SGCN: A Graph Sparsifier Based on Graph Convolutional Networks

Jiayu Li[1(✉)], Tianyun Zhang[2], Hao Tian[1], Shengmin Jin[1], Makan Fardad[2], and Reza Zafarani[1]

[1] Data Lab, EECS Department, Syracuse University, Syracuse, NY 13244, USA
{jli221,haotian,shengmin,reza}@data.syr.edu
[2] EECS Department, Syracuse University, Syracuse, NY 13244, USA
{tzhan120,makan}@syr.edu

Abstract. Graphs are ubiquitous across the globe and within science and engineering. With graphs growing in size, *node classification* on large graphs can be space and time consuming, even with powerful classifiers such as Graph Convolutional Networks (GCNs). Hence, some questions are raised, particularly, whether one can keep only some of the edges of a graph while maintaining prediction performance for node classification, or train classifiers on specific subgraphs instead of a whole graph with limited performance loss in node classification. To address these questions, we propose *Sparsified Graph Convolutional Network* (SGCN), a neural network graph sparsifier that sparsifies a graph by pruning some edges. We formulate sparsification as an optimization problem, which we solve by an Alternating Direction Method of Multipliers (ADMM)-based solution. We show that sparsified graphs provided by SGCN can be used as inputs to GCN, leading to better or comparable node classification performance with that of original graphs in GCN, DeepWalk, and GraphSAGE.

Keywords: Graph sparsification · Node classification · Graph convolutional network

1 Introduction

Graphs have become universal and are growing in scale in many domains, especially on the Internet and social media. Addressing graph-based problems with various objectives has been the subject of many recent studies. Examples include studies on link prediction [16] and graph clustering [21], or node classification [2], which is the particular focus of this study.

In node classification, one aims to classify nodes in a network by relying on node attributes and the network structure. There are two main categories of node classification methods: (1) methods that directly use node attributes and structural information as features and use [local] classifiers (e.g., decision trees) to classify nodes, and (2) random walk-based methods (often used in semi-supervised learning), which classify nodes by determining the probability p that

H. W. Lauw et al. (Eds.): PAKDD 2020, LNAI 12084, pp. 275–287, 2020.
https://doi.org/10.1007/978-3-030-47426-3_22

a random walk starting from node $v_i \in V$ with label c will end at a node with the same label c. The performance of random walk-based methods implicitly relies on graph structural properties, e.g., degrees, neighborhoods, and reachabilities.

In recent studies, neural network classifiers [27] are widely used for both types of methods due to their performance and flexibility. A well-established example is the Graph Convolutional Network (GCN) [14], a semi-supervised model that uses the *whole* adjacency matrix as a *filter* in each neural network layer.

However, there is a major difficulty faced by methods that directly use the whole graph to extract structural information: the size of the graph. Unlike node attributes, as a graph with n nodes grows, the size of its adjacency matrix increases at an $O(n^2)$ rate, which introduces an unavoidable space and computational cost to classifiers. One engineering solution is to store the adjacency matrix in a sparse matrix (i.e., save non-zeros); however, the process is still extremely slow and requires massive storage when the graph is dense or large.

The Present Work: Sparsified Graph Convolutional Network (SGCN). To address space and computational challenge in node classification, we explore whether one can just rely on a subgraph instead of the whole graph, or some edges (potentially weighted), to extract structural information. We propose *Sparsified Graph Convolutional Network* (SGCN), a neural network graph sparsifier to prune the input graph to GCN without losing much accuracy in node classification. We formulate graph sparsification as an optimization problem, which we efficiently solve via the Alternating Direction Method of Multipliers (ADMM) [3]. We also introduce a new gradient update method for the pruning process of the adjacency matrices, ensuring updates to the matrices are consistent within SGCN layers.

To evaluate SGCN, we compare its performance with other classical graph sparsifiers on multiple real-world graphs. We demonstrate that within a range of pruning ratios, SGCN provides better sparsified graphs compared to other graph sparsifiers. We also show that node classification performance using these sparsified graphs can be better or comparable to when original graphs are used in GCN, DeepWalk [18], and GraphSAGE [11]. In sum, our contributions can be summarized as:

1. We propose Sparsified Graph Convolutional Network (SGCN), the first neural network graph sparsifier aiming to sparsify graphs for node classification;
2. We design a gradient update method that ensures adjacency matrices in the two SGCN layers are updated consistently;
3. We demonstrate the sparsified graphs from SGCN perform better in node classification that those provided by other graph sparsifiers; and
4. We show that sparsified graphs obtained from SGCN with various pruning ratios, if used as inputs to GCN, lead to classification performances similar to that of GCN, DeepWalk and GraphSAGE using the whole graphs.

