J. (2002). Incremental singular value decomposition algorithms for highly scalable recommender systems. In *Fifth international conference on computer and information science* (pp. 27–28). Citeseer.
Tang, J., Qu, M., Wang, M., Zhang, M., Yan, J., & Mei, Q. (2015). Line: Large-scale information network embedding. In *Proceedings of the 24th international conference on world wide web* (pp. 1067–1077). International World Wide Web Conferences Steering Committee.
Tang, J., Sun, J., Wang, C., & Yang, Z. (2009). Social influence analysis in large-scale networks. In *Proceedings of the 15th ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 807–816). ACM.
Tang, L., & Liu, H. (2009). Relational learning via latent social dimensions. In *Proceedings of the 15th ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 817–826). ACM.
Tang, L., & Liu, H. (2011). Leveraging social media networks for classification. *Data Mining and Knowledge Discovery, 23*(3), 447–478.
Tenenbaum, J. B., De Silva, V., & Langford, J. C. (2000). A global geometric framework for nonlinear dimensionality reduction. *Science, 290*(5500), 2319–2323.
Tian, F., Gao, B., Cui, Q., Chen, E., & Liu, T.-Y. (2014). Learning deep representations for graph clustering. In *AAAI* (pp. 1293–1299).
Toutanova, K., Klein, D., Manning, C. D., & Singer, Y. (2003). Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the 2003 conference of the North American chapter of the association for computational linguistics on human language technology-volume 1* (pp. 173–180). Association for Computational Linguistics.
Vincent, P., Larochelle, H., Bengio, Y., & Manzagol, P.-A. (2008). Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on machine learning* (pp. 1096–1103). ACM.
Wang, D., Cui, P., & Zhu, W. (2016). Structural deep network embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 1225–1234). ACM.
Zhang, K., Chen, E., Liu, Q., Liu, C., & Lv, G. (2017). A context-enriched neural network method for recognizing lexical entailment. In *AAAI* (pp. 3127–3134).
Zhang, T. (2004). Solving large scale linear prediction problems using stochastic gradient descent algorithms. In *Proceedings of the twenty-first international conference on machine learning* (p. 116). ACM.