A Semi-Supervised Deep Network Embedding Approach Based on the Neighborhood Structure

Wenmao Wu
School of Computer Science and Engineering MOE Key Laboratory of Computer Network and Information Integration, Southeast University, Nanjing 211100, China.

Zhizhou Yu
School of Computer Science and Engineering MOE Key Laboratory of Computer Network and Information Integration, Southeast University, Nanjing 211100, China.

Jieyue He
School of Computer Science and Engineering MOE Key Laboratory of Computer Network and Information Integration, Southeast University, Nanjing 211100, China.

Follow this and additional works at: https://tsinghuauniversitypress.researchcommons.org/big-data-mining-and-analytics

Part of the Computer Engineering Commons, Computer Sciences Commons, and the Data Science Commons

Recommended Citation
A Semi-Supervised Deep Network Embedding Approach Based on the Neighborhood Structure

Wenmao Wu, Zhizhou Yu, and Jieyue He*

Abstract: Network embedding is a very important task to represent the high-dimensional network in a low-dimensional vector space, which aims to capture and preserve the network structure. Most existing network embedding methods are based on shallow models. However, actual network structures are complicated which means shallow models cannot obtain the high-dimensional nonlinear features of the network well. The recently proposed unsupervised deep learning models ignore the labels information. To address these challenges, in this paper, we propose an effective network embedding method of Structural Labeled Locally Deep Nonlinear Embedding (SLLDNE). SLLDNE is designed to obtain highly nonlinear features through utilizing deep neural network while preserving the label information of the nodes by using a semi-supervised classifier component to improve the ability of discriminations. Moreover, we exploit linear reconstruction of neighborhood nodes to enable the model to get more structural information. The experimental results of vertex classification on two real-world network datasets demonstrate that SLLDNE outperforms the other state-of-the-art methods.

Key words: network embedding; deep learning; network analysis

1 Introduction

Over recent years, networks have become important and pervasive data carriers, with the scale of data becoming larger and the relationship between data becoming more complex than ever before. Network structures generate a huge potential for data mining, which is beneficial for many network analysis tasks, such as vertex classification[1], visualization[2], and link prediction[3]. Therefore, network analysis technology is very important to many real-world applications. However, traditional network analysis technology has some limitations. Firstly, the computational complexity of traditional methods is too high to be applied to modern large-scale networks. Secondly, when analyzing these modern networks, it is difficult to capture the relationship between nodes, which means that traditional methods have trouble quantifying the similarity between nodes. Thirdly, traditional technologies of network analysis cannot be applied particularly well to most off-the-shelf machine learning methods. To tackle these problems, network embedding methods have been proposed with the aim of devising low-dimensional node representations.

In the early stages, matrix factorization methods, such as Singular Value Decomposition (SVD), were applied to network embedding. The goal of these network embedding methods, such as Isometric mapping (Isomap)[4], Locally Linear Embedding (LLE)[5], Laplacian eigenmaps[6], and Directed Graph Embedding (DGE)[7], is to find a low-dimensional vector space from which the main focus is then to reconstruct networks. However, these methods lack the preservation of the network structure, leading to a large amount of structural and nodal information loss, and thus resulting in an unacceptably low level of overall
More recently, the field of Natural Language Processing (NLP) has made great progress and the utilization of word vectors provides a good environment for network learning. Inspired by NLP frameworks, such as Word2Vec\cite{8-10}, network embedding methods based on word vector learning models, such as Skip-gram, improve the effectiveness of node representation. Some representative methods have been proposed, e.g., DeepWalk\cite{11}, Large-scale Information Network Embedding (LINE)\cite{12}, and Node2Vec\cite{13}. Among those approaches, LINE and Predictive Text Embedding (PTE)\cite{14} designed two objective functions attempting to address the local and global network structures, respectively, instructing researchers to consider network structures at both the local and global levels. Despite the desirable performance achieved by these network embedding approaches, all are shallow models, meaning that they are not effective at capturing high-dimensional nonlinear network structures.

Network embedding can be regarded as the process of converting the representation of nodes from an original high-dimensional space into a low-dimensional vector space. The main objective is to devise a mapping function between the two vector spaces. For highly nonlinear problems, deep neural networks can be useful, because deep learning is very good in the big data context at extracting higher-dimensional abstract features from data and applying the mapping function of high-dimensional data to low-dimensional features. Some embedding methods based on deep neural networks, such as Structural Deep Network Embedding (SDNE)\cite{15} and Stacked Denoising Autoencoders (SDAE)\cite{16}, have been proposed to address these problems.

