Depth-based Subgraph Convolutional Auto-Encoder for Network Representation Learning

Zhihong Zhang[a], Dongdong Chen[a], Zeli Wang[b,*], Heng Li[b], Lu Bai[c], Edwin R. Hancock[d]

[a] *Xiamen University, Xiamen, China*
[b] *The Hong Kong Polytechnic University, Hongkong, China*
[c] *Central University of Finance and Economics, Beijing, China*
[d] *University of York, York, UK*

Abstract

Network representation learning (NRL) aims to map vertices of a network into a low-dimensional space which preserves the network structure and its inherent properties. Most existing methods for network representation adopt shallow models which have relatively limited capacity to capture highly non-linear network structures, resulting in sub-optimal network representations. Therefore, it is nontrivial to explore how to effectively capture highly non-linear network structure and preserve the global and local structure in NRL. To solve this problem, in this paper we propose a new graph convolutional autoencoder architecture based on a depth-based representation of graph structure, referred to as the depth-based subgraph convolutional autoencoder (DS-CAE), which integrates both the global topological and local connectivity structures within a graph. Our idea is to first decompose a graph into a family of K-layer expansion subgraphs rooted at each vertex aimed at better capturing long-range vertex inter-dependencies. Then a set of convolution filters slide over the entire sets of subgraphs of a vertex to extract the local structural connectivity information. This is analogous to the standard convolution operation on grid data. In contrast to most existing models for unsupervised learning on graph-structured data, our model can capture highly non-linear structure by simultaneously integrating node features and network structure into network representation learning. This significantly

*Corresponding author: Zeli Wang
Email address: `jerry.wang@connect.polyu.hk.` (Zeli Wang)

improves the predictive performance on a number of benchmark datasets.

Keywords: Network Representation Learning, Graph Convolutional Neural Network, Node Classification

1. Introduction

Many naturally occurring problems can be represented using an underlying graph or network structure, such as natural language processing [1], computer vision [2], or social network analysis [3]. It is therefore crucial to accurately extract useful information from a network. One promising strategy is to map vertices of a network into a low-dimensional space, *i.e.* learn vector representations for each vertex, with the goal of encoding meaningful information conveyed by the graph in the learned embedding space. As a result, the learned representation can be used directly with existing machine learning methods to perform various network analysis tasks, such as vertex classification [3] and clustering [4].

A successful network representation learning model should exhibit two desirable properties: a) a High non-linearity: As [5] stated, the structure and inherent properties of a network are highly non-linear. b) Structure-preserving [6]: The learned representation should preserve both the global topological arrangement information (*e.g.* long-range dependencies) and the local connectivity structure of a network.

Traditional methods for graph dimensionality reduction such as Local Linear Embedding (LLE) [7], Laplacian eigenmap [8] and IsoMAP [9, 10] are based on a graph affinity matrix decomposition, which treats the leading eigenvectors as a representation. However, the time complexity of these methods is at least quadratic in the number of graph nodes, which makes them not applicable to large-scale networks.

Motivated by the Skip-Gram model [1], which was originally used for word representation learning in natural language processing, there is recent work which attempts to embed very large networks using random walk and matrix factorization based learning objective functions [11, 12, 13, 14, 15, 16, 17]. DeepWalk [12] is one of the pioneering works for learning node representations in networks. By regarding the vertices as words, DeepWalk [12] first performs random walks over a network to generate vertex sequences. Then, these linear sequences of vertices are treated as sentences and fed into the Skip-Gram model to learn vertex representations. This idea has been developed by numerous subsequent methods [14, 11, 13]. The resulting random-walk approaches have been extensively verified on a variety of classification and visualization tasks involving large-scale networks.

Despite the success of these network embedding approaches, the methods have three common weaknesses. Firstly, they mainly focus on preserving the local structure of a network. Therefore, what they learn mostly depends on what the random walks can capture on sample. DeepWalk [12] and node2vec [11] adopt short random walks to explore the local neighborhoods of nodes, while LINE [13] is concerned with even more local relationships (nodes at most two hops away). This focus on local structure implicitly ignores long-distance global relationships, and the learned representations can fail to capture the important global structural properties of graphs. Secondly, they all adopt shallow models which have a relatively limited capacity to capture highly non-linear structures in the underlying network. Finally, the above methods all consider only network structure, e.g., the links between nodes, but ignore the node and edge attribute information, which are significant features of the network and could potentially benefit network representation learning.

Inspired by the impressive success of deep learning on processing regular grid data, a substantial interest has arisen in the generalization of deep learning models to graph-structured data [18]. Several convolutional neural network architectures for learning over graphs have been proposed (e.g., [19, 20, 21, 22, 23]). For example, QS-CNN [23] proposes a supervised graph convolutional method that performs a quantum walk on the graph in order to determine the nodes belonging of each receptive field used for convolution in a CNN. However, the network architectures and loss functions of these methods are designed for node or whole-graph classification, and very little work investigates how to extend deep learning models, and in particular auto-encoders, to network representation learning. To our knowledge, only the work of [24, 15] uses a basic autoencoder to extract complex features and model the non-linear structure of a network. The input of SDNE [15] is a vector of node adjacency matrix elements which represents the neighborhood of each vertex. The encoder generates the network representation and the decoder reconstructs the input. However, the structure of SDNE can only encode the neighborhood information, rather than both neighborhood and node information. Moreover, SDNE is a stack of multiple fully connected layers rather than convolutional layers, leading to too many parameters (neurons).

To address the above issues, we propose a depth-based subgraph convolutional autoencoder (DS-CAE) for network representation learning by comprehensively modeling both node content information and network structure. Specifically, we commence by establishing a family of K-layer expansion subgraphs for each vertex of a graph using graph labeling procedures (e.g. node degree, PageRank), aimed at better capturing long-range vertex interdependencies contained within a graph. We then design a set of fixed-size convolution filters and integrate them with these subgraphs (as depicted in Figure 3). The idea is to apply convolution filters which slide over the entire sets of subgraphs of a vertex to extract local features in a manner analogous to the standard convolution operation on grid data. In particular, the convolution operation captures the local structural information within the graph, and has the weight sharing property among different positions of subgraph. The main contributions of our work are summarized as follows:

- To our knowledge, DS-CAE is the first convolution-based deep learning method for unsupervised network representation learning. Similar to the convolution for image processing, a subgraph-based convolution operation is proposed to scan a tree which is extracted from various graph structures.

- In contrast to most existing models [12, 13] for unsupervised learning on graph-structured data, DS-CAE can capture highly non-linear network structure by simultaneously integrating raw node information and network structure into network representation learning.

- Due to the *deep* representation, DS-CAE can map graphs to highly non-linear deeply learned spaces to effectively preserve both the local and global information in original space. The code will be made available for public use.

2. Related Work

Our algorithm is an unsupervised approach to learning over graphs. It represents an advance in applying a new depth-based subgraph convolutional autoencoder to graph-structured data for node embedding.

Network embedding. Many NRL models which use random walks and matrix factorization have been proposed [25, 14, 15, 16, 26]. For example, DeepWalk [12] performs truncated random walks on networks to transform unweighted graph structural information into a large collection of linear structures represented by vertex sequences. These linear vertex sequences are then treated as sentences and fed to the Skip-Gram model to learn vertex representations. This idea was later extended by node2vec [11], which introduced hyperparameters to DeepWalk that tune the depth and breadth of the random walk. LINE [13] formulates a clearer objective function which directly depends on the structure of the graph instead of relying on random walks for capturing the structure of the input graphs. This objective function is based on the definition of proximity in graphs, which includes first-order and second-order local relational network information. GraRep [14] uses higher order proximities to learn multi-scale representations for a graph. It defines different loss functions to capture different k-order proximities and finally concatenates the representation of the different scales together. In addition to the above NRL methods that focus on network topology, significant effort has also been explored aimed at incorporating user-generated contents (e.g. text and label information) into NRL to learn more informative network representations [27]. Text-associated DeepWalk (TADW) [28] improves the matrix factorization underpinning DeepWalk by considering both network structure and text features. Max-margin DeepWalk (MMDW) [29] on the other hands integrates vertex labeling information into network representation learning and learns discriminative network representations for network vertices using max-margin constraints between vertices carrying different labels.

Deep Neural Networks. Deeply learned features for NRL give promising performance when compared to traditional methods [30]. This is due to their highly nonlinear modeling capacity and flexible scalability. The autoencoder is one well-known deep learning based NRL method. In a manner similar to other deep learning architectures, the auto-encoder is an end-to-end learning architecture stacked with multiple layers of learning nodes. An auto-encoder consists of an encoder and a decoder. The encoder uses raw data as input and produces a representation as output. The decoder uses the learned features from the encoder as input and reconstructs the original input as output. The original auto-encoder is stacked with multiple fully connected layers. To effectively and efficiently process image data, the convolutional auto-encoder (CAE) [31] is stacked with convolutional layers. Compared with the original auto-encoder, CAE can capture topological information residing in images and can reduce the number of network parameters. In graph signal processing, significant interest has arisen in the generalization of CAE to graph data [18]. This problem is not as straightforward since the three basic operations of a) convolution, b) pooling and c) weight-sharing are only designed for regular grids. These three points make the adaptation of CAE to graphs both theoretically and implementation-wise challenging.

3. Preliminary Concepts

In this section, we introduce some preliminary concepts that will be used for developing the work presented in this paper.

3.1. Graphs

A graph G is a pair of sets (V, E), where $V = \{v_1, ..., v_n\}$ is the set of vertices and $E \subseteq V \times V$ is the set of edges, formed by pairs of vertices. Each graph can be represented by an adjacency matrix A of size $n \times n$, where n is the number of vertices in G. In particular, the element $a_{i,j} = 1$ if there is an edge between vertex v_i and vertex v_j, *i.e.* v_i and v_j are adjacent, and $a_{i,j} = 0$ otherwise. A walk is a sequence of edges and vertices, where the endpoint of each edge are adjacent. A path is a walk in which all vertices are distinct (except possibly the first and last). We denote $d(v_i, v_j)$ as the length of the shortest path between vertex v_i and vertex v_j, and denote k-hop(v_i) as the k-neighborhoods of vertex v_i, *i.e.* $d(v_i, v_j) = k$ for any vertices v_j of k-hop(v_i).

3.2. Graph Labeling

A graph labeling is an assignment of integers to vertices subject to certain conditions. Formally, given a graph $G = (V, E)$, a graph labeling Λ is a function Λ: $V \rightarrow S$ from the vertices V to an ordered set S. A ranking is a function R: $V \rightarrow \{1, ..., |V|\}$. Every labeling induces a ranking R with $R(u) < R(v)$ if and only if $\Lambda(u) > \Lambda(v)$. Examples of graph labeling procedures are those based on sorting node degree [32], PageRank [33], eigenvector centrality [34] and other procedures commonly used in complex network analysis [35]. In this paper, our graph labeling procedure is the commonly used PageRank algorithm [33] for node classification. When we select a node to construct a tree in Section 4.1, the pagerank value of the node is regarded as a measure of the importance for that node. When two nodes are in the same rank order because they have the same pagerank value relative to the target node, we take the effect of two nodes on the target node to be almost the same in terms of spatial structure.

