Frontiers of Information Technology & Electronic Engineering
www.jzus.zju.edu.cn; engineering.cae.cn; www.springerlink.com
ISSN 2095-9184 (print); ISSN 2095-9230 (online)
E-mail: jzus@zju.edu.cn

A local density optimization method based on a graph convolutional network*

Hao WANG[1,2], Li-yan DONG[1,2], Tie-hu FAN[‡3], Ming-hui SUN[1,2]

[1]College of Computer Science and Technology, Jilin University, Changchun 130012, China

[2]MOE Key Laboratory of Symbolic Computation and Knowledge Engineering, Jilin University, Changchun 130012, China

[3]College of Instrumentation & Electrical Engineering, Jilin University, Changchun 130012, China

E-mail: wanghao18@mails.jlu.edu.cn; dongly@jlu.edu.cn; fth@jlu.edu.cn; smh@jlu.edu.cn

Received Nov. 30, 2019; Revision accepted Apr. 21, 2020; Crosschecked Sept. 24, 2020

Abstract: Success has been obtained using a semi-supervised graph analysis method based on a graph convolutional network (GCN). However, GCN ignores some local information at each node in the graph, so that data preprocessing is incomplete and the model generated is not accurate enough. Thus, in the case of numerous unsupervised models based on graph embedding technology, local node information is important. In this paper, we apply a local analysis method based on the similar neighbor hypothesis to a GCN, and propose a local density definition; we call this method LDGCN. The LDGCN algorithm processes the input data of GCN in two methods, i.e., the unbalanced and balanced methods. Thus, the optimized input data contains detailed local node information, and then the model generated is accurate after training. We also introduce the implementation of the LDGCN algorithm through the principle of GCN, and use three mainstream datasets to verify the effectiveness of the LDGCN algorithm (i.e., the Cora, Citeseer, and Pubmed datasets). Finally, we compare the performances of several mainstream graph analysis algorithms with that of the LDGCN algorithm. Experimental results show that the LDGCN algorithm has better performance in node classification tasks.

Key words: Semi-supervised learning; Graph convolutional network; Graph embedding; Local density
https://doi.org/10.1631/FITEE.1900663 **CLC number:** TP391

1 Introduction

Graph convolutional network (GCN) is one important learning framework that enables feature representations of directed and undirected graphs (Lin and Cohen, 2010; Niepert et al., 2016). This network becomes increasingly popular in graph analysis applications, including link prediction, semi-supervised node classification, and recommendation. The adjacency and degree matrices of GCN graphs serve as model inputs for calculating a Laplacian matrix and training neuron parameters of GCN, while a convolutional neural network defines features of nodes or edges with multiple and continuous attributes in the graph (LeCun et al., 1989, 1998, 2015). To obtain the canonical representation of a non-Euclidean structure graph, the GCN algorithm calculates a Laplacian matrix for a graph matrix. A Laplacian matrix can be used to perform feature decomposition in the spectral domain and dimension reduction. This means that this approach can effectively reduce data dimensionality and cluster complexity. As local information at each node in the graph is ignored, however, Laplacian matrix features remain incomplete, which influences model formation.

In the struc2vec model (Ribeiro et al., 2017), nodes with similar adjacency and local topology are

‡ Corresponding author

* Project supported by the National Natural Science Foundation of China (Nos. 61272209 and 61872164)

ORCID: Hao WANG, https://orcid.org/0000-0002-9613-6169; Tie-hu FAN, https://orcid.org/0000-0003-1496-9464

generally considered to have similar characteristics. This means that similar adjacency and local topology are used to define and compute the structural identity of individuals in social networks (Lorrain and White, 1971; Pizarro, 2007; Narayanan et al., 2016). This model uses a function to evaluate the similarity of two nodes in the graph through their local topology, and then makes them closer in the sample sequence. These steps are iterated until convergence is achieved.

The aim of this study is to combine the similar neighbor hypothesis in social networks with a GCN to solve graph-based learning problems. We use the definition of a high-order degree matrix to explain the concept of local density, and expand this for defining graph nodes. We design a local density GCN (LDGCN) to obtain a practicable convolutional network model that can be used for classification and prediction tasks (Chen et al., 2018). The proposed algorithm uses just graph adjacency and high-order degree matrices, and then calculates a distinct Laplacian matrix. The matrix is used to map graph topological features and local information at each node to neural network parameters in the training steps. The novelty of LDGCN is that we can transform the degree matrix to a high-order one, while local density information is saved in the Laplacian matrix. This means that local node density information can be used in the similarity calculation, distinct from the similarity definition used in the struc2vec model (Fig. 1).