The paper is organized as follows. We review related work in Sect. 2. We provide the SGCN problem definition in Sect. 3. Section 4 details the problem formulation, solution, and time complexity of SGCN. We conduct experiments in Sect. 5 and conclude in Sect. 6.

2 Related Work

Graph Neural Networks. Inspired by the major success of convolutional neural networks in computer vision research, new convolutional methods have emerged for solving graph-based problems. There are two main types of graph convolutional networks: *spectral-based* methods and *spatial-based* methods.

Spectral-based methods, which include GCNs [6,14], are based on spectral graph theory. Spectral-based convolutional networks often rely on graph signal processing and are mostly based on normalized graph Laplacian. Other examples include the work of Bhagat et al. [17], which aims to represent a graph by extracting its locally connected components. Another is DUIF, proposed by Geng et al. [9], which uses a hierarchical softmax for forward propagation to maximize modularity. One main drawback of spectral-based methods is the need to perform matrix multiplication on the adjacency matrix, which is costly for large graphs.

Spatial-based methods focus on aggregating the neighborhood for each node. These methods can be grouped into (1) recurrent-based and (2) composition-based methods. Recurrent-based methods update latest node representation using that of their neighbors until convergence [5,20]. Composition-based methods update the nodes' representations by stacking multiple graph convolution layers. For example, Gilmer et al. [10] develop a message passing neural network to embed any existing GCN model into a message passing (the influence of neighbors) and readout pattern. Spatial-based methods are often more flexible and easier to apply to large networks.

Graph Sparsification. For graph sparsification, previous studies have distinct objectives from that of ours. Generally speaking, most graph properties of a dense graph can be approximated from its [sparsified] sparse graph. *Cut sparsifiers* [1,8,13] ensure the total weight of cuts in the sparsified graph approximates that of cuts in the original graph within some bounded distance. Spectral sparsifiers [23,24] ensure sparsified graphs preserve spectral properties of the graph Laplacian. There are various applications for graph sparsification. Some examples include, the work of Serrano et al. [22], which aims to identity the *backbone of a network* that preserves structural and hierarchical information in the original graph; the study by Satuluri et al. [19], which applies local sparsification to preprocess a graph for clustering; the study by Lindner et al. [15], which proposes a local degree sparsifier to preserve nodes surrounding local hub nodes by weighing edges linking to higher degree nodes more; and the work by Wilder and Sukthankar [26], which aims to minimize divergence of stationary distribution of a random walk while sparsifying the graph.

These studies are similar, but with different objectives from that of ours. Instead of preserving graph properties, the neural network sparsifier proposed in this work focuses on node classification, so that the space cost is reduced due to sparsification, while node classification performance is maintained.

3 Problem Definition

Consider an undirected graph $G = (V, E)$, its nodes $V = \{v_1, \ldots, v_n\}$, and its edges $E = \{e_1, \ldots, e_m\}$. Let $n = |V|$ denote the number of nodes and $m = |E|$ denote the number of edges. Given adjacency matrix A of G and features for each node $v : X(v) = [x_1, \ldots, x_k]$, the forward model (i.e., output) of a two-layered graph convolutional network (GCN), as formulated by Kipf and Welling [14], is

$$Z(\hat{A}, W) = \text{softmax}(\hat{A}\ \text{ReLU}(\hat{A}XW^{(0)})W^{(1)}), \tag{1}$$

where $\hat{A} = \widetilde{D}^{-\frac{1}{2}}\widetilde{A}\widetilde{D}^{-\frac{1}{2}}$, $\widetilde{D} = \text{diag}(\sum_j \widetilde{A}_{ij})$, $\widetilde{A} = A + I_N$, X is the matrix of node feature vectors $X(v)$, and $W^{(0)}$ and $W^{(1)}$ are the weights in the first and second layer, respectively. Functions $\text{softmax}(x_i) = \exp(x_i)/\sum_i \exp(x_i)$ and $\text{ReLU}(\cdot) = \max(0, \cdot)$ both perform entry-wise operations on their arguments. Graph sparsification aims to reduce the number of edges $|E|$ in the original graph G to $|E_s|$ in a subgraph G_s, i.e., $|E_s| < |E|$, such that subgraph G_s, when used as input to GCN, results in similar classification performance to that of the original graph G. In pruning, adjacency A is pruned to $A_p = A - B \odot A$, where B is a matrix and $\odot$ is Hadamard product. Thus, the new $\widetilde{A}$ is $\widetilde{A} = A_p + I_N$ and $\hat{A}$ is the updated filter for A_p. We will explore how the ratio of graph sparsification in this filter affects SGCN performance.