3.3. PageRank Algorithm

PageRank is a link analysis algorithm for websites represented by a directed graph first proposed by Brin and Page [36]. It is used to determine a rough estimate of the importance and each website. The algorithm is based on both quantitative and qualitative assumptions. The quantitative assumption is that in the web graph representation, the more links a page receives from other pages, the more important that page. The qualitative assumption is concerns the link structure to high-quality pages. The more quality pages that point to a page, the more important that page. Based on these two assumptions, the PageRank algorithm initially gives each page the same importance score, and then the current score of each page is equally assigned to each out-chain of that page. In this way, each page receives a new score by summing all the weights passed to it from the incident in-chains. These score values are recursively and iteratively updated until convergence to give the final PageRank value. The PageRank algorithm can be applied to any collection of entities represented by a directed graph.

4. Proposed DS-CAE Model

4.1. The Depth-Based Representation for a Graph

In order to exploit topological information concerning the arrangement of vertices and edges in a graph, we develop a K-layer depth-based representation for a graph from each vertex. Concretely, the representation is constituted of two steps: (1) construct an m-ary tree from each vertex using the graph grafting and graph pruning algorithm; (2) the leaf nodes of the i-level m-ary tree are further replaced by their own neighborhood m-ary trees. Hence a K-level m-ary tree is recursively constructed for each vertex.

For each vertex, a receptive field of the same size should be constructed. However, the sizes of the different node's 1-hop neighborhood are different. We therefore propose to use graph grafting and graph pruning to normalize each node's neighborhood subgraph so that it is an m-ary tree. The normalization includes cropping of excess nodes if the neighborhood is larger than required and padding with dummy nodes if it is smaller than required.

Graph Grafting: For node v whose 1-hop size is less than m, we use graph grafting to choose nodes from node k-hop(v) (k >=2) to fill node 1-hop(v). As shown in Figure 1, besides the pink vertex itself, we still need to incorporate m = 1 vertex into the receptive field from node k-hop(v) (k >=2). We use nodes from node 2-hop(v), if number of nodes in 2-hop are

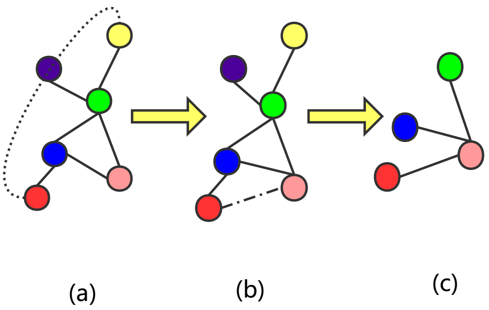


Figure 1: An illustrative example of graph grafting. Vertices connected in dotted line are the pink vertex's 2-hop, the red vertex has a higher pagerank value than other vertices of pink vertex's 2-hop.

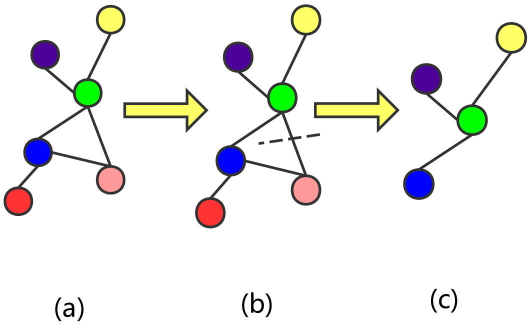


Figure 2: An illustrative example of graph pruning. The pink vertex has a smaller pagerank value than other vertices of green vertex's 1-hop.

not sufficient, then we choose nodes from 3-hop and so on instead. If there exist more nodes than required, we use the m nodes with highest pagerank values. In this way, the neighborhood subgraph consisting of exactly m vertices is extracted and normalized as an m-ary tree. Then we rank the leaf nodes of the m-ary tree according to their pagerank values.

Graph Pruning: For node v whose 1-hop size is greater than m, we use graph pruning to select nodes from node 1-hop(v). As shown in Figure 2, besides the green vertex itself, we need to remove one node so that only $m = 3$ vertices is preserved, we remove nodes with smaller pagerank values. In this way, the neighborhood subgraph consisting of exactly m vertices is extracted and normalized as an m-ary tree. We then rank the leaf nodes of the m-ary tree according to their PageRank values.

Mapping Graph to Tree: With the help of graph grafting and graph pruning, we normalize each node's subgraph as an m-ary tree. The leaf nodes of each m-ary tree are replaced by their own neighborhood m-ary trees. In this way, a K-level m-ary tree is recursively constructed for each vertex. Algorithm 1 gives the pseudocode for the Mapping Graph to Tree algorithm.

Algorithm 1: Mapping Graph to Tree

```
Input: graph, receptive field size $m + 1$, pagerank algorithm, graph
       grafting, graph pruning, the depth $K$
Output: normalized neighborhood graph ($K$-level $m$-ary tree) for
        each vertex
initialization;
compute pagerank value for each vertex;
construct an $m$-ary tree with each vertex by the graph grafting and
graph pruning algorithm;
for $i = 2, i \leq K - 1$ do
    The leaf nodes of the $i$-level $m$-ary tree are further replaced by
    their own neighborhood $m$-ary trees;
end
return $K$-level $m$-ary tree for each vertex;
```

4.2. DS-CAE Model for Network Representation Learning

We first list the notation used in the paper in Table.1 and a hat ˆ above the parameters represents the parameters of the decoder. Then, we present our depth-based subgraph convolution autoencoder for the K-level m-ary tree. Figure 3 shows an example of the whole process with $K = 3$ and $m = 3$. The encoder consists of multiple non-linear mappings that transform the input data into the representation space. The decoder also consists of multiple non-linear mappings which transform the representation into reconstruction space.

Table 1: Important notations and their descriptions.

Symbol	Dimensions	Definition
$node(p, q)$	Scalar	the q-th node in level p of the K-level m-ary tree
σ	Scalar	the activation function
$f_l, \hat{f_l}$	Scalar	the number of filters in layer l
n	Scalar	number of vertices
$\odot$	Scalar	element-wise multiplication
$X^{l,t}, \hat{X}^{l,t}$	$1 * (1 + m)^{K-l-1}$	the hidden representation for the t-th feature channel in layer l
$Y^{l-1,r}, \hat{Y}^{l-1,r}$	$1 * (1 + m)^{K-l}$	the r-th feature of the input data in layer l
$X^{l,t}_{p,q}, \hat{X}^{l,t}_{p,q}$	Scalar	the t-th feature of node(p,q) in layer l
$Y^{l-1,r}_{p,q}, \hat{Y}^{l-1,r}_{p,q}$	$1 * (1 + m)$	the r-th feature of node $(p, q)'s$ receptive field in layer l
$W^{l,t,r}$ $\hat{W}^{l,t,r}$	$1 * (1 + m)$ $1 * 1$	the filter mapping from the r-th feature to the t-th feature in layer l
$b^{l,t}$, $\hat{b}^{l,t}$	Scalar	the bias of the t-th filter in layer l
d_i, $\hat{d}_i$	$1 * [(1 + m)^{K-1}$ $* feature\_channel]$	input and reconstructed data of subtree with node i as the root node
$D, \hat{D}$	$N * [(1 + m)^{K-1}$ $* feature\_channel]$	$D = \{d_i\}^n_{i=1}, \hat{D} = \{\hat{d}_i\}^n_{i=1}$
r_i	Vector $(1 * x)$	the representation of vertex i
R	$N * x$	$R = \{r_i\}^n_{i=1}$
$A = \{a_{i,j}\}^n_{i,j=1}$	$N * N$	the adjacency matrix

When convolutional auto-encoders are applied to images, a square grid is moved over each image with a particular step size to extract structural features as the hidden representations of each layer. More precisely, a receptive field in the preceding layer becomes a neuron in the next layer after a con-

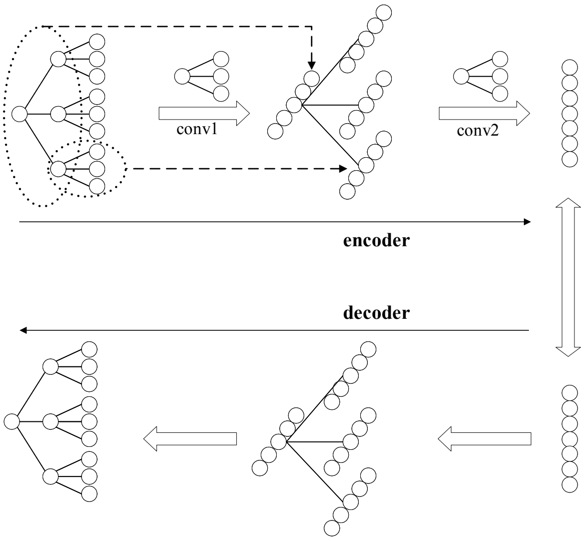


Figure 3: The framework of the unsupervised network representation learning with depth-based subgraph convolutional auto-encoder. Here, we give an example of 3-layer 3-ary tree, and design the filter as a subtree with 3-ary. The tree will be reduced by one layer with each layer of convolution and finally achieve a representation. The number of channels in a tree is determined by the number of filters in the upper convolution. The decoder is the inverse process of the encoder.

