



Weighted Graph Classification by Self-Aligned Graph Convolutional Networks Using Self-Generated Structural Features

Xuefei Zheng^{1,2}, Min Zhang¹, Jiawei Hu², Weifu Chen^{1(✉)}, and Guocan Feng¹

¹ School of Mathematics, Sun Yat-sen University, Guangzhou, China
alexandrap@163.com, Zhangjmin@gdpu.edu.cn,

{chenwf26,mcsfgc}@mail.sysu.edu.cn

² Tencent-CDG-FIT, Shenzhen, China
jessejwhu@tencent.com

Abstract. Directed weighted graphs are important graph data. The weights and directions of the edges carry rich information which can be utilized in many areas. For instance, in a cashflow network, the direction and amount of a transfer can be used to detect social ties or criminal organizations. Hence it is important to study the weighted graph classification problems. In this paper, we present a graph classification algorithm called Self-Aligned graph convolutional network (SA-GCN) for weighted graph classification. SA-GCN first normalizes a given graph so that graphs are trimmed and aligned in correspondence. Following that structural features are extracted from the edge weights and graph structures. And finally the model is trained in an adversarial way to make the model more robust. Experiments on benchmark datasets showed that the proposed model could achieve competitive results and outperformed some popular state-of-the-art graph classification methods.

Keywords: Graph classification · Graph convolutional networks
Graph normalization · Structural features · Adversarial training

1 Introduction

Many data, such as social networks, cash flow networks or structures of proteins, can be naturally represented in the form of graphs. Graphs can be simply classified into directed graphs and undirected graphs, based on whether nodes are connected by directed or undirected edges. For examples, cash flow networks are directed graphs, since money is transferred from one account to another; protein-protein interaction networks are undirected graphs, where undirected edges are used to characterize the interaction between protein molecules. One of the most important types of graph is weighted graph, in which each edge is associated with a numerical weight. Since it is natural to measure the relationship between

nodes based on edge weights, weighted graphs have been widely used to store network objects. Given a collection of weighted graphs, the work considers how to classify the graphs. Unlike grid data, there are several challenges in graph classification:

- Graphs usually differ in size, i.e., the number of nodes and the number of edges of graphs are different;
- the nodes of any two graphs are not necessarily in correspondence, which, however, is critical in comparing graphs;
- although sub-structures are basic elements for classifying graphs, how to use the structural information in graph classification is still an open problem.

In the past decades, Support Vector Machines (SVMs) have been shown the great power in classification [15]. When applying SVMs to graph classification problems, the core issue is how to define kernels for graphs. The most popular way to define graph kernel is based on subgraphs [13, 16]. In summary, those algorithms first recursively decomposed a graph into subgraphs, then counted how many times each given subgraph occurred in the graph, and finally kernel functions were defined to measure the similarity between the vectors of the subgraph-occurring frequencies. The major limitation of those algorithms is the high computational complexity, due to the decomposition of the graph, which restricts the graph kernels only suitable to small number of subgraphs with few nodes.

Recently, convolutional neural networks (CNNs) have achieved great success in many fields [7, 8]. Traditional CNNs are designed for grid data, and there is an implicit order of the components. Hence, a receptive field can be moved in particular directions. However, graph data don't have such specific ordering and it is necessary to redefine convolution operators for graph data. Many researches tried to extend CNNs for graph data. One of the popular approaches is to generalize the convolution in the graph Fourier domain [3, 4]. The basic procedure of those algorithms was first to do the eigenvalue decomposition of the Laplacian matrix of a graph, and the first d eigenvectors were then used as the filters parameterized by diagonal matrices whose diagonal elements were related to the eigenvalues of the graph Laplacian matrix. Although the definition of spectral graph convolution in this way is attractive and the experimental results reported are significantly improved, it was time-consuming (the computational complexity is $\mathcal{O}(n^2)$, where n is the number of nodes of a graph) to calculate the eigenvectors of the Laplacian matrix, in particular for large-scale graphs. Meanwhile, those methods assumed that the input vertices were in correspondence so that the nodes could be easily transformed into a linear layer and fed to a convolutional architecture. For handling general graph data (with different nodes and edges), Niepert et al. [11] proposed an algorithm called PATCHY-SAN to extract locally connected regions from graphs using the Weisfeiler-Lehman algorithm [18] for graph labelling so that the nodes of graphs were ordered and convolution can be implemented as traversing a node sequence and obtain convincing classification accuracy. One of the drawbacks of PATCHY-SAN is that graph labelling methods are usually not injective and PATCHY-SAN used a

software named NAUTY [9] to break ties between same-label nodes, which will weaken the ordering meaning and lead to eliminating important nodes.

