Efficient Nodes Representation Learning with Residual Feature Propagation

Fan Wu[1], Duantengchuan Li[2(✉)], Ke Lin[3], and Huawei Zhang[1]

[1] School of Computer Science and Technology, Wuhan University of Technology, Wuhan 430070, China
[2] National Engineering Research Center for E-Learning, Central China Normal University, Wuhan 430079, China
[3] Department of Control Science and Engineering, Harbin Institute of Technology Shenzhen, Shenzhen 518055, China

Abstract. Graph Convolutional Networks (GCN) and their variants have achieved brilliant results in graph representation learning. However, most existing methods cannot be utilized for deep architectures and can only capture the low order proximity in networks. In this paper, we have proposed a Residual Simple Graph Convolutional Network (RSGCN), which can aggregate information from distant neighbor node features without over-smoothing and vanishing gradients. Given that node features of the same class have certain similarity, a weighted feature propagation is considered to ensure effective information aggregation by giving higher weights to similar neighbor nodes. Experimental results on several datasets of node classification demonstrate the proposed methods outperform the state-of-the-art methods in terms of effectiveness and efficiency.

Keywords: Graph convolutional networks · Graph representation learning · Feature propagation · Node classification

1 Introduction

The goal of graph representation learning is to represent nodes on the graph by low-dimensional dense vectors while maintaining the property characteristics of nodes and the structural features of graphs. Graph convolutional networks (GCN) [5], a variant of Convolutional Neural Networks (CNNs), have shown efficacious performance in graph representation learning. GCN can learn appropriate node representation by aggregating neighbor node information. Moreover, in order to capture the high-order similarity of nodes, a non-linear transformation is introduced in each layer of GCN propagation [8,17]. Recently, GCN have been widely utilized in graph structure data researches, such as node classification [9], node clustering [21], graph classification [10], and link prediction [6]. In addition, researchers have successfully applied GCN and subsequent variants to their application areas, such as knowledge graph [13], computer vision [11], natural language processing [18], and recommendation system [19].

K. Karlapalem et al. (Eds.): PAKDD 2021, LNAI 12713, pp. 156–167, 2021.
https://doi.org/10.1007/978-3-030-75765-6_13

In GCN, because each layer of graph convolution needs to aggregate features from the connected node, the dependence relationship between nodes should be known before model training. This makes the optimization method of min-batch no longer applicable to GCN, which will make GCN training very difficult.

Considering these limitations of GCN, many researchers have made some improvements to solve the above problems. In [2], the authors introduced GraphSAGE, a general inductive manner for learning node representation on large graph structure data. This method randomly sampled a fix-sized neighborhood for each node and aggregated node features from this neighborhood by a specific aggregator. Moreover, in order to resolving dependence relationship between nodes, Zeng *et al.* [20] constructed mini-batch by sampling the training graph and built a complete GCN on the sampled subgraph for each iteration. Although the large graph structure data can be processed by these methods, it is hard to stack more layers to obtain high-order node information.

Inspired by the great success of residual connections, dense connections and dilated convolution in deep learning, Li *et al.* [7] adapted these ideas into GCN to solve the vanishing gradients problem and proposed Deep Graph Convolutional Networks (DeepGCNs). Although DeepGCNs can extract deeper node information in the graph and have several advantages over previous methods. Unfortunately, it consumes bulky computing resources and prodigious time in the inference process, which means its application to large graph structure data would be difficult. The large graph structure data are very common in practical applications. However, previous works fail to efficiently aggregate deeper node information and separate dependence relationship between nodes during training processes in large graph structure data.

To build a high-efficiency graph representation learning model and separate dependence of nodes during training processes, a Residual Simple Graph Convolutional NetWork (RSGCN) by removing the non-linear activation function of DeepGCNs is proposed. In RSGCN, residual feature propagation enables the model to learn higher order node information and restrain the over-smoothing of the graph. Furthermore, as average aggregation confuses the importance of different classes to nodes itself, we propose a weighted feature propagation model RSGCN+ to learn the important information from similar nodes. RSGCN+ ensures effective information aggregation by giving higher weights to similar neighbor nodes, which is measured by the cosine similarity between node features. Finally, our models can learn accurate node representation. The major contributions of this paper are summarized as follows.