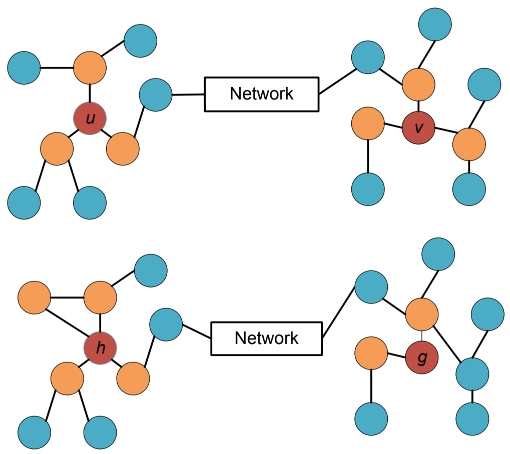

Fig. 1 LDGCN similarity definitions for two different social networks

The first- and second-order degree matrices are used for illustration in this study (Fig. 1). Thus, in the social network discussed here, nodes u and v are considered to be similar because their first- and second-order adjacencies are the same. When the first-order adjacencies between two nodes are the same, their second-order adjacencies will hardly be the same. The higher the order, the greater the difference between these two nodes. Nodes h and g in the social network are considered different. Although the sums of their first- and second-order adjacencies are the same, a gap nevertheless remains, because the low-order adjacency has greater effect on the target node than the high-order one (the first-order degree of node h is 4 and the first-order degree of node g is 2).

The goal of this study is to explore an optimization method that can be applied to GCN to enhance its performance. Therefore, a method for defining the graph representation using local density similarity is proposed. Given the interference of the higher-order degree matrices, it is easier to distinguish graph nodes. Several contributions are made:

1. An LDGCN algorithm based on the similar neighbor hypothesis in graph embedding technology is applied to a GCN. This algorithm means that nodes in the graph can be discriminated more effectively while the GCN algorithm performs at a higher level.

2. The algorithm used here renders the preprocessing of neural network data more complete even though the size of training data remains unchanged.

3. To calculate a high-order degree matrix based on the definition of local density, two LDGCN implementation methods, balanced and unbalanced, are applied. These approaches optimize the model and have no effect on training time.

2 Related work

In the earlier work, unsupervised network embedding technology (Wang et al., 2019) has been used to obtain low-dimensional, potential learning vector representations. These learned feature vectors can be applied in various social network settings, such as classification (Tang and Liu, 2011), link prediction, and visualization (Yang LM et al., 2015; Chen et al., 2018). Embedding techniques from natural language processing (NLP) have a long history; the skip-gram algorithm in NLP has been proposed to learn embedding vectors from words in text. This algorithm considers words with similar semantics in the text to form a class (Mikolov et al., 2013; Mnih and

Kavukcuoglu, 2013; Le and Mikolov, 2014). The method of learning feature representations of nodes or edges using a skip-gram algorithm in a social network was first proposed in the DeepWalk model (Perozzi et al., 2014). This model builds a sequence at each node in a graph using the random walk algorithm (Page et al., 1998; Langville and Meyer, 2006; Fouss et al., 2007), and then applies these sequences as the training corpus for the skip-gram algorithm. This enables local correlations between nodes in these walk sequences to be learned and their features to be defined. However, the random walk algorithm in the DeepWalk model is uncontrollable. The node2vec model (Grover and Leskovec, 2016) was therefore developed to optimize the random walk algorithm; this approach uses p and q to artificially control the depth and breadth of a random walk, respectively, and allows an unsupervised social network learning model to learn feature representations within different local ranges. p and q typically vary with a dataset or social network in different domains, and there are different optimal p and q values under different conditions. In the definition of general similarity, adjacent nodes on a graph are considered similarly. However, in the struc2vec model (Ribeiro et al., 2017), the similarity of two nodes is defined by analyzing the similarity of their local topologies. Graph embedding technology is generally used in large-scale social networks.

A semi-supervised learning method aims to deal with unlabeled data by analyzing labeled graph data. In this context, GCN is a popular semi-supervised learning approach that transforms a non-Euclidean graph into a Laplacian matrix for training purpose. As the properties of nodes and edges in this graph will be embodied in a Laplacian matrix, decomposed features can be obtained using feature decomposition of this matrix in the spectral domain (Shuman et al., 2013). Thus, GCN-based semi-supervised training can adapt to datasets in different fields (Lin and Cohen, 2010; Kipf and Welling, 2016).

3 Basic theory and the LDGCN algorithm

To introduce our algorithm, we initially outline the graph convolution principle. The LDGCN algorithm and its two implementations are explained in conjunction with mathematical logic and procedures.

3.1 Graph convolution principle

Feature decomposition in the spectral domain aims to provide a prerequisite for convolution operations on a non-Euclidean graph. The most important feature of a GCN is matrix feature decomposition.