Nevertheless, most of the previous work on embedding methods employs unsupervised schemas. Thus, although these methods are capable of capturing complex network structures for various tasks, they may not be as effective at some particular prediction tasks due to not utilizing the information on known labels. Node labels often store useful features of the vertices, knowledge of which increases the model’s ability at node discrimination. In addition, previous methods, such as SDNE, only consider a single node as the model’s input when using the deep model for network embedding; thus the learning process of adjacent nodes is relatively isolated and only limited features are ultimately obtained. Therefore, in this paper, we propose a semi-supervised deep network embedding approach called “Structural Labeled Locally Deep Nonlinear Embedding” (SLLDNE), based on the neighborhood structure and making use of the label information in the process of representation learning.

In this paper, in order to integrate the known labels information in a deep model, we train a Support Vector Machine (SVM)\cite{17} classifier which enlarges the distance between different nodes. Influenced by the SVM classifier, the representations of vertices are more distinct, which presents advantages for some particular prediction tasks.

In order to address the problems of isolation and limited features in deep models, and inspired by the LLE algorithm\cite{5}, we further integrate the representations of vertices’ neighborhoods into the learning progress. As a result, the learned representations of vertices store more features from their neighboring nodes.

Vertex classification is empirically tested by experiments on two real-world network datasets. The results indicate that the learned representations of our method are superior in performance compared with baselines methods, and demonstrate that our method can effectively integrate the information from known labels and neighboring nodes.

In summary, the main contributions of this work are as follows:

- We make use of a deep model and known label information simultaneously, especially in exploiting multiple node information through a deep model. We propose a semi-supervised deep network embedding approach called SLLDNE, based on the neighborhood structure and incorporating known label information to improve node discrimination.
- We utilize the representations of a vertex’s neighbors to improve the quality of the representations of vertices in the training process of the deep neural networks, reconstructing the embedding of nodes by combining the nodes and their neighbors in the current layer into the representation of nodes in the next layer. As a result, the representations of learned vertices store more features and achieve excellent performance in downstream tasks.
- We conduct an extensive experimental study over two real-world datasets. The experimental results demonstrate the effectiveness of SLLDNE, which outperforms state-of-the-art methods.
The rest of paper is organized as follows: in Section 2, we review some state-of-the-art methods; in Section 3, we introduce some of the basic concepts of network embedding and provide details of the SLLDNE model; we then report on our experiments and provide comprehensive experimental results in Section 4; Section 5 concludes.

2 Related Work

Network embedding aims to convert the network structure from an original high-dimensional space into a low-dimensional vector space. The learned low-dimensional representations can then be used in numerous network analysis tasks.

Earlier network embedding methods, mainly based on matrix factorization, were initially designed as dimensionality reduction techniques, aiming to reconstruct the entire network to the greatest extent. These approaches (e.g., Isomap[4], LLE[5], and Laplacian eigenmaps[6]) firstly construct the matrix of graph distances using the feature vectors of the vertices, and then obtain coordinate vectors in a low-dimensional space by embedding the affinity matrix. The disadvantage of Isomap is the time complexity of computing the shortest path between nodes. LLE proposes that each entry is able to be reconstituted from its neighbor nodes. Learned embeddings of LLE preserve the affinity between vertices. The intuition of Laplacian eigenmaps is that the adjacent nodes in a network are as close as possible after the reduction in dimensionality. Different from LLE, Laplacian eigenmaps select the eigenvector of the top-k minimum non-zero eigenvalues of the Laplace matrix as the final embeddings. However, these methods usually rely on calculating the eigenvalues of a specific matrix, resulting in high time complexity, and thus they are inadequate when it comes to dealing with large networks. Furthermore, the deficiency of structural information leads to unsatisfactory performance in downstream tasks.

In recent years, many network embedding models have been proposed to generate efficient vertex representations. With the great progress of NLP, a large number of representation learning models inspired by Skip-gram[9] were proposed, such as DeepWalk and Node2Vec. DeepWalk takes random walks on the network, regards the generated vertex sequences based on the previous random walks as a text, and trains the Skip-gram model to maximize the probability that a neighbor of a node will appear. Although DeepWalk is very effective, it is only proven to capture the global structure of networks. LINE designs two clear objective functions for local and global features of network structures. Through negative sampling, LINE has the ability to handle massive networks with nodes numbering in the millions and edges in the billions.

More recently, deep learning has made great progress. Deep learning captures abstract high-dimensional features by combining low-dimensional features to obtain an efficient representation of networks. This relatively short dense vector representation, called distributed feature representation, is naturally suitable for network embedding. There have been some deep models proposed for solving network embedding problems. Tian et al.[18] adopted a stack autoencoder[19] to handle the clustering problem. The authors of Ref. [20] set their focus on heterogeneous network embedding. SDNE[15] proposes to address several problems, and is able to obtain both the local and global features of a network structure. For the global structure, SDNE adopts a stack autoencoder to map input data into nonlinear implicit space through multilayer nonlinear mapping. The embeddings of nodes can then be obtained through the encoder process. As for the local network structure, SDNE references the idea of Laplacian eigenmaps to constrain the latent representations of vertices, making each pair of vertices more similar in the latent space. To solve the sparsity problem, SDNE introduces a penalty to the reconstruction error of the nonzero elements. The explicit loss function of SDNE is composed of a local structure component and a global structure component.