4 SGCN: Sparsified Graph Convolutional Networks

We first illustrate the problem formulation and solution, followed by SGCN algorithm, a new gradient update method, and the SGCN time complexity analysis.

4.1 Problem Formulation and Solution

Problem Formulation. The output of graph convolutional networks in Eq. (1) is a function of $\hat{A}$ and W, but as $\hat{A}$ can be written as a function of A, the output can be stated as $Z(A, W)$. For semi-supervised multiclass classification, loss function of the neural networks is the cross-entropy error over labeled examples:

$$f(A, W) = -\sum_{l \in \mathcal{Y}_L} \sum_f Y_{lf} \ln(Z_{lf}), \tag{2}$$

where $\mathcal{Y}_L$ is the set of node indices that have labels, Y_{lf} is a matrix of labels, and Z_{lf} is the output of the GCN forward model. Our aim is to achieve a sparse graph, with weight matrices being fixed in SGCN. In the following, we will use $f(A)$ to present the loss function and formulate our problem as:

$$\begin{aligned} &\underset{A}{\text{minimize}} \quad f(A), \\ &\text{subject to} \quad \|A\|_0 \leq \eta. \end{aligned} \tag{3}$$

For Eq. (3), we define an indicator function to replace constraint:

$$g(\Lambda) = \begin{cases} 0 & \text{if } \|\Lambda\|_0 \leq \eta; \\ +\infty & \text{otherwise.} \end{cases}$$

Therefore, Eq. (3) formulation can be rewritten as

$$\underset{A}{\text{minimize}} \quad f(A) + g(A). \tag{4}$$

Solution. In Eq. (4), the first term $f(\cdot)$ is the differentiable loss function of the GCN, while the second term $g(\cdot)$ is the non-differentiable indicator function; hence, problem (4) cannot be solved directly by gradient descent. To deal with this issue, we propose to use Alternating Direction Method of Multipliers (ADMM) to rewrite problem (4). ADMM is a powerful method for solving convex optimization problems [3]. Recent studies [12,25] have demonstrated that ADMM also works well for some nonconvex problems.

The general form of a problem solvable by ADMM is

$$\begin{aligned} \underset{\alpha,\ \beta}{\text{minimize}} \quad & f(\alpha) + g(\beta), \\ \text{subject to} \quad & P\alpha + Q\beta = r. \end{aligned} \tag{5}$$

The problem can be decomposed to two subproblems via augmented Lagrangian. One subproblem contains $f(\alpha)$ and a quadratic term of α; the other contains $g(\beta)$ and a quadratic term of β. Since the quadratic term is convex and differentiable, the two subproblems can often be efficiently solved. Hence, we rewrite problem (4) as

$$\begin{aligned} \underset{A}{\text{minimize}} \quad & f(A) + g(V), \\ \text{subject to} \quad & A = V. \end{aligned} \tag{6}$$

The augmented Lagrangian [3] of problem (6) is given by

$$L_\rho\big(A, V, \Lambda\big) = f(A) + g(V) + \text{Tr}[\Lambda^T(A - V)] + \frac{\rho}{2}\|(A - V)\|_F^2,$$

where Λ is the Lagrangian multiplier (i.e., the dual variable) corresponding to constraint $A = V$, the positive scalar ρ is the penalty parameter, $\text{Tr}(\cdot)$ is the trace, and $\|\cdot\|_F^2$ is the Frobenius norm.