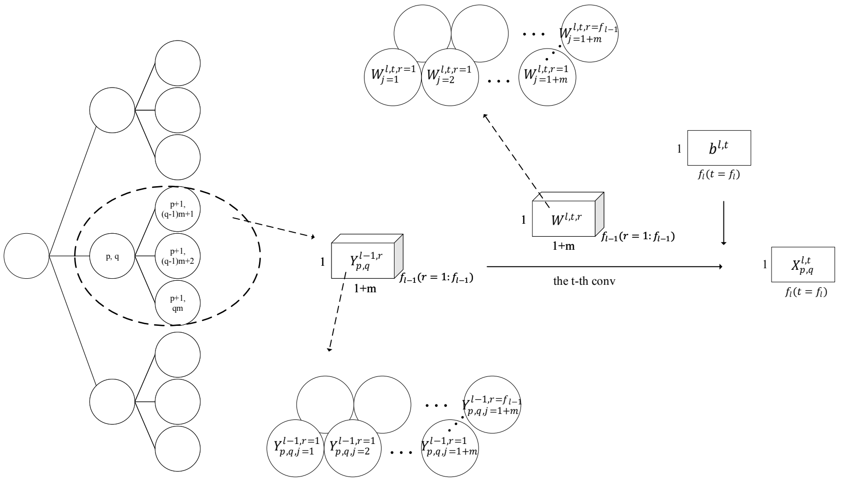


Figure 4: This figure illustrated the convolution process by channel. The $Y_{p,q,j}^{l-1,r}$ is the r-th feature of node $(p,q)^{'}s$ receptive field in layer l, which multiplies with the corresponding weight in filters, then we sum up these products and add the bias, the t-th feature of node(p,q) in layer l can be achieved.

volution operation. In this way, the local structural features of images are well captured. By generalizing convolutional auto-encoders to the K-level m-ary tree obtained in the previous step of graph normalization, we scan a subgraph-based window (m-ary tree) along the tree to extract structural features as the hidden representation of each layer. Given the input x_i, a K-level m-ary tree with the root vertex i, the hidden representation $X_{p,q}^{l,t}$ for node (p,q), feature t and layer l is given by

$$X_{p,q}^{l,t} = \sigma(\sum_{r=1}^{f_{l-1}}(\sum_{j=1}^{m+1} W_j^{l,t,r} Y_{p,q,j}^{l-1,r}) + b^{l,t}) \qquad p \leq K - l + 1 \qquad (1)$$

where p is the p-level of the K-level m-ary tree, q is the q-th node, node (p,q) is the q-th node in level p of K-level m-ary tree, f_{l-1} is the number of filters in layer $l-1$, σ is the activation function, $X_{p,q}^{l,t}$ is the t-th feature of node(p,q) in layer l, $W^{l,t,r}$ is the filter mapping from the r-th feature to the

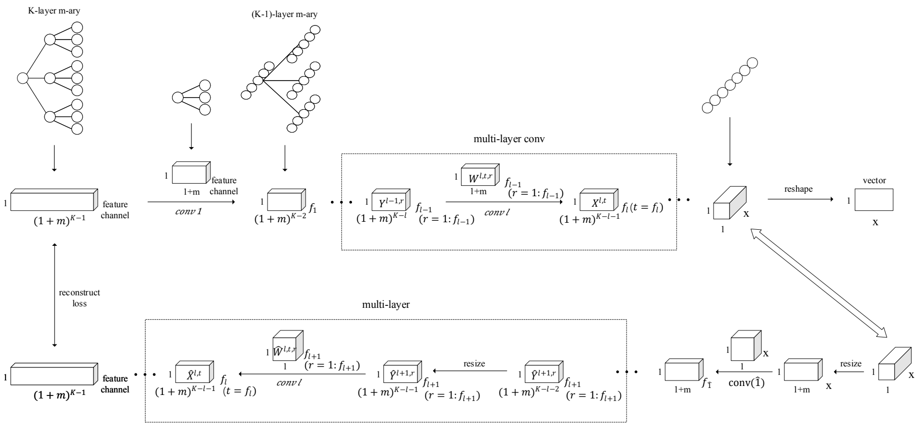


Figure 5: The framework of the convolution process in detail. The whole K-layer m-ary tree as an input of node i, and the dimension changes after every convolution can be clearly shown in the figure. The sketch map in the dotted box corresponds to formula (3) and (4) respectively.

t-th feature in layer l and $b^{l,t}$ is the bias of the t-th filter in layer l.

$$Y_{p,q}^{l-1,r} = \{X_{p,q}^{l-1,r}, X_{p+1,(q-1)m+1}^{l-1,r}, ..., X_{p+1,qm}^{l-1,r}\} \qquad (2)$$

where $Y_{p,q}^{l-1,r}$ is the r-th feature of node (p,q) receptive field in layer $l-1$. Figure 4 shows the convolution process through each channel.

The hidden representation $X^{l,t}$ for the t-th feature channel in layer l can be expressed more concisely using tensor notation as

$$X^{l,t} = \sigma(\sum_{r=1}^{f_{l-1}}(W^{l,t,r} \odot Y^{l-1,r}) + b^{l,t}) \qquad l \leq K \qquad (3)$$

where $\odot$ is the element-wise multiplication.

After we obtain X^l, *i.e.* the network representation, we can compute the output $\hat{d}_i$ by revising the encoder step. So the hidden representation $\hat{X}^{l,t}$ for the t-th feature channel in layer l can be expressed as

$$\hat{X}^{l,t} = \sigma(\sum_{r=1}^{\hat{f}_{l+1}}(\hat{W}^{l,t,r} \odot \hat{Y}^{l+1,r}) + \hat{b}^{l,t}) \qquad (4)$$

where a bilinear interpolation on the output of layer $l+1$ gives the input of layer l. The whole convolution process is shown in Figure 5.

The goal of the depth-based subgraph convolutional autoencoder is to minimize the reconstruction error for the input data of the vertex i, referred to as d_i and the reconstructed data of vertex i an denoted by $\hat{d}_i$. The loss function associated with this process can be shown to be

$$\epsilon = \sum_{i=1}^{n} \parallel d_i - \hat{d}_i \parallel_2^2 \qquad (5)$$

However, Salakhutdinov et al. [37] proved that minimizing the reconstruction loss function does not explicitly preserve the similarity between samples. The local pairwise proximity can be regarded as the information obtained from the neighborhood vertices used to constrain the similarity of the latent representation of a pair of vertices. Therefore, we add the local pairwise proximity to exploit the similarity of linked vertices. In this case, the loss function is defined as

$$\begin{aligned}\epsilon_e &= \sum_{i,j=1}^{n} a_{i,j} \parallel r_i - r_j \parallel_2^2 \\ &= 2tr(R^T L R) \qquad (6)\end{aligned}$$

where L is the Laplacian matrix, given by $L = D - A$, where $D \in R^{n\times n}$ is a diagonal matrix with elements $D_{i,i} = \sum_j a_{i,j}$ and A is the adjacency matrix of the network with elements $a_{i,j}$.

For simplicity of notation, we denote the final obtained network representations as $R = \{r_i\}_{i=1}^{n}$ ($i.e.$ $R = \sigma(W^K Y^{K-1} + b^K)$). For v_i and v_j not linked by an edge, then $a_{i,j} = 0$. Otherwise, for an unweighted graph $a_{i,j} = 1$ and for a weighted graph, $a_{i,j} > 0$. The objective function in Eq. (6) incurs a penalty when vertices linked by an edge are mapped far away from each other in the representation space. As a result, a pair of linked vertices should ideally be mapped to close positions in the representation space.

We combine Eq. (5) and Eq. (6) and jointly minimize the following objective function

$$\mathcal{E} = \epsilon + \alpha\epsilon_e + \beta\epsilon_{reg} \qquad (7)$$

where

$$\epsilon_{reg} = \frac{1}{2}\sum_l \sum_t \sum_r (\parallel W^{l,t,r} \parallel_2 + \parallel \hat{W}^{l,t,r} \parallel_2)$$

is the ℓ_2-norm regularization term to prevent overfitting.

Learning Filters To optimize the aforementioned model, the goal is to minimize the loss ϵ with respect to $\{W^{l,t,r}, \hat{W}^{l,t,r}, b^{l,t}, \hat{b}^{l,t}\}$. The key step in learning the filters in this way is to calculate the matrix partial derivatives $\partial\mathcal{E}/\partial W^{l,t,r}$ and $\partial\mathcal{L}/\partial\hat{W}^{l,t,r}$, the resulting matrix has the same size as the l-th layer filters used to update the weights. The details how to compute the partial derivative is as follows:

$$\frac{\partial\mathcal{L}}{\partial\hat{W}^{l,t,r}} = \frac{\partial\epsilon}{\partial\hat{W}^{l,t,r}} + \beta\frac{\partial\epsilon_{reg}}{\partial\hat{W}^{l,t,r}} \qquad (8)$$

$$\frac{\partial\mathcal{L}}{\partial W^{l,t,r}} = \frac{\partial\epsilon}{\partial W^{l,t,r}} + \alpha\frac{\partial\epsilon_{e}}{\partial W^{l,t,r}} + \beta\frac{\partial\epsilon_{reg}}{\partial W^{l,t,r}} \qquad (9)$$

We first consider the partial derivative $\partial\epsilon/\partial\hat{W}^{l,t,r}$ for the last convolution layer, for which $l = K$-1 in the decoder process, and $\hat{d}_i = \{\hat{X}^{l,t}\}_{t=1}^{f_l}$. As a result the partial derivative formula for the this layer is

$$\frac{\partial\epsilon}{\partial\hat{W}^{l,t,r}} = \frac{\partial\epsilon}{\partial\hat{D}} \cdot \frac{\partial\hat{D}}{\partial\hat{W}^{l,t,r}} \qquad (10)$$

Turning our attention to the first term appearing in the derivative above, according to Eq. (5) we have:

$$\frac{\partial\epsilon}{\partial\hat{D}} = 2(\hat{D} - D) \qquad (11)$$

The above formula gives a matrix with the same dimensions as the filters on the last convolution layer. To compute the derivative $\partial\hat{D}/\partial\hat{W}^{l,t,r}$, we can easily obtain $\partial\hat{d}_i/\partial\hat{W}^{l,t,r}$ and $\partial\hat{d}_i/\partial W^{l,t,r}$ since $\hat{d}_i = \{\hat{X}^{l,t}\}_{t=1}^{f_l}$ and $\hat{X}^{l,t} = \sigma(\sum_{r=1}^{\hat{f}_{l+1}}(\hat{W}^{l,t,r} \odot \hat{Y}^{l+1,r}) + \hat{b}^{l,t})$, and so $\partial\hat{D}/\partial\hat{W}^{l,t,r}$ can be easily compute using the fact that $\hat{D} = \{\hat{d}_i\}_{i=1}^{n}$. As a result

$$\frac{\partial\hat{D}}{\partial\hat{W}^{l,t,r}} = \sum_{i=1}^{n}\frac{\partial\hat{d}_i}{\partial\hat{W}^{l,t,r}} \qquad (12)$$

$$\frac{\partial\hat{D}}{\partial W^{l,t,r}} = \sum_{i=1}^{n}\frac{\partial\hat{d}_i}{\partial W^{l,t,r}} \qquad (13)$$

Here the result is still a matrix with size equal to the filters on the last convolution layer. Therefore, the last layer of $\partial\ell/\partial\hat{W}^{l,t,r}$ is accessible. Based on back-propagation, we can recursively obtain $\partial\ell/\partial\hat{W}^{l,t,r}$, $l = 1, ..., K-1$ and $\partial\ell/\partial W^{l,t,r}$, $l = 1, ..., K$. For the second term, according to ℓ_2-norm regularization, we could get

$$\frac{\partial\epsilon_{reg}}{\partial\hat{W}^{1,t,r}} = \hat{W}^{1,t,r} \qquad (14)$$

$$\frac{\partial\epsilon_{reg}}{\partial W^{1,t,r}} = W^{1,t,r} \qquad (15)$$

The first term of Eq. (8), *i.e.*, Eq. (10), can be computed using Eq. (11) and Eq. (12), while the second term of Eq. (8) can be computed using Eq. (14).