– A Residual Simple Graph Convolutional Network (RSGCN) is proposed by removing the non-linear activation function of DeepGCNs. With the residual feature propagation, RSGCN can aggregate information from distant neighbor node features without over-smoothing and vanishing gradients. More importantly, RSGCN can achieve high effectiveness and efficiency during training process.
– Given that node features of the same class have certain similarity, we propose a weighted feature propagation model RSGCN+ to ensure effective

information aggregation by giving higher weights to similar neighbor nodes, which further improves the node representation and the robustness of the model.
– To verify the performance of the proposed methods, three standard benchmark datasets for citation networks are taken as the comparing experiments. The results demonstrate that our models obtain significant improvements for the semi-supervised node classification tasks in the terms of both prediction accuracy and the training efficiency.

2 Preliminaries and Related Work

2.1 Primary Definition

Given an undirected attributed graph $\mathcal{G} = (\mathcal{V}, \mathbf{A})$, where $\mathcal{V} = \{\nu_i\}_{i=1,\ldots,n}$ represents the nodes and $\mathbf{A} = \{a_{ij}\} \in \mathbb{R}^{n\times n}$ is the adjacency matrix of the graph $\mathcal{G}$. If there is an edge between node v_i and node v_j, then $a_{ij} = 1$, otherwise it equals to 0. For ease of notation, the neighbor set of node v_i can be denoted as $\mathcal{A}_i = [j|a_{ij} = 1]$. Note that $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ denotes the adjacency matrix $\mathbf{A}$ with self-loops and the degree matrix $\tilde{\mathbf{D}} = \text{diag}\{d_1, d_2, ..., d_n\} \in \mathbb{R}^{n\times n}$ is a diagonal matrix where the i-th value on the diagonal $d_i = \sum_j \tilde{a}_{ij}$ is equal to the degree of the i-th node of matrix $\tilde{\mathbf{A}}$. For the semi-supervised node classification tasks, we observe the labels of a subset of the nodes in the graph $\mathcal{G}$. The goal of node classification is to predict the unknown node labels based on the graph structure and node features we known the labels.

2.2 Graph Convolutional Network

For each node $v_i \in \mathcal{V}$, h_i^0 represents initial node representation, which is d-dimensional feature vector $x_i \in \mathbb{R}^d$. Then, GCN can learn node representation for each node based on node initial features and graph structure. Specifically, for each node v_i in the graph convolution layer, the node representation is updated recursively with the following three steps: feature propagation, linear transformation, and non-linear activation.

Feature Propagation. For each node v_i, the feature propagation step aggregates the node information from node itself representation h_i^k at previous layer k and graph neighbors $\mathcal{A}_i$,

$$h_i{}^{(k+1)} = \frac{1}{d_i+1} h_i{}^{(k)} + \sum_{j=1}^{n} \frac{a_{ij}}{\sqrt{(d_i+1)(d_j+1)}} h_j{}^{(k)} \tag{1}$$

where d_i denotes the degree of node v_i. Besides, the update of entire graph can be expressed as a simple matrix operation. The symbol $\mathbf{S} = \tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}$

represents the "normalized" adjacency matrix with added self-loops. Thus, the update process in Eq. (1) for all nodes can be expressed as,

$$\bar{\mathbf{H}}^{(k+1)} = \mathbf{S}\mathbf{H}^{(k)} \tag{2}$$

Intuitively, this step makes each node aggregate information from connected node and eventually has a positive influence on node classification tasks. Theoretically, feature propagation output layer is regarded as the Laplacian smoothing of the node features at the previous layer [8,15].