In the case of graph G, the node and edge are denoted as V and E, respectively. If graph G has n nodes, V can be expressed as $V=\{v_1, v_2, \ldots, v_n\}$, and E is a set of node pairs. This means that the graph can be expressed as $G=(V, E)$, and the Laplacian matrix $\boldsymbol{L}$ has numerous definitions. We use the definition $\boldsymbol{L}=\boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2}$ to define the Laplacian matrix of the graph, where $\boldsymbol{A}$ and $\boldsymbol{D}$ denote the adjacency and degree matrices of graph G, respectively. Matrix $\boldsymbol{A}$ is an $n\times n$ matrix, and consists of 0 and 1, with the former indicating no link between two nodes and the latter indicating a link between two nodes. The degree matrix $\boldsymbol{D}$ represents the number of edges involved in each node in the graph. As $\boldsymbol{L}$ is a positive semidefinite matrix with n linearly independent feature vectors, it can be decomposed as follows:

$$\boldsymbol{L}=\boldsymbol{U}\begin{pmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_n \end{pmatrix}_{n\times n}\boldsymbol{U}^{-1}, \tag{1}$$

where $\boldsymbol{U}$ is a mapping matrix in the Fourier space and contains some graph features. This matrix comprises a series of column vectors $\boldsymbol{u}_l$, where $l\in\{1, 2, \ldots, n\}$. The matrix composed of λ in Eq. (1) is a diagonal matrix with n feature values. As the feature vectors of the Laplacian matrix (symmetric matrix) are orthogonal to each other and matrix $\boldsymbol{U}$ satisfies $\boldsymbol{U}\boldsymbol{U}^{\mathrm{T}}=\boldsymbol{E}$, spectral decomposition can be expressed as follows:

$$\boldsymbol{L}=\boldsymbol{U}\begin{pmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_n \end{pmatrix}_{n\times n}\boldsymbol{U}^{\mathrm{T}}. \tag{2}$$

Based on the feature vectors obtained using spectral decomposition of the Laplacian matrix, a Fourier transform based on the graph can be expressed as

$$\hat{g}(\lambda_l)=\sum_{i=1}^{n} g(i)u_l^*(i),\ l\in\{1, 2, \ldots, n\}, \tag{3}$$

where $g(\cdot)$ is used to obtain an attribute feature representation of each node; in a GCN, it takes the form of a matrix with n vectors. The length of these n vectors represents the dimension of the feature in each case, while $u_l(i)$ denotes the i^{th} component of the l^{th} feature vector, and $\boldsymbol{u}_l^*$ is the conjugate form of $\boldsymbol{u}_l$. Thus, in a matrix form, the Fourier transform in a graph can be expressed as

$$\begin{pmatrix} \hat{g}(\lambda_1) \\ \hat{g}(\lambda_2) \\ \vdots \\ \hat{g}(\lambda_n) \end{pmatrix} = \begin{pmatrix} u_1(1) & u_1(2) & \dots & u_1(n) \\ u_2(1) & u_2(2) & \dots & u_2(n) \\ \vdots & \vdots & & \vdots \\ u_n(1) & u_n(2) & \dots & u_n(n) \end{pmatrix} \begin{pmatrix} g(1) \\ g(2) \\ \vdots \\ g(n) \end{pmatrix}. \quad (4)$$

This relationship can be expressed as $\hat{\boldsymbol{g}}=\boldsymbol{U}^{\mathrm{T}}\boldsymbol{g}$, where $\hat{\boldsymbol{g}}$ denotes the normalized representation of each node in each dimension. Similarly, the inverse Fourier transform is $\boldsymbol{g}=\boldsymbol{U}\hat{\boldsymbol{g}}$. A convolution operation can therefore be performed based on this spectral decomposition. The Fourier transform of the neuron parameter h during the training process can be expressed as follows:

$$\hat{h}(\lambda_l) = \sum_{i}^{n} h(i) u_l^*(i). \quad (5)$$

In the GCN training process, the information contained in the graph will be continuously mapped to the neuron parameter h using matrix $\boldsymbol{U}$:

$$(\boldsymbol{g} * \boldsymbol{h})_{\mathrm{G}} = \boldsymbol{U} \begin{pmatrix} \hat{h}(\lambda_1) & & \\ & \ddots & \\ & & \hat{h}(\lambda_n) \end{pmatrix}_{n\times n} \boldsymbol{U}^{\mathrm{T}} \boldsymbol{g}. \quad (6)$$

As discussed above, the LDGCN algorithm uses a high-order degree matrix to calculate matrix $\boldsymbol{L}$; this means that the degree of similarity between each node in the graph is changed. This assumption also means that the vectors representing these nodes in matrix $\boldsymbol{U}$ vary. Thus, as matrix $\boldsymbol{U}$ is used for mapping, it will optimize the formation of neuron parameter h during each training process. Experimental results show that this change improves the performance of a GCN under certain conditions.