By defining the scaled dual variable $U = (1/\rho)\Lambda$, the augmented Lagrangian can be equivalently expressed in the scaled form:

$$L_\rho\big(A, V, U\big) = f(A) + g(V) + \frac{\rho}{2}\|A - V + U\|_F^2 - \frac{\rho}{2}\|U\|_F^2.$$

When we apply ADMM [3] to this problem, we alternately update the variables according to

$$A^{k+1} := \underset{A}{\arg\min} \quad L_\rho\big(A, V^k, U^k\big), \tag{7}$$

$$V^{k+1} := \underset{V}{\arg\min} \quad L_\rho\big(A^{k+1}, V, U^k\big), \tag{8}$$

$$U^{k+1} := U^k + A^{k+1} - V^{k+1}, \tag{9}$$

until

$$\|A^{k+1} - V^{k+1}\|_F^2 \leq \epsilon, \quad \|V^{k+1} - V^k\|_F^2 \leq \epsilon. \tag{10}$$

In (7), we solve the first subproblem:

$$\underset{A}{\text{minimize}} \quad f'(A) := f(A) + \frac{\rho}{2}\|A - V^k + U^k\|_F^2 \ . \tag{11}$$

In the above problem, as the loss function $f(A)$ and the ℓ_2-norm are differentiable, we can use gradient descent to solve it. As $f(A)$ is nonconvex with respect to the variable A, there has been no theoretical guarantee on the convergence, when solving problem (11). We present a method to solve (11) in Sect. 4.3.

In (8), we solve the second subproblem, which is

$$\underset{V}{\text{minimize}} \quad g(V) + \frac{\rho}{2}\|A^{k+1} - V + U^k\|_F^2 \ . \tag{12}$$

As $g(\cdot)$ is the indicator function, problem (12) can be solved analytically [3], where the solution is

$$V^{k+1} = \mathbf{\Pi_S}(A^{k+1} + U^k), \tag{13}$$

where $\mathbf{\Pi_S}(\cdot)$ is the Euclidean projection onto set $\mathbf{S} = \{\Lambda \mid \|\Lambda\|_0 \leq \eta\}$.

Finally, we update the scaled dual variable U according to (9). This is one ADMM iteration. We update the variables iteratively until condition (10) is satisfied, indicating the convergence of ADMM.

4.2 SGCN Algorithm

In the solution provided in Sect. 4.1, we need to maintain η, the number of non-zero elements. The Euclidean projection in Eq. (13) maintains η elements in $\widetilde{A}^{k+1} + U^k$ with the largest magnitude and sets the rest to zero. This is proved to be the optimal and the analytical solution to subproblem (12) for edge pruning of graphs. In GCN, filters in the loss function in Eq. (2) consist of $\widetilde{D}^{-\frac{1}{2}}\widetilde{A}\widetilde{D}^{-\frac{1}{2}}$, where $\widetilde{A} = A + I_N$ and $\widetilde{D} = \sum_j \widetilde{A}_{ij}$. Variable I_N is the identity matrix and $\widetilde{A}$ is a [modified] adjacency matrix. Variables $\widetilde{A}$ and $\widetilde{D}$ in each layer are fixed and non-trainable in the original GCN. To solve graph sparsification based on GCN and maintain classification performance, variable $\widetilde{A}$ should be trained and updated iteratively. As variable $\widetilde{D}$ depends on $\widetilde{A}$, $\widetilde{A}$ in the original loss function cannot be directly differentiated. Thus, we expand the forward model into:

$$Z(A) = \text{diag}(\sum\nolimits_j (A+I)_{ij})^{-\frac{1}{2}}(A+I)\text{diag}(\sum\nolimits_j (A+I)_{ij})^{-\frac{1}{2}}XW. \tag{14}$$

In Eq. (14), no variable depends on $\widetilde{A}$. However, we cannot still train variables A and W simultaneously as the differentiation of A depends on W and vice versa. Hence, we first train weights variable W in SGCN. By fixing variable W in this model, the adjacency matrix A can be regarded as a trainable variable. With ADAptive Moment estimation (ADAM) optimizer, gradients of the variable (adjacency matrix A) can be updated in SGCN. Algorithm 1 provides the SGCN pseudo-code. We use variable A to initialize variable V in each layer using function `Initialize()` and apply function `Zerolike()` to V to ensure variable U has the same shape as V with all the zero elements.