Linear Transformation and Non-linear Activation. After feature propagation, linear transformation and non-linear activation is identical to a standard multilayer perceptron. In a GCN layer, there is a learned weight $\mathbf{W}^k$ as linear transformation after the feature propagation, which can transform node representation linearly. Finally, a non-linear activation produces the node representation of the $(k+1)$-th layer as,

$$\mathbf{H}^{(k+1)} = \sigma\left(\bar{\mathbf{H}}^{(k+1)}\mathbf{W}^k\right) \tag{3}$$

where $\sigma(\cdot)$ is a non-linear activation function.

2.3 Simplifying Graph Convolutional Network

Recently, considerable literature has grown up around the theme of simplifying GCN in order to reduce training time and memory. A Simple Graph Convolutional Network (SGCN) is proposed [16], which removes the non-linear activation function in Eq. (3) as,

$$\mathbf{H}^{(k)} = \mathbf{S}\mathbf{S}\ldots\mathbf{S}\mathbf{H}^{(0)}\mathbf{W}^0\mathbf{W}^1\ldots\mathbf{W}^k \tag{4}$$

where $\mathbf{W}^0\mathbf{W}^1\ldots\mathbf{W}^k$ can be rewritten as a single matrix $\mathbf{W}$ and the repeated multiplication with the matrix $\mathbf{S}$ can be simplified to a single matrix $\mathbf{S}^k$. The above linear matrix multiplication turns to,

$$\mathbf{H}^{(k)} = \mathbf{S}^k\mathbf{H}^{(0)}\mathbf{W} \tag{5}$$

With the simplification of SGCN, k times feature propagation $\mathbf{S}^k$ can be calculated before training, and the parameters are much less than GCN, which makes it easy to apply SGCN to large graph structure data. Many experiments show that removing the non-linear activation function in GCN does not have a negative impact on performance in many graph tasks. However, [16] shows that SGCN has the best node classification performance at feature propagation depth of 2 or 3. When feature propagates for too many times, the node representation information propagated to well-connected node rapidly increase. This leads to the over-smoothing issue, which means the features of each node are mixed by too many neighbors and lose locality.

2.4 Deep Graph Convolutional Networks

In GCN, the depth has a crucial function: after k layers each node can aggregate feature information from the nodes that are k-hops away in the graph. However, GCN with deep layers will lead to vanishing gradients, which makes accuracy drop sharply in classification tasks. Inspired by the success of the Deep CNNs technology, DeepGCNs [7] employed residual/dense connections to solve the above problem.

ResNet [3] can alleviate the problems of vanishing gradients and network degradation caused by increasing depth in deep neural networks. The node representation of the $(k+1)$-th layer in ResGCN can be defined as:

$$\mathbf{H}_{res}^{k+1} = \sigma\left(\mathbf{S}\mathbf{H}^{(k)}\mathbf{W}^{k}\right) + \mathbf{H}^{(k)} \tag{6}$$

where $\mathbf{W}^k$ has the same dimension as $\mathbf{H}^{(0)}$. Although DeepGCNs can effectively stack more layers, and the performance does not decline severely with depth increasing like GCN, it consumes abundant computing resources and prodigious time in the training process. Thus, it is difficult to apply it to large graph structure data.

3 Our Proposed Methods

In this section, we propose Residual Simple Graph Convolutional Network (RSGCN), a model of node representation learning that extracts deep node information. The overall architecture of the proposed models is shown in Fig. 1, which can be summed as two processes: (1) For mitigating over-smoothing, we propose residual feature propagation RSGCN (dashed-blue) to retain more node itself information. (2) On the basis of residual feature propagation, we adjust the final node features by adding weighted feature propagation RSGCN+ (dashed-red).