We continue by calculating

$$\frac{\partial\epsilon_e}{\partial W^{l,t,r}} = \frac{\partial\epsilon_e}{\partial R} \cdot \frac{\partial R}{\partial W^{l,t,r}} \qquad (16)$$

Since the final network representations $R = \sigma(W^K Y^{K-1} + b^K)$, forms the last layer of the encoder process, the calculation of the second term $\partial R/\partial W^{l,t,r}$ is relatively easy. For the first term of $\partial\epsilon_e/\partial R$, according to Eq. (6) we have:

$$\frac{\partial\epsilon_e}{\partial R} = 2(L + L^T) \cdot R \qquad (17)$$

The first and third terms of Eq. (9) can be obtained in the same manner as Eq. (8), while the second term of Eq. (9) can be obtained using Eq. (16) and Eq. (17).

5. Experiments and Comparisons

In this section, we summarize the datasets, benchmarks, and evaluation tasks that are commonly used in developing new network embedding methods.

5.1. Datasets

To demonstrate the effectiveness of the proposed approach on node classification and visualization tasks, we conduct experiments on six popular real word networks currently used in network embedding literature. The datasets

Table 2: Statistics of datasets

Properties	Pubmed	Cora	Wikipedia	Email-Eu	Citeseer	Cornell
Nodes	19,717	2,708	2405	1,005	3312	195
Edges	44,338	5,429	17981	25,571	4715	304
Classes	3	7	19	42	6	5
Node features	500	1433	640	-	3703	1703

can be roughly divided into three groups according to the nature of the networks: citation networks, communication networks, and language network. Table.2 summarizes the extent and properties of the six data sets studied.

Citation Networks

Pubmed[1] dataset [38] consists of 19,717 scientific papers from the Pubmed database on the subject of diabetes, where each paper is classified into one of three classes. This citation network that joins the papers consists of 44,338 links, and each paper is represented by a Term Frequency Inverse Document Frequency (TFIDF) vector drawn from a dictionary with 500 terms.

Cora[2] dataset [38] contains 2,708 machine learning articles categorized into seven possible machine learning subjects. Each article is represented by a 01-valued word vector where each feature corresponds to the presence or absence of a term draw from a dictionary. The dictionary contains 1,433 unique entries. This graph contains 5,429 citation edges. We treat the citation links as undirected edges and construct a binary, symmetric adjacency matrix.

CiteSeer [3] dataset [39] contains 3,312 research articles crawled from the CiteSeer repository [38], where each article is categorized into six classes. Each article in the dataset is described by a 0/1-valued word vector indicating the absence/presence of the corresponding word from the dictionary. The dictionary contains 3,703 unique words. This graph contains 4,732 citation edges.

Communication Networks

[1] `https://linqs-data.soe.ucsc.edu/public/Pubmed-Diabetes.tgz`

[2] `https://linqs-data.soe.ucsc.edu/public/lbc/cora.tgz`

[3] `https://linqs-data.soe.ucsc.edu/node/236`

Email-Eu-Core[4] dataset [40] is generated from email data of a large European research institution. There is an edge (u, v) in the dataset if person u sent person v email. The dataset contains ”ground-truth” community memberships of the individuals and each individual belongs to exactly one of 42 departments at the research institute. Note that each vertex of this dataset has no vertex information so that we only take the structural information of the vertex as the input to investigate the ability of DS-CAE on dataset without vertex feature.

Language Networks

Wikipedia[5] dataset [28] are composed of 2,405 real-world webpages from 19 classes and 17,981 hyperlinks between them, where the vertex represents a webpage and the edge indicates that there is a hyperlink from one webpage to another. Webpage text content is often collected as vertex features.

Cornell dataset is subnetwork from the WebKB dataset[6]. This is network of webpages and hyperlinks. Each webpage belongs to one of five classes: course, faculty, student, project, and staff, which serve as ground-truth. The Cornell dataset [41] consists of 195 webpages (*i.e.*, 42 course, 32 faculty, 83 student, 19 project, 19 staff) classified into one of five classes. This webpage network consists of 304 links. Each webpage in the dataset is described by a 0/1-valued word vector indicating the absence/presence of the corresponding word from the dictionary. The dictionary consists of 1703 unique words.

5.2. Baseline Algorithms

We compare our proposed DS-CAE with the following six state-of-art methods.

DeepWalk [12] performs random walks to transforms a graph structure into linear sequences. It then processes the sequences using Skip-Gram model [1] with hierarchical softmax as the loss function.

LINE [13] learns network representations in large-scale networks using first-order and second-order proximities separately rather than exploiting

[4] `http://snap.stanford.edu/data/email-Eu-core.html`

[5] `https://linqs.soe.ucsc.edu/data`

[6] `https://linqs-data.soe.ucsc.edu/public/lbc/WebKB.tgz`

random walks to capture network structure. It then concatenates two representations together.

Laplacian Eigenmaps [8], which use 1st order proximity to preserve the graph structure, learn the graph representation by factorizing Laplacian matrix.

TADW [28] employs matrix factorization to incorporate text features of vertices into network representation learning.

SDNE [15] has multiple layers of non-linear functions to capture the non-linear network structure and exploit the first-order and second-order proximity jointly to preserve the network structure.

GraphSAGE [42] learns node representations through aggregation of its neighborhood information.

Note that SDNE is a semi-supervised method, TADW and GraphSAGE incorporate text features of vertices into NRL.

5.3. Parameter Settings and Evaluation Metrics

DS-CAE is a multi convolutional layer deep model and the number of filters in each layer varies with different datasets. The number of filters in each layer of DS-CAE is listed in Table 3. We apply grid search to set the hyper-parameters α, β on the validation set. As mentioned in [12], for DeepWalk, we set window size as 10, walk length as 40, walks per vertex as 80. As suggested in [13], for LINE, we set the mini-batch size of stochastic gradient descent as 1, starting value of learning rate as 0.025, the number of negative samples as 5, and total number of samples as 10 billion. We also concatenate 1-step and 2-step representation to form final representation and do $\mathcal{L}_2$ normalization to achieve optimal performance. As suggested in [15], for SDNE, we apply grid search to set the hyper-parameters α, β, ν on the validation set

Table 3: Parameters for DS-CAE and SDNE

Dataset	DS-CAE	SDNE
Pubmed	2000-1800	19717-2000-100
Cora	2000-1800	2708-1000-100
Wikipedia	2000-1800	2405-1000-100
Email-Eu	1200-1000	1005-500-100
Citeseer	2000-1800	3312-1000-100
Cornell	600-400	195-150-100

Table 4: Evaluation results of node classification on the Pubmed dataset.

Metric	Percentage Method	10%	20%	30%	40%	50%	60%	70%	80%	90%
Micro-F1 (%)	DeepWalk	76.57 ± 0.28	77.40 ± **0.26**	77.58 ± **0.24**	77.78 ± 0.28	77.97 ± 0.30	78.04 ± 0.26	78.30 ± 0.28	78.55 ± 0.25	79.17 ± 0.29
	LINE	75.13 ± 0.31	77.33 ± 0.29	78.38 ± 0.31	78.96 ± 0.37	79.49 ± 0.36	79.67 ± 0.27	80.08 ± 0.38	81.18 ± 0.33	81.35 ± 0.28
	LE	50.31 ± 0.43	57.51 ± 0.47	61.14 ± 0.41	62.28 ± 0.49	63.05 ± 0.39	63.19 ± 0.40	64.15 ± 0.45	64.58 ± 0.42	66.43 ± 0.43
	TADW	68.84 ± 0.40	69.49 ± 0.42	70.37 ± 0.41	73.27 ± 0.39	75.81 ± 0.42	78.04 ± 0.45	79.37 ± 0.34	80.34 ± 0.44	80.95 ± 0.36
	SDNE	72.86 ± 0.34	74.69 ± 0.39	75.32 ± 0.41	75.68 ± 0.32	75.95 ± 0.37	76.08 ± 0.39	76.32 ± 0.37	75.99 ± 0.35	76.35 ± 0.37
	GraphSAGE	75.21 ± 0.34	76.84 ± 0.36	78.48 ± 0.39	79.06 ± 0.30	79.80 ± 0.41	79.85 ± 0.33	79.99 ± 0.37	80.03 ± 0.34	80.12 ± 0.38
	DS-CAE	**80.99** ± **0.26**	**83.00** ± 0.27	**84.89** ± 0.30	**85.49** ± **0.28**	**86.61** ± **0.29**	**86.84** ± **0.24**	**87.41** ± **0.27**	**87.75** ± **0.25**	**88.59** ± **0.27**
Macro-F1 (%)	DeepWalk	75.20 ± 0.25	75.89 ± 0.24	76.06 ± **0.19**	76.20 ± **0.18**	76.26 ± **0.22**	76.12 ± 0.23	76.49 ± 0.26	76.60 ± 0.23	77.90 ± 0.28
	LINE	71.18 ± 0.29	72.47 ± 0.27	73.65 ± 0.28	74.90 ± 0.34	76.19 ± 0.32	76.50 ± 0.25	77.23 ± 0.31	77.54 ± 0.30	78.12 ± 0.27
	LE	35.01 ± 0.39	42.38 ± 0.41	45.37 ± 0.37	46.18 ± 0.43	48.99 ± 0.36	49.38 ± 0.31	51.59 ± 0.38	52.04 ± 0.35	55.36 ± 0.35
	TADW	61.93 ± 0.35	64.75 ± 0.38	67.07 ± 0.37	70.64 ± 0.33	72.99 ± 0.38	75.04 ± 0.30	77.13 ± 0.29	78.56 ± 0.34	79.70 ± 0.31
	SDNE	68.50 ± 0.32	70.25 ± 0.34	71.42 ± 0.31	71.67 ± 0.29	71.97 ± 0.35	72.01 ± 0.38	72.21 ± 0.37	71.68 ± 0.33	72.06 ± 0.36
	GraphSAGE	74.47 ± 0.31	76.56 ± 0.35	77.81 ± 0.36	78.00 ± 0.27	78.74 ± 0.36	78.61 ± 0.28	78.67 ± 0.27	78.28 ± 0.34	78.97 ± 0.30
	DS-CAE	**80.52** ± **0.23**	**82.85** ± **0.24**	**84.63** ± 0.22	**85.32** ± 0.24	**86.33** ± 0.28	**86.26** ± **0.21**	**87.16** ± **0.19**	**87.47** ± **0.22**	**88.50** ± **0.26**

and the dimension of each layer is listed in Table 3. GraphSAGE provides a variety of approaches to aggregating features within a sampled neighborhood and we choose the unsupervised method of GraphSAGE-mean because it almost results in the best accuracy.