3.2 Definition and implementation of a high-order degree matrix and the unbalanced method

In a GCN model, based on the graph convolution theory, a simple training form of forward calculation can be expressed as

$$\boldsymbol{Z}=\mathrm{softmax}(\boldsymbol{L}\tanh(\boldsymbol{LXW}^0)\boldsymbol{W}^1), \quad (7)$$

where $\boldsymbol{W}^0$ is the parameter matrix of the input layer and $\boldsymbol{W}^1$ the parameter matrix of the hidden layer. $\boldsymbol{W}^0$ and $\boldsymbol{W}^1$ can be viewed as convolution kernels of CNN and are therefore constantly adjusted via training. $\boldsymbol{X}$ is the input to this layer.

The degree matrix $\boldsymbol{D}$ within the original GCN is denoted as $\boldsymbol{D}_1$. The second-order degree matrix for each node is therefore defined as $\boldsymbol{D}_2$, while the third-degree matrix is defined as $\boldsymbol{D}_3$. These higher-order degree matrices represent the densities of nodes at different ranges from the target in sequence. Thus, when calculating the high-order matrix, the first-order degree matrix $\boldsymbol{D}_1$ and others are multiplied by a coordination parameter α to adjust the level of the higher-order degree influence. Variables are set to different values depending on different social networks. Definition of the degree matrix used in the LDGCN algorithm is expressed as

$$\boldsymbol{D}=\boldsymbol{D}_1+\alpha_1\boldsymbol{D}_2+\alpha_2\boldsymbol{D}_3+\ldots+\alpha_{x-1}\boldsymbol{D}_x. \quad (8)$$

In general cases, only $\boldsymbol{D}_2$ is used because the use of higher-order matrices will bring greater cost (it is difficult to calculate their coordination parameters). Therefore, we provide an example of calculating the degree matrix $\boldsymbol{D}$ using the unbalanced method (Fig. 2). In this case, the calculation is divided into four steps, each step to calculate the result of a row. As the node represented by the third row is not connected to any other, there is no need to calculate the result in this case. Although the degree of growth in each row is unbalanced during the calculation process (e.g., modification of the fourth node degree is affected by changes in the first three nodes), this nevertheless enables a breakthrough that can be optimized. We therefore propose an extension method (balanced method); this calculation (Fig. 2) is performed in extreme cases, where the coordination parameter is 1, although in practical applications, coordination parameters are usually quite small. We have proved the optimization of the unbalanced and balanced methods through experiments, and found that the unbalanced method does not perform well compared with the balanced one.

As a result of the local density definition, when the node is closer to the target, its contribution to the target is greater. This means that in a general social network, only a second-degree matrix is usually used. A higher-order degree matrix (such as a third-order one) will decrease the efficiency and increase the time complexity of the model, and it is difficult to determine a higher-order coordination parameter. A high-order degree matrix means that matrix $\boldsymbol{L}$ contains more local information and that h formed in the training process is more accurate. Calculation steps for a third-order degree matrix are summarized in Algorithm 1.

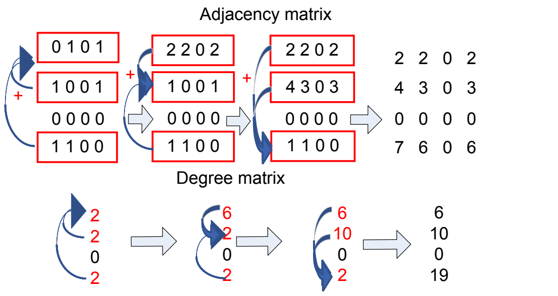

Fig. 2 Calculation of a second-order degree matrix graph with four nodes

Data in red boxes is summed in this case to obtain results. References to color refer to the online version of this figure

Algorithm 1 Third-order degree matrix computation

Input: adjacency matrix $\boldsymbol{A}$

Output: Laplacian matrix $\boldsymbol{L}$

1 $\boldsymbol{D}$=sum($\boldsymbol{A}$.row)
2 define $\boldsymbol{D}_2$ and $\boldsymbol{D}_3$
3 $\boldsymbol{D}_2$, $\boldsymbol{D}_3$=$\boldsymbol{D}$
4 **for** $\boldsymbol{A}_i$ in $\boldsymbol{A}$
5 **for** A_{ij} in $\boldsymbol{A}_i$
6 **if** nodes v_i and v_j are connected **do**
7 calculate $\boldsymbol{D}=\boldsymbol{D}+\alpha_1\boldsymbol{D}_2$
8 **for** A_{jk} in $\boldsymbol{A}_j$
9 **if** nodes v_j and v_k are connected **do**
10 calculate $\boldsymbol{D}=\boldsymbol{D}+\alpha_2\boldsymbol{D}_3$
11 **end if**
12 **end for**
13 **end if**
14 **end for**
15 **end for**
16 calculate $\boldsymbol{L}=\boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2}$
17 return matrix $\boldsymbol{L}$

In Algorithm 1, steps 8–12 are used to obtain third-order degree matrices but are not often applied in actual applications. $\boldsymbol{D}$ in line 1 in Algorithm 1 is the same as $\boldsymbol{D}_1$ in Eq. (8).