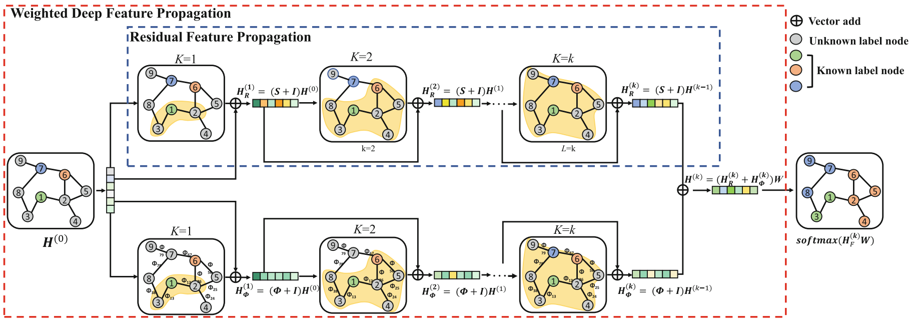
Weighted Deep Feature Propagation
Residual Feature Propagation
Vector add
Unknown label node
Known label node
K=1
K=2
K=k
k=2
L=k
$H_R^{(1)} = (S+I)H^{(0)}$
$H_R^{(2)} = (S+I)H^{(1)}$
$H_R^{(k)} = (S+I)H^{(k-1)}$
$H^{(0)}$
$H^{(k)} = (H_R^{(k)} + H_\Phi^{(k)})W$
$softmax(H_F^{(k)}W)$
$H_\Phi^{(1)} = (\Phi+I)H^{(0)}$
$H_\Phi^{(2)} = (\Phi+I)H^{(1)}$
$H_\Phi^{(k)} = (\Phi+I)H^{(k-1)}$

Fig. 1. Outline of our models framework.

3.1 Residual Feature Propagation

Considering that non-linear activation functions have almost no benefit in the node representation, we can simplify ResGCN by removing the non-linear activation functions. Hence, Eq. (6) can become as follows:

$$\mathbf{H}_R^{(k+1)} = \mathbf{S}\mathbf{H}_R^{(k)}\mathbf{W}^k + \mathbf{H}_R^{(k)} \tag{7}$$

In order to better mine the node feature information and simplify the model, we move the linear transformation to the end of each layer, so Eq. (7) could be changed to Eq. (8).

$$\mathbf{H}_R^{(k+1)} = (\mathbf{S} + \mathbf{I})\,\mathbf{H}_R^{(k)}\mathbf{W}^{k'} \tag{8}$$

where $\mathbf{I} \in \mathbb{R}^{n \times n}$ denotes the identity matrix. The node representation of the k-th layer can be defined as:

$$\mathbf{H}_R^{(k)} = (\mathbf{S} + \mathbf{I})\,(\mathbf{S} + \mathbf{I}) \ldots (\mathbf{S} + \mathbf{I})\,\mathbf{H}^{(0)}\mathbf{W}^{0'}\mathbf{W}^{1'} \ldots \mathbf{W}^{k'} \tag{9}$$

where $\mathbf{W}^{0'}\mathbf{W}^{1'} \ldots \mathbf{W}^{k'}$ can be rewritten as a single matrix W and the repetitive multiplication operation of the matrix $\mathbf{S} + \mathbf{I}$ can be simplified to a single matrix $(\mathbf{S} + \mathbf{I})^k$. The node representation of residual feature propagation can be defined as:

$$\mathbf{H}_R^{(k)} = (\mathbf{S} + \mathbf{I})^k\,\mathbf{H}^{(0)}\mathbf{W} \tag{10}$$

In residual feature propagation, their node features of inputs are added to the inputs of the next feature propagation, which means that node features can be well preserved. In this way, RSGCN enables more feature propagation, which can aggregate information from more distant neighbor nodes with weaker over-smoothing impact. In addition, as residual feature propagation can be calculated before training, the scale of parameters in RSGCN is lessened and the training efficiency is raised vastly. Thus, the matrix $\mathbf{H}_R^{(k)}$ can be expanded as:

$$\mathbf{H}_R^{(k)} = \left(\mathbf{S}^k + C_k^1\mathbf{S}^{k-1} + \ldots + \mathbf{I}\right)\mathbf{H}^{(0)}\mathbf{W} \tag{11}$$

In general, lower order neighbor nodes contain more important information, whereas higher order neighbor nodes may contain some noisy information. In addition, $\mathbf{S}^i\mathbf{H}^{(0)}$ contains the information about the 1 to i-hop neighbors node features and initial node features. Equation (11) represents that the more distant neighbor node features are given smaller weights, which enables the node to aggregate less noisy information.