Algorithm 1: The SGCN Algorithm

input : Adjacency matrix A, feature matrix X, ADMM iterations k, and pruning ratio $p\% = \frac{\eta}{|E|} \times 100\%$
output: Pruned adjacency matrix A_p

Train weight matrix W in SGCN;
for $i \leftarrow 1$ **to** *the number of layers* **do**
    $V_i \leftarrow$ `Initialize(`A_i`)`;
    $U_i \leftarrow$ `Zerolike(`V_i`)`;
for $k \leftarrow 0$ **to** *ADMM iterations* **do**
    Solve subproblem (11) and update A's in two layers;
    for $i \leftarrow 1$ **to** *the number of layers* **do**
        Update V_i's by performing Euclidean projection (13);
        Update U_i's by performing (9);
Fetch A_1 from the first layer;
Set the smallest $p\%$ of non-zero elements in A_1 to zero;
Obtain a pruned adjacency matrix A_p;

4.3 Adjacency Matrix Training

When training adjacency matrix A in Algorithm 1, we should maintain the adjacency matrices in the first and second layer consistent. To address this issue, we propose a method to update the gradients of the adjacency matrix, when fixing weight matrices W in the two layers. A mask m is defined using the adjacency matrix A. As we use gradient descent, the following equation based on Eqs. (2) and (14) can be applied to update the trainable variable (adjacency matrix A) at each step to solve problem in Eq. (11):

$$A_i^{k+1} = A_i^k - \gamma(m \odot \frac{\partial f'(A_i^k)}{\partial A_i^k}), \tag{15}$$

for $i = 1, \ldots, n$, where γ is the learning rate. In the process of updating A, we keep the gradient matrices of the adjacency matrix symmetric in two layers and gradients are set to zero when there are no edges between nodes. Also, diagonal elements are zero in the gradient matrix as we only update the adjacency matrix and consider no self-loops at each node. To maintain the adjacency matrices in the two layers identical, we compute average gradients for the same edge in the two adjacency matrices. We assign these average gradients to the corresponding edges in the matrices for updating elements of the adjacency matrices.

4.4 Time Complexity Analysis

The GCN training time complexity is $\mathcal{O}(L|A_0|F + LNF^2)$, where L is the number of layers, N is the number of nodes, $|A_0|$ is the number of non-zeros in an adjacency matrix, and F is the number of features [4]. Hence, assuming ADMM

takes k iterations, the SGCN time complexity is $\mathcal{O}(kL|A_0|F + kLNF^2)$. Compared to SGCN training time complexity, the time to update variables V and U according to Eqs. (13) and (9) is negligible. In our SGCN, $k = 4$ and $L = 2$. Also, we need only a few iterations to solve subproblem (11), which indicates the training time complexity of SGCN is similar to that of GCN. The time complexity for the forward model in SGCN is $\mathcal{O}(|\Gamma|FC)$, where $|\Gamma|$ is linear in the number of edges and C is the dimension of feature maps. Hence, it is less than that of GCN: $\mathcal{O}(|\epsilon|FC)$, as we have $|\Gamma| \leq |\epsilon|$.

5 Experiments

There are two natural ways to measure the effectiveness of SGCN.

- First, is to compare the node classification performance of sparsified subgraphs obtained using SGCN with that of other sparsifiers. For that, we use GCN for node classification, and compare the node classification performance of SGCN with two well-known sparsifiers: *Random Pruning* (RP) sparsifier and *Spectral Sparsifier* (SS). The RP removes edges uniformly at random from a graph with some probabilities. The SS is the state of the art spectral sparsifier [7], which sparsifies graphs in near linear-time.
- Second, is to compare the node classification performance of the sparsified graphs compared to that of the original graphs. For that, we compare the performance of GCN using sparsified subgraphs provided by SGCN with that of GCN, DeepWalk, and GraphSAGE using original graphs.

5.1 Experimental Setup

Datasets. To evaluate the performance of node classification on sparsified graphs, we conduct our experiments on four attributed graphs. These graphs have been utilized for evaluation in previous studies and are hence used for evaluation. All datasets are available online.[1] Here, we briefly introduce these datasets:

- **CiteSeer:** A citation network of publications classified into six categories. Each publication is attributed by a 0/1-valued word vector indicating the absence/presence of the corresponding word from the dictionary;
- **Cora:** Similar to CiteSeer, a citation network with 7 categories;
- **Terrorists:** This dataset contains information about terrorists and their relationships. Each terrorist is described by a 0/1-valued vector providing features of the individual; and
- **Terrorist Attacks:** The dataset provides information on terrorist attacks classified into 6 different categories, while a 0/1-valued vector provides the absence/presence of a feature.