Although SDNE is capable of capturing high-dimensional nonlinear network information and works effectively with sparse networks, it nonetheless does have two limitations: (a) SDNE is an unsupervised model that cannot learn discriminative embeddings from a network structure with known labeled information; and (b) the input to SDNE is the adjacency vector of a single node, meaning that the information obtained is limited because the model can only learn embeddings of nodes through the structural features of an individual node of a deep neural network.

SDAE[16] aims to capture the structural information of a weighted network. The stacked denoising autoencoders are used to learn embeddings based on PageRank[21]. There are also some network embedding techniques which detect community structure through
modularity maximization\cite{22} and Multiple Non-negative Matrices Factorization (MNMF)\cite{23}. We will not go into these algorithms in detail because they are well outside of the focus of this paper.

There are few algorithms that make use of a deep model and known labeled information simultaneously, especially in exploiting information from multiple nodes. To fill this void, we propose our SLLDNE method to learn discriminative embeddings for nodes in networks.

3 Framework

In this section, we first give some definitions related to network embedding. Then, having identified the limitations of previous models, we introduce our proposed model SLLDNE. Finally, we deduce the objective function of the SLLDNE model and provide pseudo code along with a description of the related parameters.

3.1 Problem definition

Network embedding is a feature learning method for network data. Its definition is as follows:

**Definition 1 (Network Embedding)** Assume a network \(G = (V, E)\), where \(V\) is the set of nodes of \(G\), and \(E\) is the set of edges of \(G\). Network embedding aims at learning a mapping function \(f: v_i \rightarrow y_j \in \mathbb{R}^d\), where \(y_i\) is a representation vector in low-dimensional vector space whose dimension is \(d \ll |V|\). The goal of network embedding is to make \(y_i\) and \(y_j\) similar in the mapped low-dimensional space if \(v_i\) and \(v_j\) are close in the original network.

Network embedding converts the embedding of vertices from the original high-dimensional space into a low-dimensional vector space. Each node in the network can be represented by a vector in the low-dimensional space, and this representation vector can be applied directly to various downstream tasks. In order to make representations capable of preserving characteristics of the local and global network structures simultaneously, we introduce the first-order and second-order proximity, which capture the local and global network structures, respectively.

**Definition 2 (First-Order Proximity)** First-order proximity describes the similarity of each pair of vertices. For any pair of vertices (i.e., \(u\) and \(v\)), if there is an edge between these two nodes, then their first-order proximity is positive, otherwise it is zero.

First-order proximity defines the direct similarity of two nodes in the network. Intuitively, if two people are friends in a social network, they may share similar hobbies. However, in real world networks, similar nodes do not necessarily connect directly, so merely preserving first-order proximity is inadequate. Therefore, it is necessary to introduce a second order of proximity.

**Definition 3 (Second-Order Proximity)** A non-negative second-order proximity describes the similarity of the neighborhood structures of nodes. If two nodes \(u\) and \(v\) have many neighbors in common, their second-order proximity will be large. Otherwise, if there are no common neighbors between \(u\) and \(v\), the second-order proximity between the two nodes is zero.

Similarly, for example, if two people have mutual friends in a social network, they may have many similar interests. Integrating first-order and second-order proximity into network embedding can achieve preferable representations of vertices. Figure 1 shows a simple example of first-order and second-order proximity.

3.2 Structural labeled locally deep nonlinear embedding

In this paper, we propose a semi-supervised deep network embedding model called SLLDNE, as illustrated in Fig. 2. SLLDNE utilizes a deep neural network as its foundation and trains an SVM classifier to improve its discriminating ability for node embeddings.

To incorporate known labeled information into deep learning models, we train an SVM classifier for node classification in a deep neural network, through an approach known as transductive learning\cite{21}. In this way, SLLDNE ensures that the classification result of known labeled nodes can be incorporated into the process of network embedding. With learned

![Fig. 1 A simple example of network. From the point of view of first-order proximity, vertex 2 and vertex 3 should be closer in low-dimensional space because they connect directly to each other through an edge. In consideration of second-order proximity, vertex 1 and vertex 2 should be more similar in low-dimensional space as they share similar neighbors.](image)
embeddings, an optimal classification boundary is established by the SVM classifier and this boundary will instruct the model to adjust its output to obtain more desirable embeddings of nodes in turn. Therefore, the learned embeddings of vertices are discriminative.

In order to exploit information from multiple nodes in the deep neural network, SLLDNE takes the information of neighboring nodes into consideration while learning the network representation. In detail, SLLDNE adopts a deep neural network to extract high-dimensional nonlinear latent features, containing several nonlinear mapping layers. The original data can then be mapped into a low-dimensional space. Moreover, in order to capture more information from the deep model, SLLDNE reconstructs the embeddings of nodes by combining nodes and their neighbors in the current layer into the representation of nodes in the next layer. In Fig. 2, each dotted rectangle and the connecting arrows represent the training process for a single node. Unlike previous deep embedding methods, the features of each vertex in SLLDNE are obtained from the features of multiple nodes on the front layer. In this way, SLLDNE extracts more information.