For node classification task, we adopt *Micro*-F1 and *Macro*-F1 as many other works do [14, 15]. *Micro*-F1 gives equal weight to each document and hence considered as an average over all the document/category pairs, it tends to be dominated by the classifier's performance on common categories. *Macro*-F1 gives equal weight to each category, it tends to be dominated by rare categories.

5.4. Experiment Results on Node Classification

We generate the representations for the vertices from the network representation learning and use them as features to classify each vertex into a

Table 5: Evaluation results of node classification on the Cora dataset.

Metric	Percentage Method	10%	20%	30%	40%	50%	60%	70%	80%	90%
Micro-F1 (%)	DeepWalk	65.76 ± 0.90	72.83 ± 0.93	74.95 ± 0.84	75.78 ± 0.95	76.73 ± 0.99	77.19 ± 0.89	77.90 ± 0.92	78.51 ± 0.93	78.26 ± 0.97
	LINE	61.51 ± 0.95	65.18 ± 1.11	66.35 ± 1.05	68.15 ± 1.01	70.53 ± 1.15	70.65 ± 1.10	70.90 ± 1.08	74.15 ± 1.03	78.4 ± 0.98
	LE	30.68 ± **0.78**	31.06 ± **0.81**	31.86 ± 0.83	31.32 ± 0.86	31.02 ± 0.80	31.83 ± 0.79	31.73 ± 0.71	32.84 ± 0.82	35.06 ± 0.76
	TADW	52.32 ± 0.91	57.98 ± 0.84	61.56 ± 0.89	65.82 ± 0.85	68.64 ± 0.96	71.67 ± 0.97	74.40 ± 0.87	76.85 ± 0.82	78.54 ± 0.93
	SDNE	62.35 ± 0.85	66.81 ± 0.89	69.46 ± 0.88	70.94 ± 0.85	71.46 ± **0.81**	72.53 ± 0.93	72.79 ± 0.78	73.98 ± 0.74	73.78 ± 0.77
	GraphSAGE	40.18 ± 0.93	44.35 ± 0.82	52.44 ± 0.89	69.01 ± 0.86	72.82 ± 0.83	76.08 ± 0.96	76.74 ± 0.85	77.92 ± 0.84	78.82 ± 0.78
	DS-CAE	**67.07** ± 0.89	**73.96** ± 0.86	**76.14** ± **0.74**	**78.62** ± **0.81**	**79.26** ± 0.88	**79.91** ± **0.76**	**81.05** ± **0.63**	**81.08** ± **0.70**	**82.54** ± **0.75**
Macro-F1 (%)	DeepWalk	64.05 ± 0.81	69.89 ± 0.86	72.21 ± 0.91	75.81 ± 0.87	75.99 ± 0.93	76.52 ± 0.90	77.04 ± 0.84	77.90 ± 0.96	77.32 ± 0.89
	LINE	58.05 ± 0.93	62.77 ± 1.06	66.65 ± 0.99	68.78 ± 0.95	70.97 ± 1.12	72.04 ± 1.08	72.84 ± 1.03	77.79 ± 0.96	79.56 ± 0.92
	LE	36.55 ± **0.75**	36.47 ± **0.73**	36.44 ± 0.77	36.40 ± 0.77	36.12 ± **0.74**	36.42 ± 0.76	36.17 ± 0.65	35.87 ± 0.67	36.26 ± 0.70
	TADW	50.98 ± 0.83	56.16 ± 0.79	61.17 ± 0.86	65.65 ± 0.81	68.01 ± 0.95	71.07 ± 0.93	74.41 ± 0.84	76.34 ± 0.79	79.80 ± 0.89
	SDNE	59.42 ± 0.81	64.63 ± 0.77	67.61 ± 0.86	69.01 ± 0.79	69.82 ± 0.84	70.93 ± 0.83	71.36 ± 0.78	72.63 ± 0.69	72.04 ± 0.75
	GraphSAGE	43.37 ± 0.85	47.69 ± 0.79	50.32 ± 0.84	55.68 ± 0.86	60.66 ± 0.81	63.99 ± 0.87	66.43 ± 0.79	71.59 ± 0.74	72.35 ± 0.71
	DS-CAE	**65.14** ± 0.78	**70.95** ± 0.75	**73.30** ± **0.73**	**77.28** ± **0.77**	**77.61** ± 0.79	**78.92** ± **0.71**	**79.57** ± **0.60**	**81.26** ± **0.62**	**82.80** ± **0.68**

Table 6: Evaluation results of node classification on the Email-Eu dataset.

Metric	Percentage Method	10%	20%	30%	40%	50%	60%	70%	80%	90%
Micro-F1 (%)	DeepWalk	52.03 ± 1.17	61.69 ± 1.20	63.52 ± 1.23	65.25 ± 1.21	65.17 ± 1.19	67.14 ± 1.25	68.21 ± 1.18	68.35 ± 1.24	68.59 ± 1.20
	LINE	36.00 ± **1.13**	46.55 ± **1.15**	50.57 ± 1.16	53.01 ± 1.18	54.52 ± 1.21	57.31 ± 1.19	58.74 ± 1.14	59.97 ±1.16	60.20 ± 1.11
	LE	29.70 ± 1.18	30.90 ± 1.19	30.11 ± 1.15	30.28 ± 1.11	30.81 ± 1.23	31.48 ± 1.14	31.19 ± 1.20	31.15 ± 1.14	32.87 ± 1.23
	TADW	30.17 ± 1.26	36.32 ± 1.22	42.47 ± 1.21	46.43 ± 1.18	51.49 ± 1.20	54.48 ± 1.13	59.93 ± 1.17	58.21 ± 1.24	61.39 ± 1.21
	SDNE	**52.38** ± 1.24	**62.56** ± 1.28	**64.02** ± 1.19	**66.13** ± 1.21	67.00 ± 1.27	67.31 ± 1.14	69.18 ± 1.17	70.66 ± 1.23	71.32 ± 1.29
	GraphSAGE	36.86 ± 1.18	46.69 ± 1.17	50.32 ± 1.22	53.68 ± 1.16	54.95 ± 1.19	57.08 ± 1.17	58.32 ± 1.14	59.99 ± 1.19	60.35 ± **1.14**
	DS-CAE	40.79 ± 1.15	54.82 ± 1.19	60.32 ± **1.11**	63.79 ± **1.05**	**68.54** ± **1.17**	**68.76** ± **1.08**	**70.32** ± **1.12**	**72.09** ± **1.12**	**73.11** ± 1.17
Macro-F1 (%)	DeepWalk	23.67 ± **1.08**	31.09 ± 1.15	35.33 ± 1.14	37.21 ± 1.18	37.87 ± 1.17	39.86 ± 1.19	40.59 ± 1.11	40.22 ± 1.13	41.84 ± 1.16
	LINE	17.82 ± 0.94	22.19 ± **1.03**	25.70 ± 1.16	28.83 ± 1.14	30.43 ± 1.18	33.50 ± 1.21	34.81 ± 1.11	35.66 ± 1.09	39.87 ±1.14
	LE	11.08 ± 1.15	11.36 ± 1.18	12.11 ± 1.11	13.48 ± **1.05**	15.15 ± 1.19	15.16 ± 1.19	16.28 ± 1.16	18.73 ± 1.12	19.54 ± 1.20
	TADW	15.54 ± 1.19	17.81 ± 1.21	19.17 ± 1.22	20.26 ± 1.17	23.03 ± 1.26	24.43 ± 1.28	27.71 ± 1.14	27.89 ± 1.13	27.79 ± 1.20
	SDNE	**23.76** ± 1.22	**31.63** ± 1.25	**36.85** ± 1.18	**37.98** ± 1.15	38.01 ± 1.20	39.78 ± 1.28	40.76 ± 1.13	40.37 ± 1.17	41.62 ± 1.21
	GraphSAGE	17.77 ± 1.14	22.69 ± 1.09	25.32 ± 1.15	28.68 ± 1.14	30.95 ± **1.07**	33.08 ± 1.16	34.32 ± 1.11	35.99 ± 1.08	38.35 ± 1.12
	DS-CAE	21.58 ± 1.12	25.57 ± 1.16	33.11 ± **1.08**	34.48 ± 1.13	**39.35** ± 1.15	**41.26** ± **1.13**	**42.85** ± **1.05**	**43.95** ± **1.07**	**44.67** ± **1.05**

Table 7: Evaluation results of node classification on the Wikipedia dataset.

Metric	Percentage Method	10%	20%	30%	40%	50%	60%	70%	80%	90%
Micro-F1 (%)	DeepWalk	**58.66** ± 0.78	**63.82** ± 0.80	**66.50** ± 0.77	**67.35** ± 0.85	65.57 ± 0.84	68.60 ± 0.79	70.22 ± 0.78	70.89 ± 0.80	68.46 ± 0.83
	LINE	56.48 ± 0.79	59.09 ± 0.83	63.90 ± 0.81	65.38 ± 0.88	65.50 ± 0.85	64.10 ± 0.80	61.97 ± 0.89	67.98 ± 0.79	67.72 ± 0.84
	LE	29.78 **± 0.37**	30.97 **± 0.42**	30.11 **± 0.43**	30.23 **± 0.40**	30.21 **± 0.44**	31.78 **± 0.42**	31.59 **± 0.41**	31.75 **± 0.43**	32.56 **± 0.41**
	TADW	54.08 ± 0.83	60.65 ± 0.89	63.59 ± 0.84	64.79 ± 0.85	66.91 ± 0.86	67.35 ± 0.84	68.00 ± 0.83	66.73 ± 0.86	64.73 ± 0.81
	SDNE	46.98 ± 0.74	52.57 ± 0.78	54.99 ± 0.79	56.25 ± 0.79	57.14 ± 0.81	57.97 ± 0.76	57.74 ± 0.79	58.26 ± 0.82	58.56 ± 0.79
	GraphSAGE	45.54 ± 0.75	47.81 ± 0.79	52.17 ± 0.82	55.26 ± 0.83	56.03 ± 0.82	57.43 ± 0.79	57.71 ± 0.81	58.89 ± 0.83	60.79 ± 0.80
	DS-CAE	49.03 ± 0.72	57.10 ± 0.78	59.39 ± 0.74	61.26 ± 0.81	**67.41** ± 0.81	**69.73** ± 0.75	**70.90** ± 0.76	**71.52** ± 0.79	**73.32** ± 0.74
Macro-F1 (%)	DeepWalk	**42.41** ± 0.72	**51.48** ± 0.76	**56.30** ± 0.73	**57.94** ± 0.75	**58.21** ± 0.79	59.30 ± 0.76	60.30 ± 0.79	63.03 ± 0.80	63.93 ± 0.76
	LINE	39.18 ± 0.74	41.54 ± 0.79	45.73 ± 0.73	49.54 ± 0.81	52.48 ± 0.79	51.37 ± 0.77	55.50 ± 0.75	55.68 ± 0.78	53.00 ± 0.75
	LE	11.88 **± 0.32**	11.76 **± 0.35**	12.21 **± 0.37**	13.88 **± 0.38**	15.75 **± 0.42**	15.26 **± 0.41**	16.38 **± 0.39**	18.33 **± 0.41**	19.44 **± 0.40**
	TADW	37.84 ± 0.72	44.62 ± 0.81	48.83 ± 0.82	50.26 ± 0.82	53.17 ± 0.84	53.31 ± 0.81	53.66 ± 0.83	51.50 ± 0.85	53.30 ± 0.80
	SDNE	29.95 ± 0.71	34.30 ± 0.73	37.02 ± 0.76	38.59 ± 0.79	38.70 ± 0.78	39.75 ± 0.79	39.74 ± 0.75	39.49 ± 0.73	38.67 ± 0.74
	GraphSAGE	25.54 ± 0.72	27.81 ± 0.75	29.17 ± 0.76	30.26 ± 0.81	33.03 ± 0.82	34.43 ± 0.80	37.71 ± 0.77	37.89 ± 0.76	37.79 ± 0.75
	DS-CAE	39.25 ± 0.69	46.91 ± 0.75	51.26 ± 0.71	55.56 ± 0.78	56.85 ± 0.81	**59.66** ± 0.74	**61.20** ± 0.73	**64.22** ± 0.75	**65.26** ± 0.72

