FEW-SHOT CLASSIFICATION ON GRAPHS WITH STRUCTURAL REGULARIZED GCNS

Anonymous authors
Paper under double-blind review

ABSTRACT

We consider the fundamental problem of semi-supervised node classification in attributed graphs with a focus on *few-shot* learning. Here, we propose Structural Regularized Graph Convolutional Networks (SRGCN), novel neural network architectures extending the well-known GCN structures by stacking transposed convolutional layers for reconstruction of input features. We add a reconstruction error term in the loss function as a regularizer. Unlike standard regularization such as L_1 or L_2, which controls the model complexity by including a penalty term depends solely on parameters, our regularization function is parameterized by a trainable neural network whose structure depends on the topology of the underlying graph. The new approach effectively addresses the shortcomings of previous graph convolution-based techniques for learning classifiers in the few-shot regime and significantly improves generalization performance over original GCNs when the number of labeled samples is insufficient. Experimental studies on three challenging benchmarks demonstrate that the proposed approach has matched state-of-the-art results and can improve classification accuracies by a notable margin when there are very few examples from each class.

1 INTRODUCTION

Graph is the most natural structure to model interactions between different entities. For example, graphs are often used to model social networks (Facebook, Twitter), biological networks (protein-protein interaction), and citation networks (arXiv). There has been an increasing research interest in learning tasks on arbitrary structured graphs recently, e.g., (Bruna et al., 2014; Monti et al., 2017; Defferrard et al., 2016; Kipf & Welling, 2017; Hamilton et al., 2017; Velickovic et al., 2018).

In this paper, we consider the fundamental problem of semi-supervised node classification in attributed graphs, where the labels of a small training set are available. Such label information is critical to the performance of classification algorithms, however, it usually requires the inspection of human experts, and thus is often expensive to obtain sufficient label information. It is therefore important to design effective learning methods for the setting where only minimum amount of supervision is available: the classifier needs to generalize well after seeing few or even zero training samples from each class.

To tackle the complexity of huge real-world networks, representation learning-based approaches have become popular for learning tasks on structured data. One line of research is *network embedding* (Tang et al., 2015; Grover & Leskovec, 2016; Bojchevski & Günnemann, 2018). This approach first learns a lower-dimensional representation for each node in the graph in an unsupervised manner, followed by a semi-supervised classifier, e.g., logistic regression or multi-layer perceptron (MLP) layer for node classification.

Another popular line of research is trying to generalize convolutional neural networks (CNNs) to graph data, e.g., (Bruna et al., 2014; Duvenaud et al., 2015). One family of such methods that is most related to the present work is Graph Convolutional Networks (GCNs) Kipf & Welling (2017). GCNs are neural network models operating directly on graphs, which can be viewed as an effective first-order approximation of spectral graph convolutions Hammond et al. (2011).

The above two approaches achieve impressive classification accuracies and have drawn a great amount of attention. However, deep neural networks generally perform poorly on few-shot learn-

ing tasks Ravi & Larochelle (2017). This work proposes a simple yet powerful framework, called Structural Regularized Graph Convolutional Network (SRGCN), for few-shot and zero-shot learning tasks on graphs.

In the conventional GCN model of Kipf & Welling (2017), the convolution filter is restricted to operate in a 1-hop neighborhood. In each layer, the hidden features are first linearly transformed by a trainable weight matrix; the new hidden feature vector of a node is computed by aggregating the hidden features from its neighboring nodes followed by a non-linear activation. Such iterative aggregation schemes have prove to effectively learn node representations and achieve state-of-the-art classification accuracy by utilizing topological and feature information simultaneously. When training examples are abundant, label information can be efficiently propagated to the entire graph in a couple of iterations; indeed, the best performance of GCN is typically achieved with a 2-layer architecture Kipf & Welling (2017). On the other hand, when labels are extremely scarce, intuitively more rounds of aggregations are needed hence deeper networks. However, deeper versions of GCN often lead to worse performance Kipf & Welling (2017); Xu et al. (2018), and according to the analysis of Xu et al. (2018), this is mainly because some critical feature information may be "washed out" via the iterative averaging process.