After obtaining the embeddings of nodes through the deep learning model, the known labeled information is sent to the SVM classifier. Finally, SLLDNE simultaneously trains the SVM classifier and the deep neural network model.

Additionally, the first-order and second-order proximity can also be preserved during the reconstruction process. When reconstructing nodes, adjacent vertices interact with each other, capturing the first-order proximity of the network. Meanwhile, nodes with similar neighbors tend to structure similar embeddings, capturing the second-order proximity of the network. Specifically, when two nodes have the same neighbors, their embeddings will also be the same.

In comparison to SDNE, SLLDNE simultaneously integrates the first-order and second order proximity into the model, which means there is no need to calculate the reconstruction error or independently design an objective function for first-order proximity. Thus, the number of parameters and computational complexity are greatly reduced. In the next section, we will provide further details of SLLDNE and its loss function.

3.3 Loss function

As previously mentioned, the loss function of SLLDNE is inspired by the nonlinear reduction algorithm of LLE\cite{5}. The LLE algorithm posits that each vertex can be reconstructed by a linear weighted combination of its neighbors. The local reconstruction matrix of each node is calculated by the adjacency matrix, mapping the nodes from the original high-dimensional space into a low-dimensional space.

The objective function of LLE on the basis of locally linear reconstruction is as follows:

$$\phi(Y) = \sum_i |Y_i - \sum_j w_{ij} Y_j|^2$$  \hspace{1cm} (1)

where $Y_i$ and $Y_j$ are the embedding vectors of the nodes respectively, and $w_{ij}$ is the element of local reconstruction matrix $W$. If node $i$ is adjacent to node $j$, $w_{ij}$ is between 0 and 1, otherwise $w_{ij} = 0$. $\sum_j w_{ij} Y_j$ represents the process of locally linear reconstruction.

In the SLLDNE model, it is essential to extend the linear reconstruction from a shallow model to a deep neural network model. So, it is necessary to weigh each hidden layer as follows:

$$y_i^{k+1} = \sum_j y_j^k w_{ij}$$  \hspace{1cm} (2)

where $y_i$ and $y_j$ are the representations of node $i$ and node $j$ in the associated hidden layers, respectively, $w_{ij}$ is the element of reconstruction matrix $W$ of the network and $k$ denotes the $k$-th layer.

In order to avoid the hidden layers being too large in neural networks, the reconstruction result of each node is averaged through the vertex degree. In other words, each node is regarded as an average of the linear weight of its nearest neighbor. The revised function is shown...
as follows:

$$y_i^{k+1} = \frac{1}{d_i} \sum_j y_j^k w_{ij}$$  \hspace{1cm} (3)$$

where $d_i$ is the degree of node $i$.

The next step is to build the reconstruction matrix $W$, which consists of $w_{ij}$ in Eq. (3). Intuitively, the weight of a neighbor node of node $u$ should be in proportion to its importance to $u$. Therefore, $W$ can represent the importance of nodes. In practical applications, the representative evaluation metric of a node’s importance is PageRank\cite{21}, thus we can set the elements of $W$ to PageRank values. After that, the value of PageRank is normalized and set between 0 and 1 to get $PR_{\text{NORM}}$. Therefore, $w_{ij} = PR_{\text{NORM}}_{ij}$, where $w_{ij}$ is the element of $W$. Equation (3) can then be rephrased with the matrix as follows:

$$Y^{k+1} = D^{-1} A \bigodot W Y^k$$ \hspace{1cm} (4)$$

where $\bigodot$ means the Hadamard product and $A$ is the adjacency matrix of the network (especially, $A_{ii} = 1$).

In deep neural networks, models need to introduce multiple nonlinear activation functions to map the original data from a high-dimensional space to a low-dimensional space. Therefore, the latent representation of each hidden layer is shown as follows:

$$Y^{k+1} = \sigma(Y^k W^k)$$ \hspace{1cm} (5)$$

where $W^k$ is the $k$-th layer’s parameter matrix of the deep neural network and $\sigma(\cdot)$ is the nonlinear activation function. In this work, we have adopted ReLU ($\sigma(x) = \max(0, x)$) as the activation function. After introducing the activation function, the neural network model can be obtained as follows:

$$Y^{k+1} = \sigma((D^{-1} A \bigodot W Y^k)W^k)$$ \hspace{1cm} (6)$$

Because SVM is a machine learning model for small datasets with solid theoretical foundations, it is selected as the classifier to identify embeddings of known labeled nodes, which will adjust the output of the deep neural networks to minimize the SVM’s loss and thus obtain more desirable embeddings of nodes. Meanwhile, the embeddings of nodes without labels can also be obtained directly through the model. In real-world networks, the number of nodes with known labels may be only a small part of the total number of nodes, so it is appropriate to use SVM to differentiate known labeled nodes. Since SLLDNE does not require a separate design of the objective function for first-order and second-order proximity, the loss function of the model is totally made up of the classification error function of SVM. In multi-classification problems, the linear support vector machine can be converted to the following form:

$$L_{\text{SVM}} = \sum_i \sum_{k=1}^K \max(0, 1 - L_i^k W^s_z) + \gamma \|W^s\|^2$$ \hspace{1cm} (7)$$

where $L_i$ is the real label of the node and $W^s$ is the parameter matrix of the SVM classifier; $Z_i$ is the output of SLLDNE, which is obtained by the deep neural network, and $\gamma$ is the normalized parameter.

### 3.4 Optimization

The parameters in Eq. (7) include the weight matrix of the deep neural network $W^k$, where $k$ is the $k$-th layer hidden representations, and parameter matrix of the SVM classifier. We adopted the Adam (adaptive moment estimation) optimization strategy to optimize the model\cite{24}. Adam uses the first-order and second-order moment estimations of the model gradient to dynamically adjust the learning rate of each parameter in the model. Suppose the gradient of the loss function is $g_t = \nabla_{\theta_{t-1}} L_{\text{SVM}}(\theta_{t-1})$, where $\theta$ is the parameter.

The updating process of the model is as shown in Eqs. (8) – (12).

$$m_t = u \cdot m_{t-1} + (1 - u) \cdot g_t$$ \hspace{1cm} (8)$$

$$n_t = v \cdot n_{t-1} + (1 - v) \cdot g_t^2$$ \hspace{1cm} (9)$$

$$\hat{m}_t = \frac{m_t}{1 - \nu^t}$$ \hspace{1cm} (10)$$

$$\hat{n}_t = \frac{n_t}{1 - \nu^t}$$ \hspace{1cm} (11)$$

$$\nabla \theta_t = - \frac{\hat{m}_t}{\sqrt{\hat{n}_t} + \epsilon} \cdot \eta$$ \hspace{1cm} (12)$$

Here, $u$ and $v$ are momentum factors, $m_t$ and $n_t$ are first-order and second-order moment estimations of model gradient, respectively, and $\hat{m}_t$ and $\hat{n}_t$ are their corrections. Combining Eqs. (6), (7), and (12), the algorithm of model SLLDNE is presented as Algorithm 1.

- **Step 1**: Initialization (Lines 1 to 2). In order to obtain proper initial parameter values for the model, a Deep Belief Network (DBN)\cite{25} is used to pretrain the parameters. We then use the adjacency matrix ($S$) and the normalized PageRank ($PR_{\text{NORM}}$) as the input to the model.

- **Step 2**: Training model (Lines 3 to 8). According to Eqs. (6) and (7), we learn the hidden representation of nodes in each layer and calculate the error of the model. Then we use the Adam optimization strategy to update $\theta$ until the model converges.
4 Experiments

In this section, we perform our proposed method on two real-world datasets through vertex classification and compare the experimental results with several baselines, including some unsupervised and some semi-supervised algorithms. In addition, we research the parameter sensitivity of SLLDNE, the validity of the weights at the time of reconstruction, and the convergence trend of the error of the model.

4.1 Configuration and datasets

Experiments for this paper are run on the Windows 10 64-bit operating system, equipped with a 4-core 3.4 GHz Intel E3 1231 V3 CPU and 32 GB ddr3-1600 MHz memory space. The GPU used for the experiment is an Nvidia GTX1080TI with 11 GB of video memory space. All of the algorithms are implemented in python. The basic software configuration is shown in Table 1.

We employ the following two real world datasets for the vertex classification task.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Nodes</th>
<th>Edges</th>
<th>Labels</th>
</tr>
</thead>
<tbody>
<tr>
<td>CiteSeer</td>
<td>3312</td>
<td>4732</td>
<td>6</td>
</tr>
<tr>
<td>Cora</td>
<td>2708</td>
<td>5429</td>
<td>7</td>
</tr>
</tbody>
</table>

4.2 Baseline methods

To evaluate SLLDNE experimentally, we take the following five approaches as baseline methods for the performance of vertex classification on CiteSeer and Cora.

- DeepWalk[11]: This is an NLP method for learning network embeddings by using random walk and skip-gram. Staring from each node, repeated random walks are used as the text corpora. Accordingly, only second-order proximity is preserved.
- LINE[12]: This method designs the objective function for first-order and second-order proximity and then generates network embeddings by negative sampling and stochastic gradient descent.
- SDNE[15]: This adopts the deep model to capture the global network structure, and also uses Laplacian eigenmap to capture the local network structure, such that first-order and second-order proximity are combined.
- TLINE[27]: This is a method based on LINE and introducing the SVM objective function to make the known labels available. It is a semi-supervised method which optimizes the classifier and the component of LINE at the same time.
- MMDW[28]: This is a semi-supervised network embedding algorithm based on DeepWalk, for matrix factorization exploiting labeled information. MMDW optimizes the max-margin classifier and DeepWalk for matrix factorization.
Of these baseline methods, DeepWalk, LINE, and SDNE are unsupervised algorithms, while TLINE and MMDW are considered semi-supervised models. Only SDNE employs a deep model.