3.2 Weighted Feature Propagation

Currently, most graph neural networks use mean aggregation to learn node representation. Valid information and noise are treated equally, which may hurt the performance of models. The graph attention network [14] introduces the attention mechanism into the GCN by assigning a learned weight parameter for neighbor nodes of each node. The huge performance improvement in node classification tasks illustrates that assigning a suitable weight to neighbor node is a better way of feature propagation. However, the attention mechanism significantly increases parameters of model, thus it is difficult to apply it to large graph structure data. Therefore, we can change our mind to consider assigning a weight to neighbor nodes based on their initial features before training.

On condition that the node features of the same class have more similarity, cosine similarity is utilized as a criterion for determining the similarity of two node features. The cosine similarity matrix $\mathbf{\Theta}$ can be defined as:

$$\mathbf{\Theta}_{ij} = a_{ij} \cdot \frac{\sqrt{\sum_{p=1}^{d} x_p y_p}}{\sqrt{\sum_{p=1}^{d} x_p}\sqrt{\sum_{p=1}^{d} y_p}} \tag{12}$$

where x_p is the p-th feature of v_i and y is the p-th feature of v_j. In order to balance the cosine similarity scale, we normalize them by using the softmax function by row. Hence, the node weight matrix can be defined as:

$$\mathbf{\Phi}_{ij} = \frac{\exp(\mathbf{\Theta}_{ij})}{\sum_{j=1}^{n} \exp(\mathbf{\Theta}_{ij})} \tag{13}$$

In order to retain more information from the node itself features, we borrowed the idea of residual feature propagation into the weighted feature propagation. The weighted feature propagation can be defined as:

$$\mathbf{H}_{\Phi}^{(k)} = (\mathbf{\Phi} + \mathbf{I})^{k}\,\mathbf{H}^{(0)}\mathbf{W} \tag{14}$$

By using residual feature propagation to obtain the final node features, RSGCN can learn part of the useful information from the node features and the graph structure. In addition, weighted feature propagation can extract further useful information about neighbor nodes and reduce the influence of irrelevant neighbor node. This information may contain some information that is not contained in the residual feature propagation. In order to preserve the useful features of both two feature propagation, we merge them in a stacked manner. Therefore, the final weighted feature propagation can be defined as:

$$\mathbf{H}_{F}^{(k)} = \mathbf{H}_{R}^{(k)} + \mathbf{H}_{\Phi}^{(k)} \tag{15}$$

The weighted feature propagation can assign a weight to neighbor node based on their similarity to node itself features. Although weighted feature propagation increases some memory to some extent, the neighbor node features can be aggregated more efficiently and rationally. In addition, weight feature propagation can be completed before training, and weighted feature propagation can be performed separately for each node, which is ideal for large graph structure data.

3.3 Classifier

Similar to common classification tasks, we can use a *softmax* function as a classifier after feature propagation and linear transformation. For a node classification task with C classes, the class prediction $\hat{\mathbf{Y}} \in \mathbb{R}^{n \times C}$ in RSGCN of k times feature propagation can be defined as:

$$\hat{\mathbf{Y}} = \text{softmax}\left(\mathbf{H}_F^{(k)}\right) \tag{16}$$

where $\text{softmax}(x) = \exp(x)/\sum_{c=1}^{C}\exp(x_c)$. For multi-class node classification tasks, we generally take cross entropy as the loss function.

4 Experiments and Discussions

4.1 General Setting

Datasets. Cora, Citeseer, and Pubmed [5] are employed to evaluate the semi-supervised node classification task, which are the standard benchmark datasets for citation networks. The statistics of datasets are summarized in Table 1. The above dataset composed of diverse scientific publications are classified into different classes. Each publication in the dataset is described by a 0/1-valued word vector indicating the absence/presence of the corresponding word from the dictionary. And the edges in the datasets represent the citation relationship between articles. In order to obtain unbiased and objective results, we have leveraged 10%–20%–70% train-validation-test settings.