[1] http://linqs.soe.ucsc.edu/data.

Preprocessing. We preprocess the data for existing node classification models [14,28]. We split the data into 10 folds for cross validation. In each training fold, we only select 20 instances for each label as the labeled instances. Other instances remain unlabeled, from which we randomly select 500 instances for validation set, which is used to train our hyper-parameters. We filter and reorder the adjacency matrices and attribute vectors to ensure they are ordered according to training/testing folds.

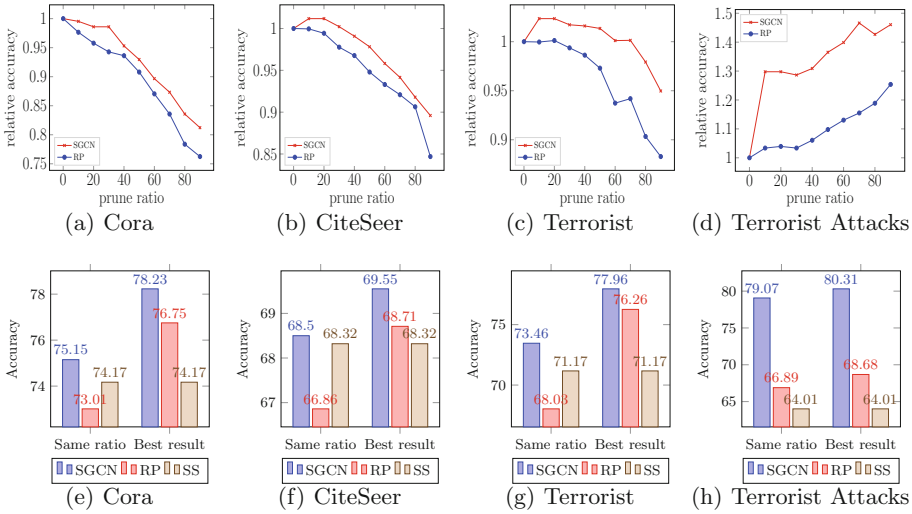

Fig. 1. Performance of GCN using sparsified subgraphs provided by SGCN, RP, and SS sparsifiers. SGCN outperforms all other sparsifiers across datasets. (Color figure online)

Parameter Setup. We vary the pruning ratio (p in Algorithm 1) from 10% to 90% in SGCN and RP. When pruning ratio is 0%, the model is the original GCN. We use the default parameters in GCN, DeepWalk, and GraphSAGE. In RP, we set random seeds from 0 to 9. For SGCN, we set ρ to 0.001 and the training learning rate to 0.001. In SS, we use the default suggested parameters for spectral sparsifier [7]. Due to obtaining 10 folds for each dataset, we run SGCN, RP and SS, in each fold of each dataset to obtain sparsified subgraphs and use these subgraphs as inputs for GCN.

5.2 Results and Performance Analysis

Comparing Sparsifiers. Fig. 1 provides the average performance of GCN with sparsified subgraphs obtained from SGCN and other graph sparsifiers. In Figs. 1(a), (b), (c) and (d), performances are provided in *relative accuracy*, where accuracy is divided by the baseline: accuracy of models from GCN. In Cora

dataset, sparsified subgraph provided by SGCN perform better in GCN than those provided by RP, as shown in Fig. 1(a). For CiteSeer dataset, Fig. 1(b) shows that sparsified subgraph provided by SGCN with pruning ratios between 0% and 30%, when used as input to GCN, can yield accurate classification models. In Terrorists datasets, applying subgraphs from SGCN as inputs to GCN can easily obtain a higher accuracy, as shown in Fig. 1(c). Finally, in Fig. 1(d), we observe that GCN performance increases as pruning ratio increase in the Terrorist Attack dataset. Here also SGCN provides better subgraphs than RP does. Figures 1(e), (f), (g) and (h) illustrate the performance of GCN using subgraphs from SGCN, RP, and SS. We compare their best performance, and the performance under the same pruning ratio, as for Spectral Sparsifier (SS) we cannot set pruning ratio. The results show that subgraphs from SGCN perform the best in node classification, and SGCN is more flexible than SS as SGCN allows different pruning ratios.