3.3 Balanced method based on a high-order degree matrix

As discussed above, high-order degree matrix calculation is implemented using the unbalanced method. However, although this approach has been found to be helpful in improving experimental accuracy, it is nevertheless unreasonable. Therefore, we sort the degree matrix to achieve a better standard to adapt to this approach. Indeed, as a part of the process of obtaining a high-order degree matrix (Fig. 2), it is clear that modification of subsequent node degrees will be affected by the previous ones. Therefore, we sort the degree matrix in descending and ascending orders, and verify and discuss these conditions experimentally.

The balanced method can adapt to the state due to high-order matrix calculation. This sorting method is realized by matrix index conversion (Algorithm 2).

In this approach, the sort() function is used to sort vector $\boldsymbol{D}$ and return a sorted index matrix $\boldsymbol{D}_{\_\mathrm{SI}}$ and a sorted degree matrix $\boldsymbol{D}_{\_\mathrm{SD}}$.

Algorithm 2 Matrix index conversion

Input: adjacency matrix $\boldsymbol{A}$

Output: $\boldsymbol{D}_{\_\mathrm{SI}}$ and $\boldsymbol{D}_{\_\mathrm{SD}}$

1 $\boldsymbol{D}$=sum($\boldsymbol{A}$.row)
2 define a sorted index matrix $\boldsymbol{D}_{\_\mathrm{SI}}$
3 define a sorted degree matrix $\boldsymbol{D}_{\_\mathrm{SD}}$
4 $\boldsymbol{D}_{\_\mathrm{SI}}$, $\boldsymbol{D}_{\_\mathrm{SD}}$=sort($\boldsymbol{D}$)
5 replace $\boldsymbol{D}$ in line 1 in Algorithm 1 with $\boldsymbol{D}_{\_\mathrm{SI}}$ and $\boldsymbol{D}_{\_\mathrm{SD}}$

Based on Algorithms 1 and 2, the LDGCN algorithm optimizes the Laplacian matrix $\boldsymbol{L}$ and matrix $\boldsymbol{U}$ on the graph by calculating a high-order degree matrix. This operation does not affect the size of matrix $\boldsymbol{U}$, and similarly does not affect GCN training complexity.

4 Experiments

Three datasets are used to evalute the performance of the LDGCN algorithm. As the GCN algorithm is applied mainly for community classification tasks, we conduct several experiments. We evalute the performance of this algorithm by setting different numbers of epochs; the most appropriate number should usually be included within the experimental scope, although a further experiment is conducted to determine the sensitivity to coordination parameters. We then compare the results of multiple mainstream algorithms and the LDGCN algorithm.

Experimental data is summarized in Table 1.

Table 1 Experimental datasets

Dataset	Number of nodes	Number of edges	Number of features	Number of labels	Label rate
Cora	2708	5429	1433	7	0.052
Citeseer	3327	4732	3703	6	0.036
Pubmed	19 717	44 338	500	3	0.003

4.1 Optimal coordination parameter

We first explore the optimal coordination parameters of different datasets because this approach is beneficial to another experiment. In our approach, the coordination parameters remain unchanged and are optimal. We find that the optimal prediction accuracies obtained by the original GCN applied to Cora, Citeseer, and Pubmed datasets are 0.815, 0.703, and 0.790, respectively (Kipf and Welling, 2016). Thus, we use the same settings; we set the number of epochs to 200 and use a two-layer neural network. The first layer uses the activation function tanh(·) (the activation function Relu is used in GCN), while the output dimension is 16 and the learning rate is set to 0.01. This setting is applied in another experiment; the prediction results from these three datasets under different coordination parameters are shown in Fig. 3.

Fig. 3 reveals that when the coordination parameter is 0, the original GCN algorithm can be used. Baselines in Fig. 3 indicate the results obtained; an improved GCN is used in the unbalanced method as presented in Section 3.2.

Experimental results show that different datasets have different optimal coordination parameters. In terms of scale, the Pubmed dataset is the largest, while the other two are similar in size and exhibit similar optimal coordination parameters. The optimal number of coordination parameters for the Pubmed dataset is much smaller than those for the other two datasets; therefore, it is not difficult to obtain just the coordination parameters of the second-order degree matrix, because with the change of α, the prediction accuracy curves of the model on the three datasets are continuous and have extreme values.

The optimal coordination parameters for the Cora, Citeseer, and Pubmed datasets are 0.030, 0.031, and 0.014, respectively. In the above experiment, Citeserer and Pubmed datasets achieve higher prediction accuracy compared with the original GCN. The prediction accuracy of the original GCN on the Cora dataset is 0.813 in our experiments.