In this paper, we present a new kind of graph convolutional networks (GCNs) for weighted graph classification. Since the proposed model could learn the order of the nodes of a graph by itself based on the importance which is measured by PageRank and degree centrality, we call the model Self-Aligned GCN (SA-GCN). According to the order of the nodes, SA-GCN trims graphs into the same size so that a correspondence of the vertices across input graphs could be fixed. Following that graphs could be compared directly. In contrast to existing GCNs, SA-GCN also considers structural features extracted from edge weights and graph global structure, which are usually neglected by other models but contain rich discriminative information for classification. Another problem could be solved by SA-GCN is small-dataset problem. As it is well-known, due to huge of parameters, a neural network tends to be overfitting if there is lack of training samples. SA-GCN tries to solve this problem by adversarial training which in essence is a way to augment training samples by adding particular noise to existing training samples but has been proved efficient for increasing the robustness of a model [6].

2 Self-Aligned Graph Convolutional Networks

In this section, we will introduce the novel Self-Aligned Graph Convolutional Networks (SA-GCNs). First we will introduce the basic notations, and we will introduce the three elements (graph normalization, structural feature generation and adversarial training) in sequence.

Let $G(V, E, A, W)$ denote a graph G with nodes $V = \{v_1, \dots, v_n\}$, edges $E = \{e_1, \dots, e_m\}$, adjacent matrix A and weight matrix W . Here each edge $e = (u, v) \in V \times V$ is a ordered pair for directed graphs and is unordered pair for undirected graph. $A_{ij} = 1$ represents that there is an edge from v_i to v_j and $A_{ij} = 0$ means there is no edge connecting v_i and v_j . If $A_{ij}=1$, v_i and v_j are called adjacent. Let $\mathcal{N}_1(v)$ denote the node set whose elements are adjacent to node v , and $\mathcal{N}_1(v)$ can be treated as the first-order neighborhood of node v . W_{ij} is the sum of all weights associated with $e(v_i, v_j)$. Obviously, if G is an undirected graph, both A and W are symmetric matrices. Define the degree of node v_i as $d_i = \sum_{j=1}^n W_{ij}$, and degree matrix as $D = \text{diag}(d_1, \dots, d_n)$ and then Laplacian matrix can be defined as $L = D - W$. For directed graphs, the indegree of a vertex is the number of head ends adjacent to the vertex, and the outdegree of the vertex is defined as the number of tail ends adjacent to the vertex. We denote the indegree and the outdegree of node v_i as d_i^{in} and d_i^{out} or simply as d_i^{in} and d_i^{out} . Details of graph theory can be referred to Ref. [5].

2.1 Graph Normalization

In order to compare the graphs, we should impose an order on the vertices across input graphs. Intuitively, nodes should be ordered based on their importance in

a graph. In this paper, PageRank and degree centrality are used to measure the importance of nodes, according to which graphs are trimmed into fixed size. Breadth-first search (BFS) is used to find the k -most-important neighbors of each selected vertex to form the neighborhoods as the receptive fields so that convolution operators can be easily implemented.

Node Ranking. Node ranking is the basis of graph normalization, and there are many criteria to measure the importance of a node in a graph. In this work, we chose PageRank (PR) [12] and degree centrality (DC) [10] as our node ranking criteria, as both of these methods are highly effective and efficient in characterizing the importance of a node in a graph, and they can be easily extended to weighted graphs.