Node Classification Performance. When using sparsified subgraphs provided by SGCN as inputs to GCN, we obtain a node-classification model, which we denote as SGCN-GCN. On all datasets, SGCN-GCN either outperforms other methods or yields comparable performance using much smaller graphs. On Cora

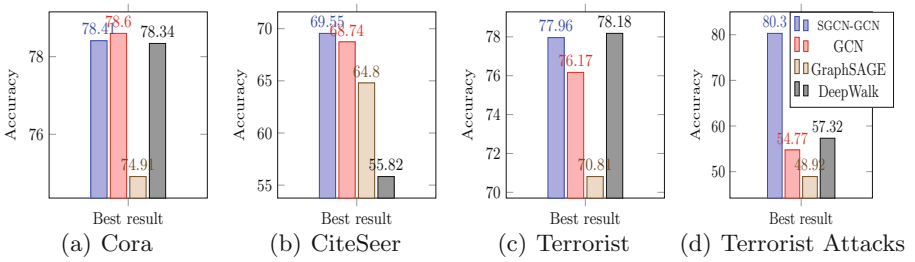

Fig. 2. The performance of SGCN-GCN, GCN, GraphSAGE, and DeepWalk. SGCN-GCN either outperforms or is comparatively accurate, using much smaller graphs. (Color figure online)

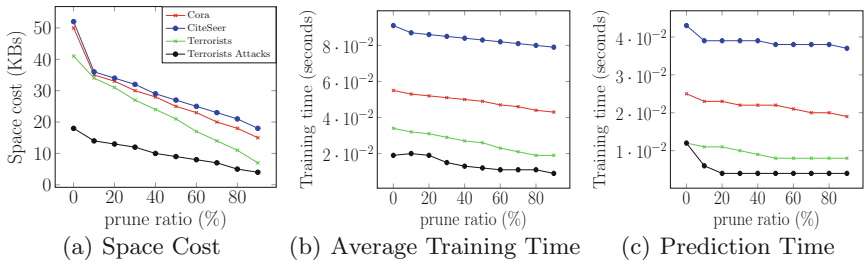

Fig. 3. Space and computational cost using subgraphs from SGCN as inputs to GCN

dataset (Fig. 2(a)), SGCN-GCN outperforms GraphSAGE, and has a comparable performance to DeepWalk and GCN. On other datasets (Figs. 2(b), (c) and (d)), SGCN-GCN outperforms other methods, with the exception of Terrorist dataset (Fig. 2(c)) on which it performs similarly to DeepWalk. Therefore, even though many edges are pruned, subgraphs provided by SGCN when used as inputs to GCN can lead to better or comparable node classification performance over these datasets.

Space and Computational Cost. Here, we feed the subgraphs from SGCN as inputs to GCN and show the actual space and computational cost. The space cost in a graph is $\mathcal{O}(|V|+|E|)$. As SGCN decreases the number of edges $|E|$, the space cost is obviously reduced, as shown in Fig. 3(a). Figure 3(b) and (c) show average training and prediction times in seconds, which have declining trends when the pruning ratio increases. Hence the proposed framework reduces space and computational cost.

6 Conclusion

When a graph is large or dense, node classification often requires massive storage or is computationally expensive. In this paper, we address this issue by proposing the first neural network architecture that can sparsify graphs for node classification. We propose *Sparsified Graph Convolutional Network* (SGCN), a neural network sparsifier. In SGCN, we formulate sparsification as an optimization problem and provide an ADMM-based solution to solve it. Experimental results on real-world datasets demonstrate that the proposed framework can sparsify graphs and its output (sparsified graphs) can be used as inputs to GCN to obtain classification models that are as accurate as using the whole graphs. Hence, SGCN reduces storage and computational cost with a limited loss in classification accuracy.

References

1. Benczúr, A.A., Karger, D.R.: Approximating s-t minimum cuts in $\tilde{O}(n^2)$ time. In: Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, 22–24 May 1996, pp. 47–55 (1996)
2. Bhagat, S., Cormode, G., Muthukrishnan, S.: Node classification in social networks. CoRR abs/1101.3291 (2011)
3. Boyd, S., Parikh, N., Chu, E., Peleato, B., Eckstein, J., et al.: Distributed optimization and statistical learning via the alternating direction method of multipliers. Found. Trends® Mach. Learn. **3**(1), 1–122 (2011)
4. Chiang, W., Liu, X., Si, S., Li, Y., Bengio, S., Hsieh, C.: Cluster-GCN: an efficient algorithm for training deep and large graph convolutional networks. CoRR abs/1905.07953 (2019)
5. Cho, K., et al.: Learning phrase representations using RNN encoder-decoder for statistical machine translation. Comput. Sci. (2014)