Table 8: Evaluation results of node classification on the Citeseer dataset.

Metric	Percentage Method	10%	20%	30%	40%	50%	60%	70%	80%	90%
Micro-F1 (%)	DeepWalk	53.10 ± 0.23	55.36 ± 0.25	57.74 ± 0.29	57.55 ± 0.22	58.63 ± 0.20	57.88 ± 0.27	58.15 ± 0.24	60.18 ± 0.27	60.63 ± 0.28
	LINE	44.46 ± 0.22	44.60 ± 0.26	47.82 ± 0.30	48.77 ± 0.27	48.79 ± 0.25	49.58 ± 0.29	50.10 ± 0.22	50.38 ± 0.24	51.49 ± 0.25
	LE	55.45 ± 0.19	54.15 ± 0.25	58.86 ± 0.27	60.31 ± 0.22	59.84 ± 0.26	58.34 ± 0.24	59.56 ± 0.21	61.99 ± 0.22	56.33 ± 0.27
	TADW	67.29 ± 0.20	69.56 ± 0.21	**70.88** ± 0.23	**70.59** ± 0.20	69.73 ± 0.19	**70.81** ± 0.18	70.12 ± 0.21	71.24 ± 0.21	70.68 ± 0.20
	SDNE	17.85 ± 0.21	18.05 ± **0.15**	18.85 ± **0.19**	19.22 ± 0.20	19.6 ± 0.18	20.11 ± 0.18	21.12 ± 0.19	21.40 ± 0.21	22.93 ± 0.20
	GraphSAGE	66.31 ± 0.19	69.61 ± 0.20	70.14 ± 0.23	70.16 ± 0.21	70.04 ± 0.18	70.28 ± 0.19	70.47 ± 0.21	70.48 ± 0.20	70.69 ± 0.19
	DS-CAE	**68.10** ± **0.18**	**69.69** ± 0.20	70.03 ± 0.22	70.29 ± **0.19**	**70.40** ± **0.16**	70.26 ± **0.17**	**70.63** ± **0.18**	**71.37** ± **0.19**	**71.39** ± **0.17**
Macro-F1 (%)	DeepWalk	49.26 ± 0.19	51.17 ± 0.22	53.01 ± 0.25	53.59 ± 0.18	54.39 ± 0.19	53.51 ± 0.17	52.65 ± 0.18	55.40 ± 0.23	55.87 ± 0.19
	LINE	40.79 ± 0.17	41.55 ± 0.19	43.96 ± 0.28	44.28 ± 0.24	44.20 ± 0.23	45.03 ± 0.20	45.59 ± 0.18	45.69 ± 0.19	45.15 ± 0.17
	LE	50.41 ± 0.16	49.14 ± 0.24	53.93 ± 0.21	55.23 ± 0.19	54.91 ± 0.16	52.95 ± 0.17	54.04 ± 0.14	57.56 ± 0.20	53.51 ± 0.18
	TADW	61.39 ± 0.16	65.17 ± **0.17**	67.70 ± 0.19	**68.91** ± 0.18	**69.05** ± 0.15	69.21 ± 0.17	68.42 ± 0.15	67.35 ± 0.19	67.32 ± 0.15
	SDNE	15.81 ± 0.19	16.13 ± 0.12	16.55 ± **0.11**	16.78 ± **0.10**	17.09 ± 0.15	17.21 ± 0.16	17.36 ± 0.13	17.93 ± 0.18	18.03 ± 0.14
	GraphSAGE	62.20 ± 0.18	65.21 ± 0.18	66.34 ± 0.21	66.73 ± 0.19	66.71 ± 0.17	66.73 ± 0.18	66.82 ± 0.15	67.04 ± 0.19	68.19 ± 0.17
	DS-CAE	**62.33** ± **0.15**	**65.37** ± 0.18	**69.74** ± 0.19	67.64 ± 0.17	68.69 ± **0.14**	**69.83** ± **0.13**	**69.78** ± **0.13**	**68.37** ± **0.16**	**68.64** ± **0.14**

Table 9: Evaluation results of node classification on the Cornell dataset.

Metric	Percentage Method	10%	20%	30%	40%	50%	60%	70%	80%	90%
Micro-F1 (%)	DeepWalk	27.84 ± 0.25	35.90 ± 0.33	38.69 ± 0.35	35.90 ± 0.34	39.80 ± 0.37	38.46 ± 0.30	44.07 ± 0.35	33.33 ± 0.39	30.00 ± 0.25
	LINE	32.95 ± 0.22	38.46 ± 0.34	41.61 ± 0.36	41.88 ± 0.33	35.71 ± 0.41	39.74 ± 0.39	35.59 ± 0.34	41.02 ± 0.32	34.99 ± 0.31
	LE	28.41 ± 0.25	37.18 ± 0.31	39.41 ± 0.28	35.90 ± 0.36	36.73 ± 0.34	38.46 ± 0.29	35.60 ± 0.32	30.77 ± 0.35	30.00 ± 0.26
	TADW	43.18 ± 0.26	41.67 ± 0.38	43.07 ± 0.37	47.01 ± 0.34	40.82 ± 0.40	**47.44** ± 0.35	**49.15** ± 0.34	41.03 ± 0.36	35.00 ± 0.35
	SDNE	35.23 ± 0.23	37.50 ± 0.25	38.07 ± 0.27	38.64 ± 0.23	39.20 ± 0.28	39.77 ± **0.23**	40.34 ± 0.24	40.91 ± 0.24	40.05 ± **0.23**
	GraphSAGE	36.84 ± 0.21	39.55 ± 0.27	41.03 ± 0.24	41.41 ± **0.20**	41.77 ± **0.25**	42.03 ± 0.26	44.06 ± 0.28	**41.41** ± 0.27	40.19 ± 0.28
	DS-CAE	**48.28** ± **0.20**	**48.39** ± **0.20**	**48.19** ± **0.23**	**50.00** ± 0.24	**42.76** ± 0.29	44.97 ± 0.25	42.40 ± **0.21**	40.15 ±**0.24**	**42.27** ± 0.25
Macro-F1 (%)	DeepWalk	19.04 ± 0.23	17.55 ± 0.31	20.92 ± 0.34	14.88 ± 0.38	22.04 ± 0.37	17.92 ± 0.25	**24.76** ± 0.26	13.52 ± 0.28	14.52 ± 0.21
	LINE	25.15 ± 0.20	23.20 ± 0.28	24.89 ± 0.24	25.58 ± 0.22	23.33 ± 0.23	20.51 ± 0.27	15.65 ± 0.31	19.88 ± 0.29	18.41 ± 0.21
	LE	24.20 ± 0.25	22.19 ± 0.28	22.25 ± 0.23	18.69 ± 0.32	22.62 ± 0.35	20.43 ± 0.27	15.18 ± 0.30	12.90 ± 0.28	9.60 ± 0.25
	TADW	25.14 ± 0.24	16.06 ± 0.39	19.56 ± 0.28	23.86 ± 0.29	20.97 ± 0.28	21.13 ± 0.25	21.89 ± 0.21	17.17 ± 0.22	15.43 ± 0.26
	SDNE	18.17 ± 0.18	19.04 ± **0.19**	19.15 ± 0.19	19.21 ± **0.18**	22.29 ± 0.23	20.30 ± **0.22**	17.24 ± 0.24	17.29 ± **0.19**	19.79 ± 0.24
	GraphSAGE	19.48 ± 0.19	17.48 ± 0.20	18.94 ± **0.18**	19.35 ± 0.19	21.19 ±**0.22**	21.01 ± 0.23	22.78 ± 0.21	**20.35** ± 0.22	19.75 ± 0.23
	DS-CAE	**26.67** ± **0.18**	**27.45** ± 0.21	**24.94** ± 0.23	**26.80** ± 0.20	**24.15** ± 0.25	**21.33** ± 0.24	21.81 ± **0.19**	16.21 ± 0.23	**19.95** ± **0.21**

set of labels. For all models we use a one-vs-rest logistic regression implemented by LibLinear [43], extended to return the most probable labels as [12]. Specifically, we randomly sample a portion of labeled vertices and use them as training data. The rest of the vertices are used as test data. We randomly sample 10% to 90% of the vertices as the training samples and use the left vertices to test the performance. We repeat this process 5 times and report the average performance in terms of both *Micro*-F1 and *Macro*-F1 as many other works do [15]. Table 4 - 9 report the average *Micro*-F1 and *Macro*-F1 values of the different algorithms on node classification. The boldfaced values are the best results. From these tables, we have following observations:

(1) For datasets with node features (Cora, Pubmed, Wiki), our proposed DS-CAE outperforms each of the competing methods with different training ratios. Specially, as shown in Table 4, for the large dataset-Pubmed, DS-CAE outperforms the best baseline in each training ratio by 4.4% to 7.3% in *Micro*-F1 and 5.3% to 8.91% in *Macro*-F1. It indicates the effectiveness and robust of DS-CAE on node classification task. However, in Cornell dataset, we observed that the performances of DS-CAE are dropping when the number of training example increases. A possible explanation is the imbalance of data samples in this dataset. The Cornell dataset [41] consists of 195 webpages (*i.e.*, 42 course, 32 faculty, 83 student, 19 project, 19 staff) classified into one of five classes. Here, the samples of student class accounts for nearly 50% of the total, while the project and staff class samples account for only 10%.