Present Work. To resolve this dilemma, we propose a novel regularized graph convolutional network architecture, called SRGCN. For coping with small label set, regularization (e.g., L_2 regularization) is usually applied to prevent over-fitting Kipf & Welling (2017); Velickovic et al. (2018), however, the effect of standard regularization methods, which controls the model complexity by including a penalty term depends solely on the parameters, is limited in few-shot learning on graphs. Our regularization on the other hand crucially depends on the structure of the underlying graph, whose form itself is also trainable. We write the loss function as $\mathcal{L} = \mathcal{L}_1 + \lambda_2 \mathcal{L}_2$, where $\mathcal{L}_1$ is the supervised loss and the regularization term $\mathcal{L}_2$ is:

$$\mathcal{L}_2 = \|g_{\theta_2} \circ f_{\theta_1}(X) - X\|^2.$$

Here $\circ$ denotes function composition, X is the input features, $f_{\theta_1}(X)$ is the function defined by a usual GCN, and $g_{\theta_2}()$ is a reconstruction function parameterized by a transposed graph convolutional network. For fixed X and θ_2, $\mathcal{L}_2$ is a function of the GCN parameter θ_1 that we want to learn, hence it is indeed a regularizer for GCN. Note this encoder-decoder structure is reminiscent of Autoencoder (Hinton & Salakhutdinov, 2006).

In other words, our neural network structure is the conventional GCN concatenated with a Transposed Graph Convolutional Network (or Deconvolutional Network (Zeiler & Fergus, 2014)), which is used to reconstruct the original node features. The loss function of the proposed framework consists of two parts: (1) the supervised loss w.r.t. the labeled part of the graph and (2) the unsupervised reconstruction errors of features, i.e., the regularizer. The SRGCN architecture has several desirable properties for few-shot learning tasks including:

1. The regularization term $\mathcal{L}_2$ is parameterized by a convolutional network, whose structure depends on the graph topology (encoded in g_{θ_2}) with a trainable θ_2, and thus has much more expressive powerful than standard regularizations.
2. The reconstruction term in the loss function effectively prevents critical feature information from being washed out during the layer-wise averaging process and could potentially be a building block for training deeper GCNs.
3. In the original GCN, the training of weight matrices is mainly guided by the supervised loss, which is less efficient for the purpose of dimensionality reduction and denoising especially when the training set is small, while the Autoencoder in SRGCN allows the neural network to learn the real signals more efficiently.
4. It is twice deeper than GCN, and thus good for label information propagation, which is particular helpful in the few-shot setting.

We assess our new framework on three widely used benchmarks: *Cora, Citeseer* and *Pubmed* citation networks. The experimental results show that the proposed approach has matched state-of-the-art results and can significantly improve generalization when there are very few examples from each class.

2 SRGCN ARCHITECTURE

2.1 PROBLEM DEFINITION

A graph with features is denoted as $G = (V, E, X)$,where $V = \{v_1, v_2, v_3, ..., v_n\}$ is the vertex set, $E = \{e_{i,j}\}_{i,j=1}^{n}$ is the edge set and $X = \{x_1, x_2, ..., x_n\}$ is the set of feature vectors of all nodes. Each node has a label indication the class it belongs to and we assume there there are totally c classes labels. Let $Y = \{y_1, y_2, ..., y_L\}$ be the labels of training samples. The task is to predict the labels for these unknown nodes, especially when the number of labeled nodes is little and some class labels don't appear in Y.

2.2 FRAMEWORK

In this paper, we propose a semi-supervised model named SRGCN to deal with this problem, whose framework is shown in Figure X. The architecture conceptually consists of two neural networks concatenated together. The first one is a GCN based encoder, while the second one is a decoder, which is essentially a graph transposed convolutional network.