6. Defferrard, M., Bresson, X., Vandergheynst, P.: Convolutional neural networks on graphs with fast localized spectral filtering. CoRR abs/1606.09375 (2016)
7. Feng, Z.: Spectral graph sparsification in nearly-linear time leveraging efficient spectral perturbation analysis. In: Proceedings of the 53rd Annual Design Automation Conference, DAC 2016, pp. 57:1–57:6. ACM, New York (2016)
8. Fung, W.S., Hariharan, R., Harvey, N.J., Panigrahi, D.: A general framework for graph sparsification. In: Proceedings of the Forty-third Annual ACM Symposium on Theory of Computing, STOC 2011, pp. 71–80. ACM, New York (2011)
9. Geng, X., Zhang, H., Bian, J., Chua, T.: Learning image and user features for recommendation in social networks. In: IEEE International Conference on Computer Vision (ICCV), pp. 4274–4282, December 2015
10. Gilmer, J., Schoenholz, S.S., Riley, P.F., Vinyals, O., Dahl, G.E.: Neural message passing for quantum chemistry. CoRR abs/1704.01212 (2017)
11. Hamilton, W.L., Ying, R., Leskovec, J.: Inductive representation learning on large graphs. In: NIPS (2017)
12. Hong, M., Luo, Z.Q., Razaviyayn, M.: Convergence analysis of alternating direction method of multipliers for a family of nonconvex problems. SIAM J. Optim. **26**(1), 337–364 (2016)
13. Karger, D.R.: Random sampling in cut, flow, and network design problems. In: Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing, Montréal, Québec, Canada, 23–25 May 1994, pp. 648–657 (1994)
14. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. CoRR abs/1609.02907 (2016)
15. Lindner, G., Staudt, C.L., Hamann, M., Meyerhenke, H., Wagner, D.: Structure-preserving sparsification of social networks. In: Proceedings of the IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2015, ASONAM 2015, pp. 448–454. ACM, New York (2015)
16. Lü, L., Zhou, T.: Link prediction in complex networks: a survey. Physica A **390**(6), 1150–1170 (2011)
17. Niepert, M., Ahmed, M., Kutzkov, K.: Learning convolutional neural networks for graphs. CoRR abs/1605.05273 (2016)
18. 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, KDD 2014, pp. 701–710. ACM, New York (2014)
19. Satuluri, V., Parthasarathy, S., Ruan, Y.: Local graph sparsification for scalable clustering. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2011, pp. 721–732. ACM, New York (2011)
20. Scarselli, F., Gori, M., Tsoi, A.C., Hagenbuchner, M., Monfardini, G.: The graph neural network model. Trans. Neur. Netw. **20**(1), 61–80 (2009)
21. Schaeffer, S.E.: Survey: graph clustering. Comput. Sci. Rev. **1**(1), 27–64 (2007)
22. Serrano, M.Á., Boguñá, M., Vespignani, A.: Extracting the multiscale backbone of complex weighted networks. Proc. Nat. Acad. Sci. U.S.A. **106**(16), 6483–8 (2009)
23. Spielman, D.A., Srivastava, N.: Graph sparsification by effective resistances. CoRR abs/0803.0929 (2008)
24. Spielman, D.A., Teng, S.: Nearly-linear time algorithms for preconditioning and solving symmetric, diagonally dominant linear systems. CoRR abs/cs/0607105 (2006)
25. Takapoui, R., Moehle, N., Boyd, S., Bemporad, A.: A simple effective heuristic for embedded mixed-integer quadratic programming. Int. J. Control, pp. 1–11 (2017)

26. Wilder, B., Wilder, T.: Sparsification of social networks using random walks 2015 (2015)
27. Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., Yu, P.S.: A comprehensive survey on graph neural networks. CoRR abs/1901.00596 (2019)
28. Yang, Z., Cohen, W.W., Salakhutdinov, R.: Revisiting semi-supervised learning with graph embeddings. In: Proceedings of the 33rd International Conference on International Conference on Machine Learning, ICML 2016, vol. 48, pp. 40–48. JMLR.org (2016)