(2) For the dataset without node features (Email-Eu), DS-CAE outperforms other baselines with training ratios from 50% to 90%. Mainly because DS-CAE takes the structure of each vertex neighborhood as input and constructs a richer representation. This means a larger training dataset can learn a better representation than other baselines and in turn gives a better precision. Note that our DS-CAE is not designed for this kind of dataset, but the performance shows a good generalization ability for our DS-CAE method.

(3) DS-CAE has an encouraging performance when train ratio is large. The accuracies of most of the baselines increase slowly as training ratio increases, that is because DeepWalk based models (DeepWalk, LINE and TADW) are ‘shallow’ models which have relatively limited capacity to capture the highly nonlinear structures of graphs. SDNE can only encode the structural information rather than both structure and node feature information. Our proposed DS-CAE outperforms GraphSAGE-mean (taking the

elementwise mean value of feature vectors) suggesting that assigning different importance to different nodes within a subgraph while dealing with different sized neighborhoods may be beneficial. DS-CAE takes the node feature of each vertex neighborhood as input and takes local pairwise proximity into consideration. This gives a richer representation and larger datasets can learn a better representation. This shows the advantage of our DS-CAE in Big Data.

To summarize, all of the above observations demonstrate that DS-CAE can learn high-quality and rich representation, which are conducive to node classification. Moreover, the experimental results on the node classification task demonstrates the effectiveness of our DS-CAE method.

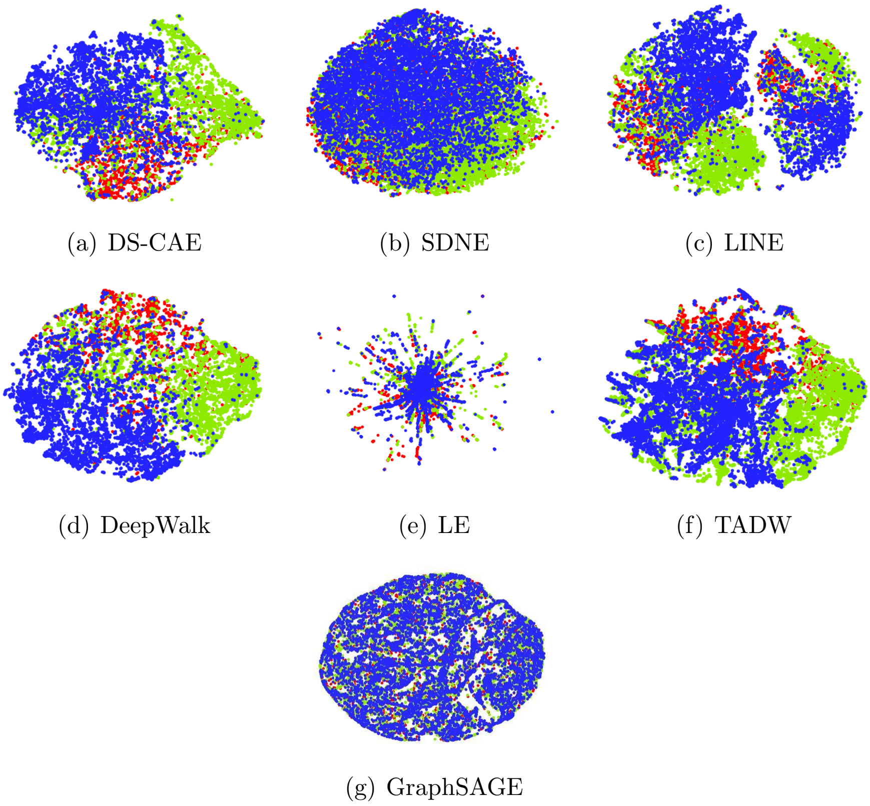

(a) DS-CAE (b) SDNE (c) LINE (d) DeepWalk (e) LE (f) TADW (g) GraphSAGE

Figure 6: Visualization of the Pubmed dataset. Each point indicates a paper.

5.5. Experiment Results on Visualization

We visualize the learned representations of citation dataset-Pubmed on a 2D space by t-SNE [44] in Figure 6, where the same category of papers were labeled as the same color. Similar to [14], we also use the Kullback-Leibler divergence as a quantitative evaluation metric, the lower the KL divergence, the better the performance. We report the results in Table 10, the boldfaced values are the best one.

Table 10: KL-divergence for the Pubmed dataset

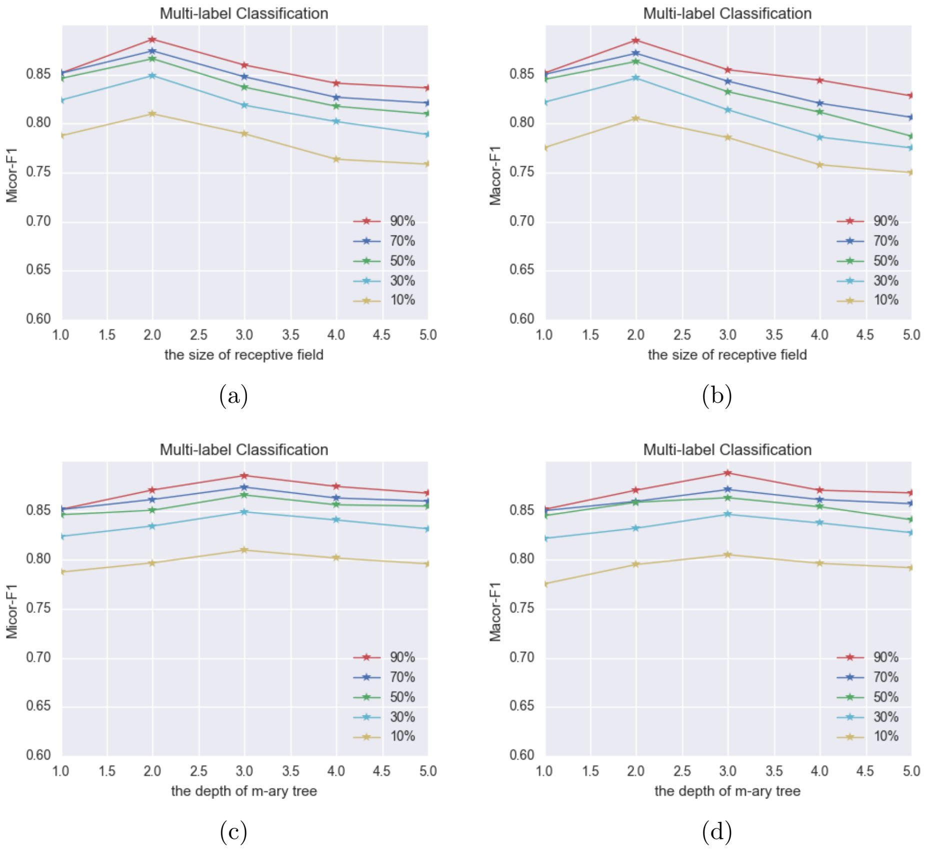
Algorithm	DS-CAE	SDNE	LINE	GraphSAGE	TADW	DeepWalk	LE
KL divergence	**0.85638**	0.91299	1.13514	1.05960	1.00638	1.09793	1.04166

From Figure 6, we observe that DS-CAE learns a better clustering and separation of the vertices. On the contrary, the baseline methods showed unclear boundaries and most points belonging to different categories are mixed with each other. The results shown in Table 10 also quantitatively demonstrate the superiority of DS-CAE in the visualization task.

5.6. Parameter Sensitivity

We investigate the parameter sensitivity in this section. Specifically, we first evaluate how the node classification accuracies vary with increasing receptive field size of $m + 1$ and the depth K of the m-ary tree. We report *Micro*-F1 and *Macro*-F1 values on the dataset of Pubmed in Figure 7.

We first investigate how the size of receptive field affects the performance in Figure 7(a) and Figure 7(b) with a fixed depth $K = 3$. We can see that the classification accuracy first increases to a maximum value at $m + 1 = 2$ and decreases with increasing m. Note that when $m + 1 = 1$, the encoder only uses the node features of each vertex. We show how the value of depth K affects the performance in Figure 7(c) and Figure 7(d) with a fixed receptive field size $m+1 = 2$. We can see that the classification accuracy first increases to a maximum value at $K = 3$ and then decreases with increasing K. This phenomenon is intuitive because to gradually propagate information from the local vertex level to the global graph level, we could expand the subtree rooted at each vertex with the aim to capture the structural information around each vertex. One approach is to gradually expand the subtree structure rooted at each vertex until it reaches the global graph level, and in so doing include more vertices. However, vertices that are far away from the root vertex are likely to have low relevance to the root vertex. Including them may



(a)

(b)

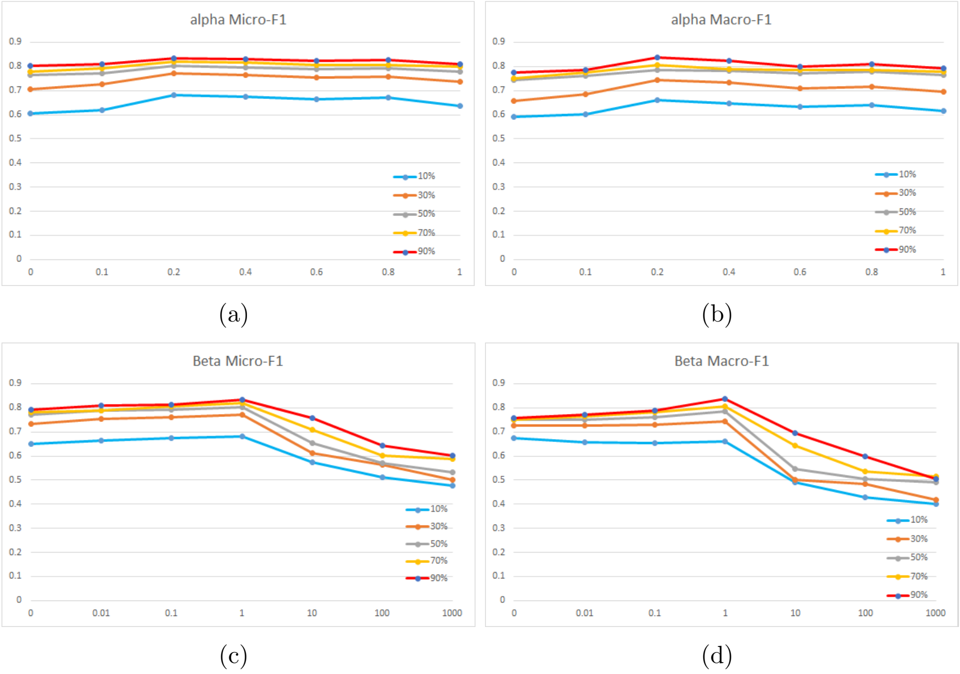
(c)

(d)

Figure 7: Impact of the receptive size and the depth of m-ary tree on performance for node classification.

encourage the propagation of noise, and this may compromise performance. Thus, to provide compromise which allows a trade-off between local and global information, we set the depth of m-ary tree as 3. Our experiments also demonstrate that the choice of these parameters provides better performance. This observation further verifies the effectiveness of our proposed DS-CAE which integrates both global topological arrangement information and local connectivity properties within a graph to conduct graph convolutional auto-encoder.