Comparison Algorithms. We compared the proposed RSGCN and RSGCN+ with many state-of-the-art methods, including DeepWalk [12], GCN [5], SGCN [16], FastGCN [1], GraphSAGE [2], and DeepGCNs [7]. Since GraphSAGE and DeepGCNs have a variety of models, we choose GraphSAGE-mean and ResGCN with good effects as representatives.

Experimental Implementation. The parameters of compared methods are adjusted as the suitable ones according to their papers. On all citation networks datasets, RSGCN is trained for 200 epochs using Adam optimizer [4] with learning rate 0.2. And the setting of hyper parameters like the feature propagation depth and weight decay are manually adjusted according to the validation set results. We select the model with the best performance of validation sets during the training to test the performance of test sets. RSGCN+ has the same parameters setting as RSGCN.

Table 1. Dataset statistics of the citation networks

Dataset	Cora	Citeseer	Pubmed
#Nodes	2708	3327	19717
#Edges	5429	4732	44338
#Features	1433	3703	500
#Classes	7	6	3

Table 2. Test Micro-F1 Score (%) averaged over 10 runs. The best and second values are marked by the bold font and underlines.

Method	Cora	Citeseer	Pubmed
DeepWalk [12]	73.51	55.06	79.36
GCN [5]	83.01	72.03	<u>86.41</u>
SGCN [16]	83.35	71.71	85.60
FastGCN [1]	80.36	70.15	85.42
GraphSAGE [2]	81.26	71.30	85.63
ResGCN [7]	82.85	71.94	86.30
RSGCN	<u>84.06</u>	<u>73.06</u>	86.33
RSGCN+	**85.10**	**74.06**	**86.95**

4.2 Results and Discussion

Performance. For accuracy comparison of DeepWalk, GCN, SGCN, FastGCN, GraphSAGE, ResGCN, RSGCN on all three datasets, the highest Micro-F1 of each model are summarized in Table 2. Table 2 shows that the performance of RSGCN is superior to GCN and its variants on the citation networks. In particular, on the Cora and Citeseer datasets, RSGCN has 1% improvement in Micro-F1 score than GCN. On the Pubmed dataset, RSGCN has similar performance to GCN result. The improvement of RSGCN performance comes from two aspects. On the one hand, RSGCN can propagate feature more times than GCN, which allows each node to aggregate feature information from more distant neighbor nodes. On the other hand, RSGCN has fewer parameters compared to GCN. This means that RSGCN has a strong generalization capability and suffers less from overfitting. Furthermore, RSGCN+ achieve the higher Micro-F1 score than RSGCN on the citation datasets. It proves that the RSGCN+ is a high-performance graph model and has the capability to enhance model effectiveness by weighted feature propagation.

Efficiency. In Table 3, we show the time to train comparison methods and our models for 200 epochs on the citation networks and the number of layers is set to be 2 for all models. In particular, RSGCN and RSGCN+ take into account the time of residual feature propagation and weighted feature propagation. The training time is measured by a PC Server equipped with an Intel(R) Xeon(R) CPU E5-2620 V4 @2.10GHz, NVIDIA TITAN V, and 64 GB RAM.

Table 3 shows RSGCN is faster than comparison methods. RSGCN achieves 80.2%/75.9%/78.3% improvement of time in training of the Cora/Citeseer/Pubmed dataset than GCN. As for other methods besides SGCN, feature propagation in each epoch with enormous parameters make training inefficient. Since SGCN with only one learned parameter matrix performs less than satisfactory, the providing source code uses two learned parameters matrix to obtain accurate classification performance. However, our models perform well

Table 3. Training time (seconds) on citation networks averaged over 10 runs. The values of brackets represent performance improvement compared to GCN method.