2.2.1 ENCODER

We first describe the encoder part of our SRGCN. In general, it is composed of m_e graph convolution layers. The input of each layer is a set of node features, usually represented as a matrix $H^l = \{h_1, h_2, ..., h_n\} \in \mathbb{R}^{n \times d_l}$, in which n is the number of nodes and d_l is the dimensionality of this hidden layer. The output of each layer is a new set of node features, denoted as $H^{l+1} = \{h'_1, h'_2, ..., h'_n\} \in \mathbb{R}^{n \times d_{l+1}}$, where typically $d_{l+1} < d_l$. More precisely, the layer-wise propagation rule applied on the lth layer can be formalized as follows:

$$H^{l+1} = \sigma(\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} H^l W^l).$$

Here, $\tilde{A} = A + I_N$ is the adjacency matrix of the graph G after adding self-loops to each node via adding the identity matrix I_N; $\tilde{D}$ is a diagonal matrix with $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$; $W^l \in \mathbb{R}^{d \times k}$ is the trainable weight matrix of layer l; and $\sigma()$ is an activation function (usually ReLU or softmax). In the first layer we have $H^0 = X$, i.e. using original features as input.

The output of the last layer, denoted as $Z \in \mathbb{R}^{d \times c}$, is the matrix containing the probabilities that every node v belongs to particular class. Here, we used θ do denote the set of all trainable parameters and optimize the supervised loss function defined as follows:

$$\mathcal{L}_1(\theta; A, X) = -\sum_{y \in Y} y \ln Z_y.$$

2.2.2 DECODER

After the dimensionality of the feature space is reduced with a graph convolutional network, we apply transposed graph convolutional operations $Y = \hat{A}^T X W^T = \hat{A} X W^T$ to reconstruct the original features of all nodes. On the other hand, since $\hat{A}X$ is a special smoothing operation and excessive smoothing will result in the loss of critical feature information, we add a pooling operation (usually max or mean) between each layer to suppress smoothing.

Therefore, the decoder part of the SRGCN consists of the m_d layers of transposed graph convolutions. Each layer takes in as input a set of hidden features, $H^l = \{h_1, h_2, ..., h_n\} \in \mathbb{R}^{n \times d_l}$ and produces a new set of node features $H^{l+1} = \{h'_1, h'_2, ..., h'_n\} \in \mathbb{R}^{n \times d_{l+1}}$, where typically $d_{l+1} > d_l$. More specifically, the lay-wise propagation rule can be written as follows:

$$H^{l+1} = \sigma(pooling(\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} H^l, H^l) W^l)$$

The transposed graph convolutional network use $H^0 = Z$ as the input, whose last layer outputs $\hat{X} = H^{m_d}$, a matrix of the same size as X. The reconstruction loss function is the squared Euclidean distance (or squared Frobenius norm of matrices):

$$\mathcal{L}_2(\theta; A, X) = \parallel \hat{X} - X \parallel^2$$

The final loss function consists of a supervised part $\mathcal{L}_1$, an unsupervised part $\mathcal{L}_2$ (reconstruction error), and a regularization part (we use L_2 regularizer in the experiments):

$$\mathcal{L} = \mathcal{L}_1 + \lambda_1 \mathcal{L}_2 + \lambda_2 \mathcal{L}_{reg}$$

2.3 Implementation Details

For all experiments in this paper, SRGCN uses the following four-layer network architecture:

$$\begin{cases} H_1 = f(\hat{A}XW_1), W_1 \in R^{n \times d_1} \\ Z = softmax(f(\hat{A}H_1W_2)), W_2 \in R^{n \times d_2} \\ H_2 = f(pooling(Z, \hat{A}Z)W_3), W_3 \in R^{n \times d_3} \\ \hat{X} = f(pooling(H_2, \hat{A}H_2)W_4), W_4 \in R^{n \times d_4} \end{cases}$$

Here, $\hat{A} = \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}}$, f is RELU, and the pooling method is max pooling.

3 Discussions

In this section we discuss the benefits SRGCN against GCN by arguing about the influence distributions following the analysis of (Xu et al., 2018).

Xu et al. (2018) define the influence of x on y by measuring how much a change in the input feature of x affects the representation of y in the output layer. For any node y, the influence distribution captures the relative influences of all other nodes. Here, instead of L_1 norm as in (Xu et al., 2018), we use F-norm to define influence score.