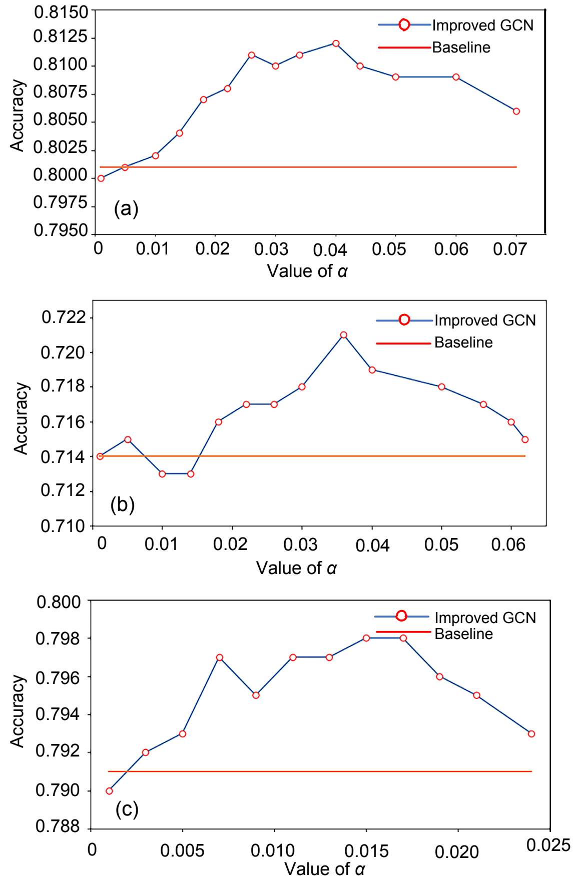

Fig. 3 Node classification prediction accuracy on the Cora (a), Citeseer (b), and Pubmed (c) datasets with changing α

4.2 Sensitivity to the epoch number

We use the optimal coordination parameters as mentioned in Section 4.1 in the experiments. To explain the effect of the epoch number, we set it to 100–1200 for the Cora and Citeseer datasets and 100–1000 for the Pubmed dataset. Experimental results are shown in Fig. 4.

Analysis of experimental results (Fig. 4) shows that the improved algorithm achieves a steady increase in the accuracy under different numbers of epochs. As the coordination parameters of the improved algorithm are optimal, the number of epochs does not affect the actual performance of the algorithm. As for the best performance of the improved algorithm and the original GCN, the former can perform better on the Cora and Citeseer datasets. Although the results obtained on the Cora dataset do not exceed those of the original GCN algorithm, the extension method outlined in Section 3.3 can be used for further optimization.

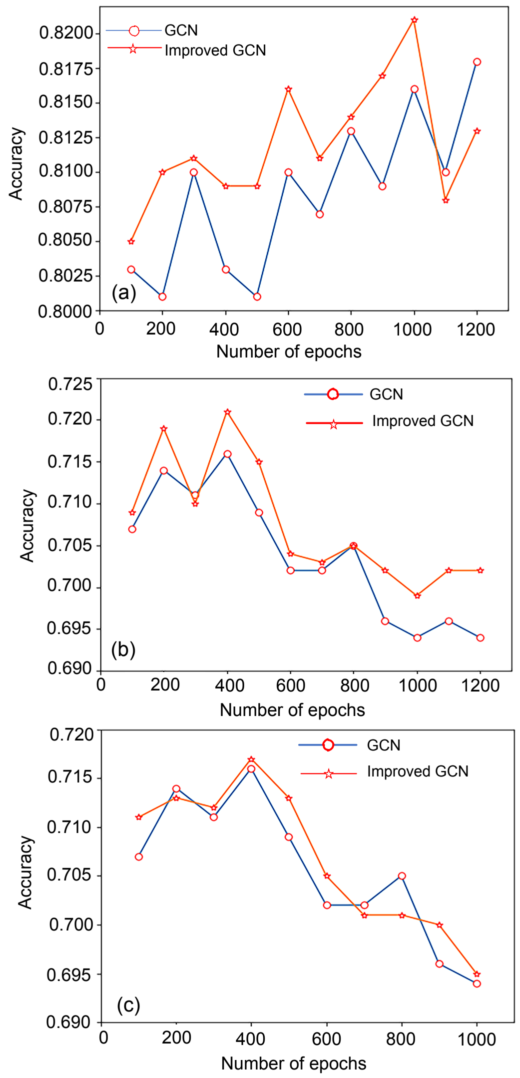

Fig. 4 Node classification prediction accuracy on the Cora (a), Citeseer (b), and Pubmed (c) datasets with different numbers of epochs

4.3 Exploratory experiment using the balanced method

As mentioned, we present only the key components of our experimental results obtained for ascending and descending sorting operations in the balanced method. Other parameters of this algorithm are consistent with those in the previous experiment. The results obtained using the balanced method are outlined in Tables 2–7.