- **PageRank.** PageRank is originally designed to use link structure to rank web pages, but can be easily extended to any directed graphs. The algorithm first assigns every node the same initial value as its PR value, then in each iteration splits the PR value of every node equally among the nodes that it points to. After each iteration, the PR value of node v_i is updated by

$$PR_i(t) = \sum_{j=1}^n A_{ji} \frac{PR_j(t-1)}{d_j^{out}} \quad (1)$$

When there is a node with outdegree 0, Eq. (1) won't work. In order to handle this problem, a damping factor p ($0 < p < 1$) is added to denote the probability that a node would split its PR value among all nodes in the graph, and $1 - p$ denotes the probability that a node would split its PR value among the nodes it points to. The update equation is changed to:

$$PR_i(t) = (1 - p) \sum_{j=1}^n A_{ji} \frac{PR_j(t-1)}{d_j^{out}} + \frac{p}{n} \quad (2)$$

- **Degree Centrality.** Degree is a simple but effective measure to reflect node importance. For an undirected graph, the degree of node v_i is the number of nodes that are directly connected with v_i through edges. The degree centrality of a node in an undirected graph is defined as

$$DC(i) = \frac{d_i}{n - 1}, \quad (3)$$

where n is the number of nodes in the graph. Hence, $n - 1$ is the largest possible degree of a node. For a directed graph, since each node has indegree d_i^{in} and outdegree d_i^{out} . The degree centrality can be defined as:

$$DC(i) = \frac{d_i^{out}}{n - 1} \quad (4)$$

It should be noted that degree centrality only reflects the local influence of a node, but doesn't consider the global structure of a graph and the quality of the neighbors of a node. Therefore, degree centrality is seldom used alone as the ranking criterion.

To rank the nodes in a graph, we first compute both the PageRank values and the degree centralities of the nodes in a graph. Then we rank the nodes based on their PageRank values, and break the ties based on the degree centralities, and eventually obtain a node ranking without ties. In the experiments, the first w important nodes were chosen to form a fixed-size ordered node sequence (denoted by P). For weighted graphs, the edges between nodes have not only directions but also weights. In general, some edges are more important than others, which means those edges are more useful. To extend PageRank to weighted graphs, we just need to change the PR value splitting strategy in the iterations. Instead of equally splitting the PR value of a node among the nodes it points to, each neighboring node get a proportion of its PR value which is defined by the edge weight divided by the sum of weights on all edges leaving the current node.

Neighborhood Generation. After we have generated the ordered sequence P with w important nodes, we consider generate the receptive field (i.e., the neighborhood) for each vertex in P . Algorithm 1 depicts the procedure. First, $\forall v_i \in P$, we use node v_i as a starting point and apply breadth-first search(BFS) to find its neighbors iteratively with an increasing distance from v_i and add them to set N_i . It should be noted that inside the while loop, the distance from v_i increases 1 in each iteration and the adjacent sets $\mathcal{N}_1(v)$ of the neighboring nodes are unified into the neighborhood N_i until there are more than k nodes in set N_i or there are no more unexplored nodes to be added into set L_i . Here $|L_i|$ represents the number of unexplored nodes in L_i . Second, the nodes in N_i are sorted by their path distance from v_i in an ascending order, and their PR values and degree centralities are in sequence used to break the ties if they occur. Finally, the top k nodes are selected to form the neighborhood of v_i 's.

```

1  Input:  $PR, DC, P = \{v_1, \dots, v_w\}, k$ 
2  Output: for every node  $v_i$  generate a regularized neighborhood  $N_i^{sorted}$ 
   1: for  $i = 1$  to  $w$  do
   2:    $N_i = [v_i]$ 
   3:    $L_i = [v_i]$ 
   4:   while  $|N_i| < k$  and  $|L_i| > 0$  do
   5:      $L_i = \cup_{v \in L_i} \mathcal{N}_1(v)$ 
   6:      $N_i = N_i \cup L_i$ 
   7:   end while
   8:   if  $|N_i| < k$  then
   9:     Fill  $k - |N_i|$  elements in  $N_i$  with 0
  10:   end if
  11:    $N_i^{sorted}$  = the top  $k$  nodes of  $N_i$  sorted by firstly the distance to node  $v_i$  in an
        ascending order, secondly PR value, and thirdly DC value.
  12:   return  $N_i^{sorted}$ 
  13: end for

```

Algorithm 1: Neighborhood Generation

Figure 1 demonstrates two classes of graphs before and after the proposed graph normalization.