Definition 3.1 (F-influence score and distribution) For a simple graph $G = (V, E)$, let h_x^k represent feature of node x $\in$V at the k-th layer of the model. We define the frobenius influence distribution as follows:

$$I_x(y) = || \frac{\partial h_y^{(k)}}{\partial h_x^{(0)}} ||_F \, / \sum_{z \in V} || \frac{\partial h_y^{(k)}}{\partial h_z^{(0)}} ||_F,$$

where $|| \cdot ||_F$ denotes the Frobenius norm of a matrix.

Definition 3.2 (Global aggregate score) We propose a new definition to measure how much can a node get influenced by all other nodes:

$$I(y) = \sum_{x \in v} I_{\{I_x(y) \neq 0\}} / \mid V \mid,$$

where $I_{\{I_x(y) \neq 0\}}$ denotes the indicator function, which equals to 1 when x satisfies $I_x(y) \neq 0$ and otherwise 0. V is the vertices set of graph G.

Theorem 1 *Let $N_k(x)$ denote the set of vertices which satisfies that there exists a length-k path connect it to x and $Labeled$ be the set of vertices which is labeled in the learning task. For every node y in a graph with a K-layer GCN architecture, we have*

$$I(y) \leq (\mid N_k(y) \mid + \sum_{x \in Labeled} \mid N_k(x) \mid) / \mid V \mid$$

If the maximal degree of the nodes in graph is m, we have: $I(y) \leq (1 + \mid Labeled \mid) m^k / \mid V \mid$.

We prove Theorem 1 in the appendix. Here the global aggregate score measures the degree of a node's representation is influenced by other nodes. In some tasks, GCN enjoys a good performance mainly because it utilizes the information that covers almost all nodes. But in a few-shot task, $I(y)$ is a small, meaning that the amount of information used is too small. In order to improve the impact distribution, we need to add extra layers but this will lead to too much averaging, which renders traditional GCN's limits in solving few-shot problems.

Let W be the optimal parameter of the optimization process $W = \arg\min_\theta L(\theta, A, labeled, X)$. From the proof of Theorem 1 in appendix, W is only affected by labeled nodes and their neighbors

in the GCN model. But regularization term in the loss function of SRGCN involves all the nodes in the graph, which makes W be affected by more nodes than before. It implies that the $I(y)$ of most nodes has a larger upper bound than before. If $I(y)$ is small, the model will rely too much on the information of few nodes in the case of few-shot.

4 RELATED WORK

Our approach extends graph convolutional networks of Kipf & Welling (2017), and thus can be categorized as a spectral approach (Bruna et al., 2014; Henaff et al., 2015; Defferrard et al., 2016). Such spectral methods generalize convolutions to the graph domain, working with a spectral operator depending on the graph structure, which learns hidden layer representations that encode both graph structure and node features simultaneously. Our network structure follows the work of Kipf & Welling (2017), which simplifies previous spectral techniques by restricting the propagation to a 1-hop neighborhood in each layer. Chen et al. (2018) propose fast GCNs, which improves the training speed of the original GCN. GAT of Velickovic et al. (2018) allows for assigning different importances to nodes of a same neighborhood via attention mechanisms. Xu et al. (2018) introduce JK networks, which adaptively adjust (i.e., learn) the influence radii for each node and task. Zügner et al. (2018) study adversarial attacks on neural networks for graphs. In contrast to this paper, such work does not explicitly consider the few-shot learning problem and their performance often degenerates rapidly as the number of labeled samples decreases.

Another direction that generalizes convolutions to the graph structured data, namely non-spectral approaches, define convolutions directly on the graph (Duvenaud et al., 2015; Atwood & Towsley, 2016; Monti et al., 2017). Such methods are easier to be adapted to do inductive learning Hamilton et al. (2017); Velickovic et al. (2018); Bojchevski & Günnemann (2018). However, few-shot learning remains a challenge for this class of methods.