Method	Cora	Citeseer	Pubmed
GCN [5]	3.99	4.15	4.51
SGCN [16]	1.24	1.73	1.74
FastGCN [1]	2.15	2.32	2.63
GraphSAGE [2]	8.35	8.79	9.12
ResGCN [7]	12.64	46.15	21.78
RSGCN	0.79 (↑ 80.2%)	1.00 (↑ 75.9%)	0.98 (↑ 78.3%)
RSGCN+	2.07 (↑ 52.0%)	3.15 (↑ 24.1%)	4.08 (↑ 9.5%)

using one learned matrix and therefore faster than SGCN. RSGCN+ consumes more time due to calculating cosine similarity, which is still faster than GCN.

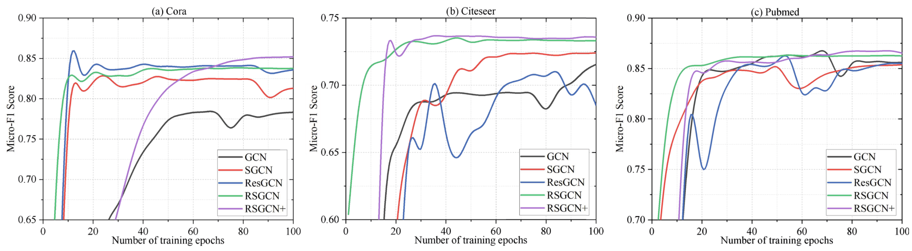

Fig. 2. Training processes of all models compared with Micro-F1 score on (a) Core, (b) Citeseer, and (c) Pubmed.

Because FastGCN and GraphSAGE will be affected by random sampling, the training has greater volatility, so they are not recorded in Fig. 2 and Fig. 3. The Micro-F1 score at a training process is depicted in Fig. 2. Figure 2 illustrates the relationship between the Micro-F1 score and the epoch on the Cora, Citeseer, and Pubmedand datasets. One can see that the proposed RGSCN and RSGCN+ not only achieve the highest Micro-F1 score in the validation set, but also require fewer epochs to converge than traditional GCN. During the training process, RSGCN and RSGCN+ demonstrate high efficiency, which shows good industrial conversion application prospects.

Training Depth Analysis. Figure 3 shows the performance in test sets of GCN, SGCN, ResGCN, RSGCN, RSGCN+ measured by Micro-F1 score with different depth on three citation datasets. For the case of 1 to 3 depth, the Micro-F1 score of above methods increases with more layers added, which suggests that deeper feature propagation may be useful. From Fig. 3, RSGCN achieve the

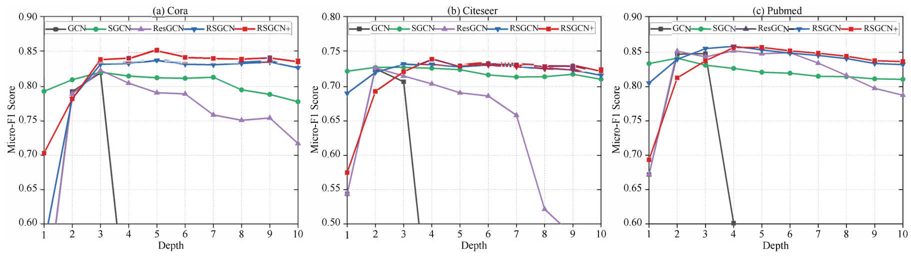

Fig. 3. Performance in test sets of five models measured by Micro-F1 score with different depth on (a) Core, (b) Citeseer, and (c) Pubmed.

best classification performance at depth between 4 and 6, while others achieves the best classification performance at depth between 2 and 3. Due to gradient vanishing problem caused by the deep network and the over-smoothing caused by the feature propagation, GCN performance decreases sharply at depth of 4. In addition, the performance of ResGCN also starts to decrease sharply at depth of 7 because enormous parameters in the ResGCN lead to over-fitting. With increasing depth of model, the effect of SGCN, RSGCN, and RSGCN+ on classification performance is less pronounced. This is largely due to the fact that SGCN, RSGCN, and RSGCN+ have fewer parameters and not over-fitting. Due to the slower feature convergence, the performance of RSGCN in shallow layers is slightly inferior to other models. However, the performance of RSGCN, and RSGCN+ is still better than SGCN. An explanation is that the residual feature propagation in RSGCN can effectively slow down the smoothness, which also gives RSGCN some edge in depth.