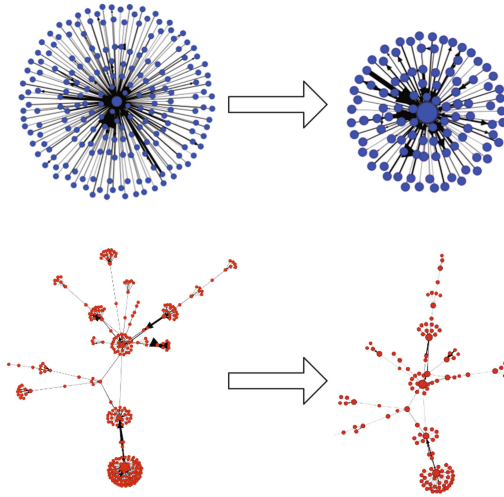


Fig. 1. Two classes of graphs before and after graph normalization. The blue one is a gamble community, and the star network structure was preserved after graph normalization. The red one is a pyramid scheme community, and the hierarchical structure was kept after graph normalization.

2.2 Node Feature Generation

After graph normalization, we consider how to extract structural features for each node from a graph. In this work, two kinds of structural features were used in the experiments:

- **PageRank and Degree Centrality.** The PR values and the degree centralities defined in Sect. 2.1 are useful structural features, as both of them could characterize the relationship between the central vertex and its neighbors.
- **Spectral Embedding.** While PageRank and Degree Centrality could reflect the local structures of a graph, spectral embedding contains the global information for graph partition [14, 17]. In this work, spectral embedding based on Laplacian Eigenmaps [2] was used in the experiments, which is defined to solve the optimization problem

$$\min_{Y^T D Y = I} \frac{1}{2} \sum_{i,j} \|y_i - y_j\|^2 W_{ij} = \text{trace}(Y^T L Y). \quad (5)$$

It can be proved that the optimal solution of (5) is the eigenvectors corresponding to the top K smallest eigenvalues of the random-walk normalized

Laplacian matrix $\hat{L} = I - D^{-1}W$, where K is the dimensions of the embedding space.

After we have computed the spectral embedding, each node is associated with c attributes ($c = K + 2$). Hence, the size of the input tensor to the convolutional network is $w \times k \times c$, where w is the number of selected important points, k is the size of the node neighborhood, and c is the number of channels.

2.3 Adversarial Training

After graph normalization and node feature generation, each graph is represented by a 3-dimension fixed size ordered tensor, that is, graphs are in correspondence. The input tensor for each graph is of dimension $w \times k \times c$, which can be imagined as an image in size $w \times k$ with c channels. We apply a $k \times 1$ convolutional filter with stride 1 on the tensors, and the rest of the architecture can be an arbitrary combination of convolution, pooling and fully connected layer. Softmax layer is used for the final classification. Assume that the input tensor of CNNs is x , the corresponding label is y , and θ is the parameter set, the loss function of SA-GCN is defined as:

$$J_1(\theta, x, y) = -\log P(y|x; \theta) \quad (6)$$

However, as we know, CNNs need to be trained with large datasets to avoid overfitting, but the datasets in the graph classification problems usually quite small. In order to train a robust model, we introduce the adversarial training objective into SA-GCN to make our model more robust and less prone to overfitting. The adversarial training objective is defined as [6]

$$J_2(\theta, x + r, y) = -\log P(y|x + r; \theta), \quad (7)$$

where r is defined as

$$r = -\epsilon \text{sign}(\nabla_x \log P(y|x; \theta)). \quad (8)$$

Thus, the objective function can be defined as the weighted average of J_1 and J_2

$$\tilde{J}(\theta, x, y) = \alpha J(\theta, x, y) + (1 - \alpha) J(\theta, x + r, y), \quad (9)$$

where $\alpha \in [0, 1]$ is a trading coefficient. This cost function can be viewed as actively adding a most destructive perturbation on the input and forcing the model to learn more robust feature to overcome the perturbation. We denote the model with adversarial training SA-GCN⁺

In summary, SA-GCN consists of three steps: (1) graph normalization, (2) node feature generation and (3) CNNs classification with adversarial training. We show the SA-GCN architecture in Fig. 2

3 Experiment

3.1 Data Description

The performance of the proposed model was tested on two types of datasets. The first type of datasets is a real cash flow networks. It contains four classes