4.3 Evaluation metrics and parameter settings
In our experiments, Micro-F1 was used as the evaluation metric for vertex classification. To be specific, for a label \( a \), we assume that \( TP(a) \), \( FP(a) \), and \( FN(a) \) are True Positive (TP), False Positive (FP), and False Negative (FN) instances of classification results, respectively. Supposing that \( C \) is the collection of labels, Micro-F1 can be defined as follows:

\[
\pi = \frac{\sum_{a \in C} TP(a)}{\sum_{a \in C} (TP(a) + FP(a))} \tag{13}
\]

\[
\rho = \frac{\sum_{a \in C} TP(a)}{\sum_{a \in C} (TP(a) + FN(a))} \tag{14}
\]

\[
\text{Micro-F1} = \frac{2 \times \pi \times \rho}{\pi + \rho} \tag{15}
\]

Our proposed SLLDNE adopts a multilayer neural network. Due to the similarity between the dimensions of the two experimental datasets, the neural network of each of our datasets has two hidden layers. The dimensions of the hidden layers are 300 and 100, respectively. According to the experiment results, more layers do not improve performance, rather making it worse.

For the baseline methods applied on the same datasets, the same hyper-parameters are set with the corresponding references. Conversely, the hyper-parameters are adjusted by grid searching on the validation set. We choose the optimal values of parameters for baseline methods. For DeepWalk, window size is set to 10, walk lengths is set to 40, and walks of each node is set to 40. The representation dimension of DeepWalk is 128. We set parameters in LINE and TLINE as follows: learning rate to 0.025, the number of samples to 1 million, and the representation dimension to 200. According to Ref. [12], we concatenate 1- and 2-step representations so as to form the final embeddings. The hyper-parameters of TLINE, \( \beta \) and \( \gamma \), are set to 0.5 and 0.02, respectively, and for MMDW, we choose the best performance for variant \( \eta \).

4.4 Experimental results and analysis of multi-label classification
In this section, to facilitate the comparison between SLLDNE and the baselines, we use the same datasets and experimental procedure.

We randomly sample 10% to 90% of the labeled nodes as training samples, while the remaining nodes are used as test samples to evaluate performance. For all models, we adopt the LibLinear package\cite{29} for classification. In order to avoid overfitting of the neural network, we use early stopping in our experiments with a window size of 20.

The results of multi-label classification are shown in Tables 3 and 4, from which we can draw the following conclusions:

1. The proposed SLLDNE achieves a significant and consistent improvement over all baseline methods on both datasets under different training ratios. Specifically, SLLDNE shows a nearly 2.37% average improvement on CiteSeer, and a 1.25% average improvement on Cora. These performance figures prove that SLLDNE is a competitive method.

2. The performance of semi-supervised models shows considerable improvements in performance (by 10% on average) over unsupervised models that ignore known labeled information. We notice that, among these unsupervised methods, SDNE does not perform well on Cora; this may be related to the characteristics of the datasets, being that CiteSeer is sparser than Cora. SDNE is a network embedding method with a deep