5 Conclusion

In this paper, we have proposed a Residual Simple Graph Convolutional Network (RSGCN), which can aggregate information from distant neighbor node features without over-smoothing and vanishing gradients. Given that node features of the same class have certain similarity, a weighted feature propagation is considered to ensure effective information aggregation by giving higher weights to similar neighbor nodes. Experimental results indicate that the proposed method performs better than compared methods on both accuracy and training efficiency in terms of quantitative assessments.

References

1. Chen, J., Ma, T., Xiao, C.: Fastgcn: fast learning with graph convolutional networks via importance sampling. In: International Conference on Learning Representations (2018)

2. Hamilton, W., Ying, Z., Leskovec, J.: Inductive representation learning on large graphs. In: Advances in Neural Information Processing Systems, pp. 1024–1034 (2017)
3. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: IEEE Conference on Computer Vision and Pattern Recognition, pp. 770–778 (2016)
4. Kingma, D., Ba, J.: Adam: a method for stochastic optimization. In: International Conference on Learning Representations (2014)
5. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907 (2016)
6. Kipf, T.N., Welling, M.: Variational graph auto-encoders. arXiv preprint arXiv:1611.07308 (2016)
7. Li, G., Muller, M., Thabet, A., Ghanem, B.: Deepgcns: can gcns go as deep as cnns? In: IEEE International Conference on Computer Vision, pp. 9267–9276 (2019)
8. Li, Q., Han, Z., Wu, X.M.: Deeper insights into graph convolutional networks for semi-supervised learning. In: AAAI Conference on Artificial Intelligence (2018)
9. Li, R., Wang, S.: Adaptive graph convolutional neural networks. In: AAAI Conference on Artificial Intelligence (2018)
10. Ma, Y., Wang, S., Aggarwal, C.C., Tang, J.: Graph convolutional networks with eigenpooling. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 723–731 (2019)
11. Monfardini, G., Di Massa, V., Scarselli, F., Gori, M.: Graph neural networks for object localization. Frontiers in Artificial Intelligence and Applications. pp. 665–669 (2006)
12. Perozzi, B., Al-Rfou, R., Skiena, S.: Deepwalk: online learning of social representations. In: Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 701–710 (2014)
13. Shang, C., Tang, Y., Huang, J., Bi, J., He, X., Zhou, B.: End-to-end structure-aware convolutional networks for knowledge base completion. In: AAAI Conference on Artificial Intelligence (2019)
14. Velickovic, P., Cucurull, G., Casanova, A., et al.: Graph attention networks. In: International Conference on Learning Representations (2018)
15. Wang, X., He, X., Wang, M., Feng, F., Chua, T.S.: Neural graph collaborative filtering. In: Proceedings of the 42nd international ACM SIGIR conference on Research and development in Information Retrieval, pp. 165–174 (2019)
16. Wu, F., Souza, A., Zhang, T., Fifty, C., Yu, T., Weinberger, K.: Simplifying graph convolutional networks. In: International Conference on Machine Learning, pp. 6861–6871 (2019)
17. Xu, K., Hu, W., Leskovec, J., Jegelka, S.: How powerful are graph neural networks? In: International Conference on Learning Representations (2018)
18. Yao, L., Mao, C., Luo, Y.: Graph convolutional networks for text classification. In: AAAI Conference on Artificial Intelligence, pp. 7370–7377 (2019)
19. Ying, R., He, R., Chen, K., Eksombatchai, P., Hamilton, W.L., Leskovec, J.: Graph convolutional neural networks for web-scale recommender systems. In: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 974–983 (2018)
20. Zeng, H., Zhou, H., Srivastava, A., Kannan, R., Prasanna, V.: Graphsaint: graph sampling based inductive learning method. In: International Conference on Learning Representations (2019)
21. Zhang, X., Liu, H., Li, Q., Wu, X.M.: Attributed graph clustering via adaptive graph convolution. In: AAAI Conference on Artificial Intelligence (2019)