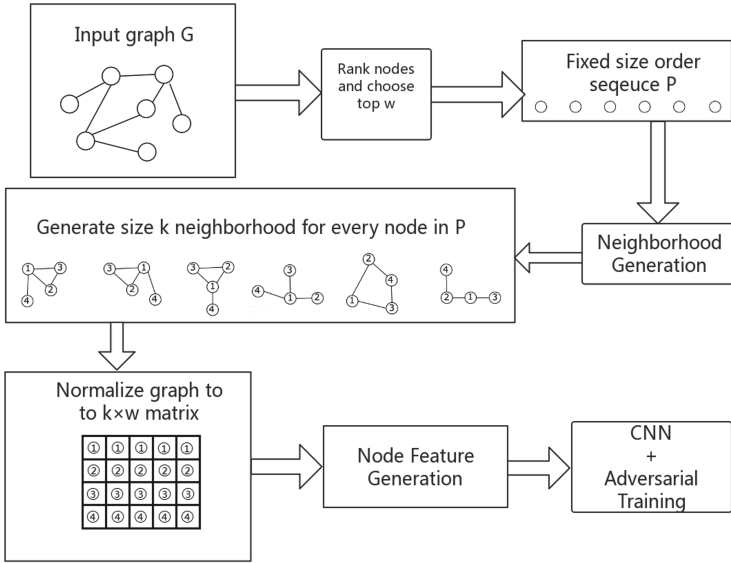


Fig. 2. Architecture of SA-GCN. For input graph G , firstly perform graph regularization with two steps of node ranking and neighborhood generation and rearrange graph to a $w \times k$ matrix, secondly generate structural feature of graphs as node feature, and lastly trains CNNs with adversarial training using node features as input.

of financial communities (pyramid scheme, illegal foreign exchange, gamble and Wechat-sale group) in total of 1944 networks. Each community is represented by a weighted graph without node information. The number of nodes of a graph varies from 100 to 3000.

The second type of datasets consists of 7 standard benchmark datasets described in line 2 to 4 of Table 3. Among them, PTC, NCI1, NCI109 and PROTEINS are bioinformatic datasets, and COLLAB, IMDB_B and IMDB_M are social network datasets. These benchmark datasets have node information, and edge features are discrete or nonexistent.

3.2 Experimental Setting

All experiments were implemented on the Tensorflow platform [1] with a single NVIDIA Titan X GPU. To compare with other models, we performed 10-fold cross validation on all datasets and repeated the experiments 10 times and reported the average prediction accuracies and standard deviations. For SA-GCN, we use $w = 100$, $k = 5$ on all datasets. For PSCN [11], DGCNN [20], and DGK [19] on standard benchmark datasets, we report the best results from the papers. For DGCNN on weighted graph dataset, We set the k of SortPooling such that 60% graphs have nodes more than k , and used weight matrix instead of 0/1 adjacent matrix, and used degree centrality as node information to extend DGCNN to weighted graph classification.

Table 1. Comparison of accuracy results with different node features

Node features	SA-GCN	SA-GCN ⁺
PageRank	84.77 ± 2.05	85.81 ± 2.39
Degree centrality	86.58 ± 2.92	87.50 ± 1.61
Spectral embedding	82.38 ± 3.61	83.49 ± 2.82
All three	89.20 ± 1.46	90.80 ± 1.40

3.3 Node Feature Selection

In this section we compared the performance of three types of node features: PageRank, degree centrality and spectral embedding with the partitions of 3. We used the same graph normalization process to rearrange graph nodes to an ordered tensor, and used PR, DC values and spectral embedding respectively as the only node feature to train CNNs. The first 3 rows of Table 1 list the results and show that degree centrality is the best out of the three. The reason behind it maybe that cash flow networks have fewer hierarchies so that the number of immediate neighbors could reflect more information of a graph. The fourth row in Table 1 shows when using all three types of node features our model reached the accuracy of 89.20%, which is higher than using either single kind of node features. The reason behind it is that these three different node features reflect different structural information of nodes. Therefore, all features could make contribute to the classification and achieved the highest accuracy.

Table 2. Compare with other methods on weighted graph dataset

Method	Accuracy
DGK	62.47 ± 2.56
DGCNN	84.59 ± 0.79
SA-GCN	89.20 ± 1.46
SA-GCN ⁺	90.80 ± 1.40

Table 2 lists the result of our model comparing with DGK [19] and DGCNN [20], and shows that SA-GCN and SA-GCN⁺ performed better than the other two algorithm on weighted graph datasets. DGK is based on deep graph kernel, and couldn't utilize edge features, hence the bad performance. DGCNN is a CNNs based method like ours, but with a different regularization process and feature extracting mechanis. DGCNN use degree centrality as node feature, and achieve higher accuracy than SA-GCN with PR value or spectral embedding as node features, but have lower accuracy than SA-GCN with degree centrality. On the one hand, these results show that degree centrality is indeed a better node feature for these datasets and shows that SA-GCN can extract more useful information than DGCNN from the same node features.