Table 2 Experimental results based on the Cora dataset for the increasing sorting operation

α	Accuracy	α	Accuracy
0.02	0.806	0.05	0.812
0.03	0.811	0.06	0.810
0.04	0.810		

Table 3 Experimental results based on the Citeseer dataset for the increasing sorting operation

α	Accuracy	α	Accuracy
0.02	0.717	0.05	0.720
0.03	0.718	0.06	0.713
0.04	**0.721**		

The best result is in bold

Table 4 Experimental results based on the Pubmed dataset for the increasing sorting operation

α	Accuracy	α	Accuracy
0.005	0.792	0.020	0.794
0.010	**0.796**	0.060	0.791
0.015	0.794		

The best result is in bold

Table 5 Experimental results based on the Cora dataset for the descending sorting operation

α	Accuracy	α	Accuracy
0.040	0.810	0.070	0.811
0.050	0.814	0.080	0.809
0.064	**0.819**		

The best result is in bold

Table 6 Experimental results based on the Citeseer dataset for the descending sorting operation

α	Accuracy	α	Accuracy
0.02	0.716	0.05	0.718
0.03	0.718	0.06	0.707
0.04	**0.719**		

The best result is in bold

Table 7 Experimental results based on the Pubmed dataset for the descending sorting operation

α	Accuracy	α	Accuracy
0.005	0.791	0.020	0.792
0.010	0.794	0.025	0.793
0.016	**0.796**		

The best result is in bold

Based on the experimental results (Tables 2–7), it is clear that the descending sorting operation achieves more significant improvements when applied to the Cora dataset. Indeed, descending or ascending operations have little effect on the other two datasets. Optimal coordination parameters based on Cora and Citeseer datasets are similar, the reason of which, we believe, is related to the dataset size. Tables 2–7 show that the accuracy of the LDGCN (balanced) algorithm reaches a limit given a certain coordination parameter.

We conduct a comparative experiment using the LDGCN algorithm and five mainstream algorithms, including SemiEmb (Weston et al., 2012), DeepWalk (Perozzi et al., 2014), Planetoid (Yang ZL et al., 2016), GCN, and Graphite (Grover et al., 2018). Experimental results of these algorithms are taken from Kipf and Welling (2016) and summarized in Table 8.

Table 8 Comparison of our LDGCN algorithm and five mainstream algorithms in terms of classification accuracies

Method	Accuracy		
	Cora	Citeseer	Pubmed
SemiEmb	0.590	0.596	0.717
DeepWalk	0.672	0.432	0.653
Planetoid	0.757	0.647	0.619
GCN	0.815	0.703	0.790
Graphite	**0.821**	0.710	0.793
LDGCN	0.819	**0.721**	**0.798**

Best results are in bold

In summary, the method outlined in Section 3.2 has been validated by the experimental results in Sections 4.1 and 4.2. The experimental results in Section 4.3 confirm the effectiveness of the extension method given in Section 3.3.

5 Conclusions

The proposed algorithm uses a graph degree matrix as its basis to increase the neural network training efficiency. However, there is one obvious problem; when using different orders, the smaller the improvement of the algorithm, the more time it takes to calculate the variable. It is generally the case that only the second order is usable.

Due to the application-based algorithm structure of GCN, this approach is used mainly for social network community division. As a semi-supervised algorithm, it is usually necessary to understand parts of the community information and train associated network parameters. In numerous graph embedding technologies, network learning topologies are stricter; this means that models like DeepWalk and node2vec focus more on local graph structures. In the Laplacian matrix calculation process of the GCN algorithm, data items are related to the local structure of the graph, so local density values can be added to the Laplacian matrix. This improvement means that the focus of a GCN can move slightly into the graph structure.

Contributors

Hao WANG designed the research, processed the data, and drafted the manuscript. Li-yan DONG, Tie-hu FAN, and Ming-hui SUN helped organize the manuscript. Hao WANG and Tie-hu FAN revised and finalized the paper.

Compliance with ethics guidelines

Hao WANG, Li-yan DONG, Tie-hu FAN, and Ming-hui SUN declare that they have no conflict of interest.