<table>
<thead>
<tr>
<th>Labeled nodes (%)</th>
<th>DeepWalk</th>
<th>LINE</th>
<th>SDNE</th>
<th>TLINE</th>
<th>MMDW</th>
<th>SLLDNE</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>49.09</td>
<td>39.82</td>
<td>52.32</td>
<td>49.33</td>
<td>56.22</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>55.96</td>
<td>46.83</td>
<td>58.08</td>
<td>55.91</td>
<td>61.54</td>
<td>61.75</td>
</tr>
<tr>
<td>30</td>
<td>60.65</td>
<td>49.02</td>
<td>60.44</td>
<td>60.38</td>
<td>63.36</td>
<td>64.25</td>
</tr>
<tr>
<td>40</td>
<td>63.97</td>
<td>50.65</td>
<td>64.92</td>
<td>63.66</td>
<td>65.18</td>
<td>68.41</td>
</tr>
<tr>
<td>50</td>
<td>64.52</td>
<td>53.77</td>
<td>66.61</td>
<td>65.55</td>
<td>66.93</td>
<td>69.99</td>
</tr>
<tr>
<td>60</td>
<td>67.29</td>
<td>54.2</td>
<td>68.23</td>
<td>67.71</td>
<td>69.52</td>
<td>71.77</td>
</tr>
<tr>
<td>70</td>
<td>66.8</td>
<td>58.94</td>
<td>68.05</td>
<td>67.54</td>
<td>70.47</td>
<td>73.34</td>
</tr>
<tr>
<td>80</td>
<td>66.82</td>
<td>59.77</td>
<td>68.34</td>
<td>67.06</td>
<td>70.87</td>
<td>74.96</td>
</tr>
<tr>
<td>90</td>
<td>63.91</td>
<td>59.37</td>
<td>66.76</td>
<td>63.58</td>
<td>70.95</td>
<td>77.41</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Labeled nodes (%)</th>
<th>DeepWalk</th>
<th>LINE</th>
<th>SDNE</th>
<th>TLINE</th>
<th>MMDW</th>
<th>SLLDNE</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>55.96</td>
<td>46.83</td>
<td>58.08</td>
<td>55.91</td>
<td>61.54</td>
<td>61.75</td>
</tr>
<tr>
<td>20</td>
<td>60.65</td>
<td>49.02</td>
<td>60.44</td>
<td>60.38</td>
<td>63.36</td>
<td>64.25</td>
</tr>
<tr>
<td>30</td>
<td>63.97</td>
<td>50.65</td>
<td>64.92</td>
<td>63.66</td>
<td>65.18</td>
<td>68.41</td>
</tr>
<tr>
<td>40</td>
<td>64.52</td>
<td>53.77</td>
<td>66.61</td>
<td>65.55</td>
<td>66.93</td>
<td>69.99</td>
</tr>
<tr>
<td>50</td>
<td>67.29</td>
<td>54.2</td>
<td>68.23</td>
<td>67.71</td>
<td>69.52</td>
<td>71.77</td>
</tr>
<tr>
<td>60</td>
<td>66.8</td>
<td>58.94</td>
<td>68.05</td>
<td>67.54</td>
<td>70.47</td>
<td>73.34</td>
</tr>
<tr>
<td>70</td>
<td>66.82</td>
<td>59.77</td>
<td>68.34</td>
<td>67.06</td>
<td>70.87</td>
<td>74.96</td>
</tr>
<tr>
<td>80</td>
<td>63.91</td>
<td>59.37</td>
<td>66.76</td>
<td>63.58</td>
<td>70.95</td>
<td>77.41</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Labeled nodes (%)</th>
<th>DeepWalk</th>
<th>LINE</th>
<th>SDNE</th>
<th>TLINE</th>
<th>MMDW</th>
<th>SLLDNE</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>67.2</td>
<td>65.13</td>
<td>69.4</td>
<td>67.45</td>
<td>74.94</td>
<td>77.28</td>
</tr>
<tr>
<td>20</td>
<td>72.53</td>
<td>70.17</td>
<td>74.15</td>
<td>71.34</td>
<td>80.83</td>
<td>82.37</td>
</tr>
<tr>
<td>30</td>
<td>75.87</td>
<td>72.2</td>
<td>74.67</td>
<td>73.37</td>
<td>82.83</td>
<td>83.68</td>
</tr>
<tr>
<td>40</td>
<td>77.64</td>
<td>72.92</td>
<td>75.14</td>
<td>73.96</td>
<td>83.68</td>
<td>85.11</td>
</tr>
<tr>
<td>50</td>
<td>80.35</td>
<td>73.45</td>
<td>76.1</td>
<td>76.31</td>
<td>84.71</td>
<td>85.82</td>
</tr>
<tr>
<td>60</td>
<td>81.47</td>
<td>75.67</td>
<td>76.4</td>
<td>77.96</td>
<td>85.51</td>
<td>86.44</td>
</tr>
<tr>
<td>70</td>
<td>83.31</td>
<td>75.25</td>
<td>78</td>
<td>77.37</td>
<td>87.01</td>
<td>87.45</td>
</tr>
<tr>
<td>80</td>
<td>84.58</td>
<td>76.78</td>
<td>78.75</td>
<td>78.11</td>
<td>87.27</td>
<td>89.85</td>
</tr>
<tr>
<td>90</td>
<td>85.61</td>
<td>79.34</td>
<td>81.19</td>
<td>81.82</td>
<td>88.19</td>
<td>90.03</td>
</tr>
</tbody>
</table>
learning model, with a desirable ability to establish representation that may benefit from dealing with sparse data.

4.5 Effect of dimensionality

Within this section, we investigate how dimensions of the output vectors of nodes affect performance. Figures 3 and 4 show Micro-F1 w.r.t. for different dimensions on the CiteSeer and Cora datasets, respectively.

In Fig. 3, the curve shows Micro-F1 scores of the embedding dimensions from 10 to 130. Since the dimensionality of embeddings is relatively small, the performance improves as the dimensionality increases, indicating that more hidden units extract more effective information and are more conducive for vertices’ classification. However, the Micro-F1 score tends to become stable and even drop slightly as the dimensionality of embeddings increases further. This may due to an inappropriate dimensionality causing the learned latent embeddings to capture noise and thus weaken the accuracy of classification.