3.4 Adversarial Training

In this section we compared our model with or without adversarial training, denoted by SA-GCN⁺ and SA-GCN respectively. Results are listed in the first and second column of Table 1 and show that adversarial training could enhance the performance of every combination of node features, and when using all three node features and adversarial training, SA-GCN⁺ achieved the best results.

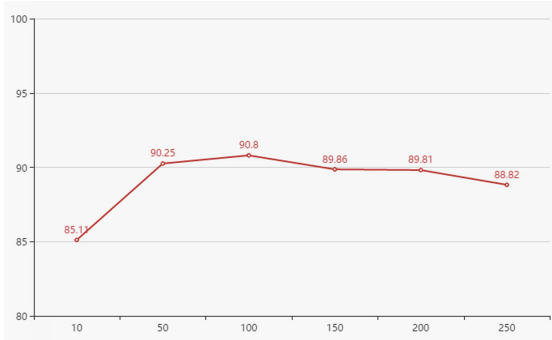


Fig. 3. Set w to 10, 50, 100, 150, 200 and 250 respectively, and report the model’s average accuracy. The horizontal axis is x , and the vertical axis is the corresponding accuracy. Model achieve the highest accuracy when $w = 100$.

3.5 Parameter Analysis

In this section we analyzed the choice of parameters in the graph normalization process. The most important parameter is w , which indicates how many nodes to keep in a graph. For cash flow network dataset, there are four classes of graphs in total of 1944 and each graph have nodes number between 100 and 3000. We chose $w \in \{10, 50, 100, 150, 200, 250\}$, $k = 5$ as parameter and train our model respectively for 10 times and report average accuracy in Fig. 3. The line chart shows that parameter w has a relatively small influence on the accuracy of the model. Accuracy didn’t go up as w increase, which indicates that keeping more nodes of a graph doesn’t provide the model with more useful information for classification; with the increase of w , accuracy slightly come down, this indicates that extra node information can be seen as noise and be deleted from the graph, which further validate the effectiveness of graph regularization process. When $w = 10$, accuracy shows acute drop, which means too little nodes are kept in the regularization process and therefore the performance of classification suffers. Accuracy reach the highest point when $w = 100$, hence is chosen as the optimal w of SA-GCN.

3.6 Compare with Others

We compared SA-GCN⁺ with 2 CNNs based methods (DGCNN [20] and PATCHY-SAN denoted as PSCN [11]) and 1 graph kernel based method (Deep Graphlet Kernel, DGK [19]) on 7 standard benchmark datasets, the results were listed in Table 3. SA-GCN⁺ achieves the highest accuracy on PROTEINS, COLLAB, PTC, NCI109 and IMDB_B, and has highly competitive on other two datasets, which proves that our model works well on unweighted graph despite the fact that it is designed for weighted graph.

Compared to DGCNN and PSCN, the difference lay between the graph regularization mechanism we chose. PSCN uses Weisfeiler-Lehman algorithm and external software to order and chose nodes. But the Weisfeiler-Lehman algorithm sorted nodes by it’s structural role and doesn’t reflect the importance of a node, which means nodes which are more important in a graph could be excluded in the process. DGCNN uses graph convolutional network to extract node features and rank and choose the nodes according to these features, which means the ranking would be refined through the process of training, which is the greatest strength of this method. The drawback is that the edge feature and the structural features of graph (like the partitions) aren’t fully utilized. The strength of SA-GCN is that we fully extract information from edge features and graph structural features and use them to the fullest extend to regularize graph and introduce them as node feature. Furthermore, we use adversarial training to resolve the overfitting problem and make our model more robust.

Table 3. Compare with other method on benchmark datasets. SA-GCN⁺ achieved state of the arts on PROTEINS, COLLAB, PTC, NCI109 and IMDB_B.