References

Chen HC, Perozzi B, Al-Rfou R, et al., 2018. A tutorial on network embeddings. https://arxiv.org/abs/1808.02590v1

Fouss F, Pirotte A, Renders JM, et al., 2007. Random-walk computation of similarities between nodes of a graph with application to collaborative recommendation. *IEEE Trans Knowl Data Eng*, 19(3):355-369. https://doi.org/10.1109/tkde.2007.46

Grover A, Leskovec J, 2016. node2vec: scalable feature learning for networks. Proc 22nd ACM SIGKDD Int Conf on Knowledge Discovery and Data Mining, p.855-864. https://doi.org/10.1145/2939672.2939754

Grover A, Zweig A, Ermon S, 2018. Graphite: iterative generative modeling of graphs. https://arxiv.org/abs/1803.10459v2

Kipf TN, Welling M, 2016. Semi-supervised classification with graph convolutional networks. https://arxiv.org/abs/1609.02907

Langville AN, Meyer CD, 2006. Google's PageRank and Beyond: the Science of Search Engine Rankings. Princeton University Press, Princeton, USA, p.234.

Le Q, Mikolov T, 2014. Distributed representations of sentences and documents. Proc 31st Int Conf on Machine Learning, p.II-1188-II-1196.

LeCun Y, Boser B, Denker JS, et al., 1989. Backpropagation applied to handwritten zip code recognition. *Neur Comput*, 1(4):541-551. https://doi.org/10.1162/neco.1989.1.4.541

LeCun Y, Bottou L, Bengio Y, et al., 1998. Gradient-based learning applied to document recognition. *Proc IEEE*, 86(11):2278-2324. https://doi.org/10.1109/5.726791

LeCun Y, Bottou Y, Hinton G, 2015. Deep learning. *Nature*, 521(7553):436-444. https://doi.org/10.1038/nature14539

Lin F, Cohen WW, 2010. Semi-supervised classification of network data using very few labels. Int Conf on Advances in Social Networks Analysis and Mining, p.192-199. https://doi.org/10.1109/ASONAM.2010.19

Lorrain F, White HC, 1971. Structural equivalence of individuals in social networks. *J Math Soc*, 1(1):49-80. https://doi.org/10.1080/0022250X.1971.9989788

Mikolov T, Chen K, Corrado G, et al., 2013. Efficient estimation of word representations in vector space. https://arxiv.org/abs/1301.3781

Mnih A, Kavukcuoglu K, 2013. Learning word embeddings efficiently with noise-contrastive estimation. Proc 26th Int Conf on Neural Information Processing Systems, p.2265-2273.

Narayanan A, Chandramohan M, Chen LH, et al., 2016. subgraph2vec: learning distributed representations of rooted sub-graphs from large graphs. https://arxiv.org/abs/1606.08928

Niepert M, Ahmed M, Kutzkov K, 2016. Learning convolutional neural networks for graphs. Proc 33rd Int Conf on Machine Learning, p.2014-2023.

Page L, Brin S, Motwani R, et al., 1998. The Pagerank Citation Ranking: Bringing Order to the Web. Technical Report SIDL-WP-1999-0120, Stanford InfoLab, Stanford, USA.

Perozzi B, Al-Rfou R, Skiena S, 2014. DeepWalk: online learning of social representations. Proc 20th ACM SIGKDD Int Conf on Knowledge Discovery and Data Mining, p.701-710. https://doi.org/10.1145/2623330.2623732

Pizarro N, 2007. Structural identity and equivalence of individuals in social networks: beyond duality. *Int Soc*, 22(6):767-792. https://doi.org/10.1177/0268580907082260

Ribeiro LFR, Saverese PHP, Figueiredo DR, 2017. struc2vec: learning node representations from structural identity. Proc 23rd ACM SIGKDD Int Conf on Knowledge Discovery and Data Mining, p.385-394. https://doi.org/10.1145/3097983.3098061

Shuman DI, Narang SK, Frossard P, et al., 2013. The emerging field of signal processing on graphs: extending high-dimensional data analysis to networks and other irregular domains. *IEEE Signal Process Mag*, 30(3):83-98. https://doi.org/10.1109/MSP.2012.2235192

Tang L, Liu H, 2011. Leveraging social media networks for classification. *Data Min Knowl Discov*, 23(3):447-478. https://doi.org/10.1007/s10618-010-0210-x

Wang HF, Zhang CY, Lin DY, et al., 2019. An artificial intelligence based method for evaluating power grid node importance using network embedding and support vector regression. *Front Inform Technol Electron Eng*, 20(6): 816-828. https://doi.org/10.1631/FITEE.1800146

Weston J, Ratle F, Mobahi H, et al., 2012. Deep learning via semi-supervised embedding. In: Montavon G, Orr GB, Müller KR (Eds.), Neural Networks: Tricks of the Trade. Springer, Berlin, Heidelberg, p.639-655. https://doi.org/10.1007/978-3-642-35289-8_34

Yang LM, Zhang W, Chen YF, 2015. Time-series prediction based on global fuzzy measure in social networks. *Front Inform Technol Electron Eng*, 16(10):805-816. https://doi.org/10.1631/FITEE.1500025

Yang ZL, Cohen WW, Salakhutdinov R, 2016. Revisiting semi-supervised learning with graph embeddings. Proc 33rd Int Conf on Machine Learning, p.40-48.