Therefore, it is important to select an appropriate dimensionality for implicit space. In the experiments of multi-label classification, we compare the performance between different dimensions on two datasets and then set the dimensionality of the latent space to 100 in order to elicit the best performance.

4.6 Validity of the weights in reconstruction

In Eq. (1), $w_{ij}$ is an element of local reconstruction matrix $W$ and the objective of $W$ is to minimize the reconstruction errors and characterize essential geometric properties of neighbors. Thereby, nodes are reconstructed from their neighbors. In this study, we adopt a deep model which also contains the process of reconstruction. Equation (6) can be converted into the following form:

$$Y_{k}^{+1} = \sigma((D^{-1} A \odot W)(Y_{k}^{k}W_{k}^{k}))$$  \hspace{1cm} (16)

where $Y_{k}^{k}W_{k}^{k}$ can be viewed as a weighted process. SLLDNE does not need to calculate the reconstruction error, as the LLE algorithm does, because its primarily objective is to classify vertices. Besides $Y_{k}^{k}$, Eq. (6) also includes $D^{-1}$ and $W$ which are equivalent to weighting the adjacency matrix. Consequently, SLLDNE reconstructs neighbor nodes by means of a deep model, $D^{-1}$ and $W$.

In order to prove the validity of the weights defined by $D^{-1}$ and $W$, we compare our proposed method with and without a weighted reconstruction process. The results on CiteSeer and Cora are shown in Figs. 5 and 6, respectively.

In Figs. 5 and 6, the curve marked with triangles represents the Micro-F1 scores of weighted SLLDNE, whereas the curve marked with squares represents the Micro-F1 scores of unweighted SLLDNE. We calculate the average improvements in Micro-F1 scores from unweighted to weighted as 0.84% on CiteSeer and 1.08% on Cora. That is to say, SLLDNE with weighted
reconstruction by $D^{-1}$ and $W$ performs better than unweighted SLLDNE. By analyzing the weighting’s impact on performance, we have successfully validated the effectiveness of reconstruction weights. Especially on the Cora dataset, the improvement in the Micro-F1 score of the weighted model is significant. Considering that the network of Cora is denser than that of CiteSeer, we can conclude that the weighted SLLDNE is more suitable for dense networks.

4.7 Convergence

During the training process of SLLDNE, the deep neural network and the SVM classifier are simultaneously optimized.

In order to avoid overfitting, we use early stopping in our experiments, with a window size of 20. In Figs. 7 and 8, we show the convergence trend of the error when the model is trained through iterations.

In Figs. 7 and 8, the curve marked with triangles represents the training error, while the curve marked with squares represents the generalization error. We notice that our proposed method SLLDNE reaches the lowest generalization error after 200 iterations, whereas the training error continues to decrease to zero. This means that the model has already started to overfit. Therefore, we perform early stopping with a window to avoid overfitting as the number of iterations increases, while also reducing the time complexity of the model and sequentially improving the efficiency of our experiments.

5 Conclusion

In this paper, we proposed an effective semi-supervised network embedding model called SLLDNE, based on the neighborhood structure and deep model. In order to improve the ability to discriminate node embeddings, we set up an SVM classifier as the supervised component to incorporate known labeled information. To further capture highly nonlinear network structures, we exploited a deep neural networks model, which has a multilayer nonlinear function. By combining the embeddings of vertices and their neighbors, we enable the learned representations to store more features. Experimental results of vertex classification on two real-world networks prove the effectiveness of SLLDNE and the substantial gains achieved compared to state-of-the-art methods.

In future research, we will focus on (1) learning embeddings for new nodes with no links to existing vertices in dynamical networks; and (2) exploring methods to integrate and balance network structure, labels, and node content information in network embedding simultaneously. Besides network structure and labels, many real-world networks are accompanied by other rich content information, such as node content, which is of prime importance.

Acknowledgment

This work was supported by the National Natural Science Foundation of China (No. 61370207) and Collaborative Innovation Center of Novel Software Technology and Industrialization.

References


Wenmao Wu is currently a master student in School of Computer Science and Engineering of Southeast University. He is also a research staff member in MOE Key Laboratory of Computer Network and Information Integration. He received the BSc degree from Xidian University in 2015. His research interests in network embedding, big data analysis, and data mining.

Zhizhou Yu is currently a master student in School of Computer Science and Engineering of Southeast University. He is also a research staff member in MOE Key Laboratory of Computer Network and Information Integration. He received the BSc degree from Southwest Jiaotong University in 2017. His research interests in network embedding and machine learning.
**Jieyue He** received the BSc and MSc degrees from Nanjing University, China, in 1985 and 1988, respectively and the PhD degree from Southeast University, China in 2006. She is currently a professor of the School of Computer Science and Engineering, Southeast University, China. Her current research interests include bioinformatics, data mining, machine learning, and big data.