Dataset	PTC	NCI1	NCI109	PROTEINS	COLLAB	IMDB_B	IMDB_M
Size	344	4110	4127	1113	5000	1000	1500
Classes	2	2	2	2	3	2	3
Avg. nodes	25.5	29.8	29.6	39.1	74.49	19.77	13
PSCN	62.29 ± 5.68	76.34 ± 1.68	-	75.00 ± 2.51	72.60 ± 2.15	71.00 ± 2.29	45.23 ± 2.84
DGCNN	58.59 ± 2.47	74.44 ± 0.47	-	75.54 ± 0.94	73.76 ± 0.49	70.03 ± 0.86	47.83 ± 0.85
DGK	57.32 ± 1.13	62.48 ± 0.25	62.69 ± 0.23	71.68 ± 0.50	73.09 ± 0.25	66.96 ± 0.56	44.55 ± 0.52
SA-GCN ⁺	62.93 ± 3.26	71.63 ± 0.61	68.86 ± 1.17	76.06 ± 2.04	74.02 ± 1.50	72.40 ± 1.84	45.76 ± 2.03

4 Conclusion

In this paper, we proposed a GCN-based model for weighted graph classification called SA-GCN. Experimental results on several popular datasets showed that the proposed model outperformed some state-of-the-art models, which indicated the advantages of the proposed model.

Directions for future work include using alternative node features and develop a criterion to compare the effectiveness each type of node features; combing the

graph regularization process with the neural network, so that both part can be trained simultaneously.

Acknowledgements. This work is partially supported by the NSFC under grants Nos. 61673018, 61272338, 61703443 and Guangzhou Science and Technology Founding Committee under grant No. 201804010255 and Guangdong Province Key Laboratory of Computer Science.

References

1. Abadi, M., et al.: TensorFlow: a system for large-scale machine learning. In: OSDI 2016, pp. 265–283 (2016)
2. Belkin, M., Niyogi, P.: Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Comput.* **15**(6), 1373–1396 (2003)
3. Bruna, J., Zaremba, W., Szlam, A., Lecun, Y.: Spectral networks and locally connected networks on graphs. In: International Conference on Learning Representations (2014)
4. Defferrard, M., Bresson, X., Vandergheynst, P.: Convolutional neural networks on graphs with fast localized spectral filtering. In: Advances in Neural Information Processing Systems, pp. 3844–3852 (2016)
5. Diestel, R.: *Graph Theory*, 3rd edn. Springer, Heidelberg (2006)
6. Goodfellow, I., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples (2015)
7. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. *Proc. IEEE* **86**(11), 2278–2324 (1998)
8. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. *Nature* **521**, 436–444 (2015)
9. McKay, B., Piperno, A.: Practical graph isomorphism, II. *J. Symb. Comput.* **60**, 94–112 (2014)
10. Newman, M.: *Networks: An Introduction*. Oxford University Press, Oxford (2010)
11. Niepert, M., Ahmed, M., Kutzkov, K.: Learning convolutional neural networks for graphs. In: Proceedings of the 33rd International Conference on Machine Learning, pp. 2014–2023 (2016)
12. Page, L., Brin, S., Motwani, R., Winograd, T.: The PageRank citation ranking: bringing order to the web. Technical report, Stanford InfoLab (1999)
13. Shervashidze, N., Borgwardt, K.M.: Fast subtree kernels on graphs. In: Advances in Neural Information Processing Systems, pp. 1660–1668 (2009)
14. Shi, J., Malik, J.: Normalized cuts and image segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.* **22**(8), 888–905 (2000)
15. Vapnik, V.: *Statistical Learning Theory*. Wiley, New York (1998)
16. Vishwanathan, S., Schraudolph, N., Kondor, R., Borgwardt, K.: Graph kernels. *J. Mach. Learn. Res.* **11**(Apr), 1201–1242 (2010)
17. Von Luxburg, U.: A tutorial on spectral clustering. *Stat. Comput.* **17**(4), 395–416 (2007)
18. Weisfeiler, B., Lehman, A.A.: A reduction of a graph to a canonical form and an algebra arising during this reduction. *Nauchno-Tech. Informatsiya* **2**(9), 12–16 (1968)
19. Yanardag, P., Vishwanathan, S.: Deep graph kernels. In: Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 1365–1374. ACM (2015)
20. Zhang, M., Cui, Z., Neumann, M., Chen, Y.: An end-to-end deep learning architecture for graph classification (2018)