Besides the reconstruction loss ϵ which is used to preserve global network structure, our objective function in Eq. (7) also uses two additional



(a) (b)

(c) (d)

Figure 8: Impact of α and β values in Eq. (7) on performance for node classification.

losses, namely a) ϵ_e is used to preserve the local network structure using first order proximity and b) ϵ_{reg} is an ℓ_2-norm regularizer term used to prevent overfitting. These two losses are controlled using two hyperparameters - α balances the significance between global network structure and local network structure, and β is a regularization coefficient that is used to control the reconstruction weights in the training graph. We have investigated the effect of different α values on the classification performance on the Cora dataset in Figure 8(a) and Figure 8(b). When α=0, the performance is totally determined by the global network structure. The larger the value of α, the more the model concentrates on the local network structure. From Figure 8(a) and Figure 8(b), we can see that the performance when α=0.2 is better than that when α=0. This demonstrates that both global network structure and local network structure are essential for network embedding methods to accurately characterize the network structure. Finally, we show how the value of β affects the performance in Figure 8(c) and Figure 8(d). We can see that the performance improves as the hyperparameter β grows from zero to unity. However, when β is too large, the performance deteriorates instead. The reason is that an appropriate regularization term is needed to constrain the model so as to prevent over-fitting. However, when the constraint is too strong, the model cannot learn effective representations.

6. Conclusions

In this paper, we have explored the challenging problem of how to use the convolutional autoencoder for modeling non-lattice graphical structures. The convolution process makes use of both global topological arrangement information and local connectivity structures within a graph. In particular, the proposed model DS-CAE can comprehensively integrate node content information and network structure into unsupervised network representation learning, thus the learned representation can capture highly non-linear relationships between nodes and complex features of a network. Experimental results on node classification and visualization tasks show our DS-CAE method is superior to a number of baseline methods.

Our future plans are to extend the work in a number of ways. First, in prior work, we have developed methods for characterizing graphs using the commute time [45] and the heat kernel [46]. For an undirected graph, both of these methods encapsulate the path length distribution between vertices. It would be interesting to use the commute time or heat kernel as a means of node ordering. Second, the current formulations of graph convolution are restricted to use vertex information and do not make use of edge labels. It would be interesting to design network representation learning framework which simultaneously learns properties from both graph vertices and edges.

References

[1] T. Mikolov, K. Chen, G. Corrado, J. Dean, Efficient estimation of word representations in vector space, arXiv preprint arXiv:1301.3781.

[2] P. F. Felzenszwalb, D. P. Huttenlocher, Efficient graph-based image segmentation, International journal of computer vision 59 (2) (2004) 167–181.

[3] S. Bhagat, G. Cormode, S. Muthukrishnan, Node classification in social networks, in: Social network data analytics, Springer, 2011, pp. 115–148.

[4] Y. Sun, B. Norick, J. Han, X. Yan, P. S. Yu, X. Yu, Pathselclus: Integrating meta-path selection with user-guided object clustering in heterogeneous information networks, ACM Transactions on Knowledge Discovery from Data (TKDD) 7 (3) (2013) 11.

[5] D. Luo, F. Nie, H. Huang, C. H. Ding, Cauchy graph embedding, in: Proceedings of the 28th International Conference on Machine Learning (ICML-11), 2011, pp. 553–560.

[6] J. Lu, J. Xuan, G. Zhang, X. Luo, Structural property-aware multilayer network embedding for latent factor analysis, Pattern Recognition 76 (2018) 228–241.

[7] S. T. Roweis, L. K. Saul, Nonlinear dimensionality reduction by locally linear embedding, science 290 (5500) (2000) 2323–2326.

[8] M. Belkin, P. Niyogi, Laplacian eigenmaps and spectral techniques for embedding and clustering, in: Advances in neural information processing systems, 2002, pp. 585–591.

[9] J. B. Tenenbaum, V. De Silva, J. C. Langford, A global geometric framework for nonlinear dimensionality reduction, science 290 (5500) (2000) 2319–2323.

[10] Y. Zhang, Z. Zhang, J. Qin, L. Zhang, B. Li, F. Li, Semi-supervised local multi-manifold isomap by linear embedding for feature extraction, Pattern Recognition 76 (2018) 662–678.

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

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

[13] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, Q. Mei, Line: Large-scale information network embedding, in: Proceedings of the 24th International Conference on World Wide Web, International World Wide Web Conferences Steering Committee, 2015, pp. 1067–1077.

[14] S. Cao, W. Lu, Q. Xu, Grarep: Learning graph representations with global structural information, in: Proceedings of the 24th ACM International on Conference on Information and Knowledge Management, ACM, 2015, pp. 891–900.

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

[16] C. Tu, H. Wang, X. Zeng, Z. Liu, M. Sun, Community-enhanced network representation learning for network analysis, arXiv preprint arXiv:1611.06645.

[17] B. Wang, Y. Hu, J. Gao, M. Ali, D. Tien, Y. Sun, B. Yin, Low rank representation on spd matrices with log-euclidean metric, Pattern Recognition 76 (2018) 623–634.

[18] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, P. Vandergheynst, Geometric learning: going beyond euclidean data, arXiv preprint arXiv:1611.08097.

[19] T. N. Kipf, M. Welling, Semi-supervised classification with graph convolutional networks, arXiv preprint arXiv:1609.02907.

[20] M. Defferrard, X. Bresson, P. Vandergheynst, Convolutional neural networks on graphs with fast localized spectral filtering, in: Advances in Neural Information Processing Systems, 2016, pp. 3844–3852.

[21] M. Henaff, J. Bruna, Y. LeCun, Deep convolutional networks on graph-structured data, arXiv preprint arXiv:1506.05163.

[22] M. Niepert, M. Ahmed, K. Kutzkov, Learning convolutional neural networks for graphs, in: International Conference on Machine Learning, 2016, pp. 2014–2023.

[23] Z. Zhang, D. Chen, J. Wang, L. Bai, E. R. Hancock, Quantum-based subgraph convolutional neural networks, Pattern Recognition 88 (2019) 38–49.

[24] F. Tian, B. Gao, Q. Cui, E. Chen, T.-Y. Liu, Learning deep representations for graph clustering., in: AAAI, 2014, pp. 1293–1299.

[25] L. Tang, H. Liu, Relational learning via latent social dimensions, in: Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, 2009, pp. 817–826.

[26] H. Bahonar, A. Mirzaei, R. C. Wilson, Diffusion wavelet embedding: A multi-resolution approach for graph embedding in vector space, Pattern Recognition 74 (2018) 518–530.

[27] X. Huang, J. Li, X. Hu, Label informed attributed network embedding, in: Proceedings of the Tenth ACM International Conference on Web Search and Data Mining, ACM, 2017, pp. 731–739.

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

[29] C. Tu, W. Zhang, Z. Liu, M. Sun, Max-margin deepwalk: Discriminative learning of network representation., in: IJCAI, 2016, pp. 3889–3895.

[30] J. Gu, Z. Wang, J. Kuen, L. Ma, A. Shahroudy, B. Shuai, T. Liu, X. Wang, G. Wang, J. Cai, et al., Recent advances in convolutional neural networks, Pattern Recognition 77 (2018) 354–377.

[31] J. Masci, U. Meier, D. Cireşan, J. Schmidhuber, Stacked convolutional auto-encoders for hierarchical feature extraction, Artificial Neural Networks and Machine Learning–ICANN 2011 (2011) 52–59.

[32] T. Opsahl, F. Agneessens, J. Skvoretz, Node centrality in weighted networks: Generalizing degree and shortest paths, Social networks 32 (3) (2010) 245–251.

[33] L. Page, S. Brin, R. Motwani, T. Winograd, The pagerank citation ranking: Bringing order to the web., Tech. rep., Stanford InfoLab (1999).

[34] P. Bonacich, Some unique properties of eigenvector centrality, Social networks 29 (4) (2007) 555–564.

[35] K. Kersting, M. Mladenov, R. Garnett, M. Grohe, Power iterated color refinement, in: Twenty-Eighth AAAI Conference on Artificial Intelligence, 2014.

[36] S. Brin, L. Page, The anatomy of a large-scale hypertextual web search engine, Computer networks and ISDN systems 30 (1-7) (1998) 107–117.

[37] R. Salakhutdinov, G. Hinton, Semantic hashing, International Journal of Approximate Reasoning 50 (7) (2009) 969–978.

[38] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, T. Eliassi-Rad, Collective classification in network data, AI magazine 29 (3) (2008) 93.

[39] A. K. McCallum, K. Nigam, J. Rennie, K. Seymore, Automating the construction of internet portals with machine learning, Information Retrieval 3 (2) (2000) 127–163.

[40] H. Yin, A. R. Benson, J. Leskovec, D. F. Gleich, Local higher-order graph clustering, in: Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2017, pp. 555–564.

[41] Q. Lu, L. Getoor, Link-based classification, in: Proceedings of the 20th International Conference on Machine Learning (ICML-03), 2003, pp. 496–503.

[42] W. L. Hamilton, R. Ying, J. Leskovec, Inductive representation learning on large graphs, CoRR abs/1706.02216.

[43] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, C.-J. Lin, Liblinear: A library for large linear classification, Journal of machine learning research 9 (Aug) (2008) 1871–1874.

[44] L. v. d. Maaten, G. Hinton, Visualizing data using t-sne, Journal of Machine Learning Research 9 (Nov) (2008) 2579–2605.

[45] H. Qiu, E. R. Hancock, Clustering and embedding using commute times, IEEE Transactions on Pattern Analysis and Machine Intelligence 29 (11).

[46] B. Xiao, E. R. Hancock, R. C. Wilson, Graph characteristics from the heat kernel trace, Pattern Recognition 42 (11) (2009) 2589–2606.