On the other hand, node classification is also one of the main applications of network embedding methods, which learns a lower-dimensional representation for each node in an unsupervised manner, followed by a supervised classifier layer for node classification (Perozzi et al., 2014; Tang et al., 2015; Grover & Leskovec, 2016; Wang et al., 2016; Bojchevski & Günnemann, 2018). DeepWalk of Perozzi et al. (2014) uses local information obtained from truncated random walks to learn latent representations. It assumes that a pair of nodes are similar if they are close in the random walks. LINE preserves the first-order and second order proximity between nodes respectively, and concatenates the representations for the first-order and second order proximity (Tang et al., 2015). A recent work of Bojchevski & Günnemann (2018) proposes Graph2Gauss. This method embeds each node as a Gaussian distribution according to a novel ranking similarity based on the shortest path between nodes. A distribution embedding naturally captures the uncertainty about the representation. The work of (Zhou et al., 2018) tackles the challenge of rare category characterization, which is a similar but different problem considered in this paper. Embedding approaches achieve comparable performance in node classification tasks, while the learned representations also prove to be extremely helpful for other downstream applications.

5 EXPERIMENT

In this section, we evaluate the efficacy of our method through experiments compared against the some strong baseline methods. This section summarizes our experimental setup and results.

5.1 EXPERIMENT SETTING

We first introduce the experiment setting up. We evaluated the performance of SRGCN in two classification tasks. (1) In the first task, the set of labeled nodes (i.e., training set) is randomly sampled from the graph, therefore, the number of nodes in each class is uncertain. Moreover, when the size of the training set is small relative to the number of node categories, it is likely that some type of labels may not appear in the training set, i.e., zero-shot learning. (2) In the second task, the number of nodes of each class is fixed in the training set, and each train example is randomly sampled from a specific class. In each experiment, the vaildation set is chosen in the same way as the training set and the rest of nodes will be used as the testing set.

Table 1: Summary of the datasets used in our experiments.

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

Baseline Methods: We compare against the six representative baseline methods: DeepWalk (Perozzi et al., 2014), LINE (Tang et al., 2015), Graph2Gauss (Bojchevski & Günnemann, 2018), GCN (Kipf & Welling, 2017) and GAT (Velickovic et al., 2018). DeepWalk, LINE and Graph2Gauss are unsupervised embedding methods. We use their learned embeddings along with their labels as training data for a logistic regression, then evaluate the performance on the rest of the nodes. GCN and GAT are semi-supervised learning methods, we use them directly to predict the category of each node.

Datasets. We consider three most commonly used citation network datasets: cora, citeseer and pubmed. The datasets contain sparse bag-of-words feature vectors for each document and a list of citation links between documents. For instance, cora is a set research papers of machine learning literature, which consists of 2708 scientific publications categorized into seven classes. Statistics for all datasets are listed in Table 1.

Parameter Settings. For the first task, we evaluate the performance of algorithms with five different sizes of the training set, $10, 20, 30, 40, 50$. For each sample size, we repeat the experiments 50 times and the training samples are selected randomly from the graph each time. In this task, zero-shot instances are likely to occur, we only use GCN and GAT as baseline methods, since we use logistic regression for classification in embedding-based method as in previous work.

To better assess the potential of SRGCN , the hyper-parameter settings of GCN, GAT and SRGCN are basically the same following (Kipf & Welling, 2017; Velickovic et al., 2018). We use the Adam optimizer with learning rate 0.005 and fix the dropout rate to be 0.5; the dimensionality of the hidden layer is set to 16. We apply L_2 regularization with $\lambda_2 = 0.0005$, and set the λ_1 of SRGCN to be 0.0001. The epoch size is 300 and we choose the model that achieves best performance with respect to the validation set.

For the second task, the number of labeled examples per class in the training set is $1, 3, 5, 7, 10$. We also repeat 50 times per scale and select labeled nodes randomly from the graph each time. In this task, we compare our method against both unsupervised embedding methods and semi-supervised learning approaches on Cora, Citeseer datasets. The hyper-parameter setting of semi-supervised learning methods are the same as in task (1). The embedding size of each unsupervised learning method is 128 and other parameters are set in common with Bojchevski & Günnemann (2018).

5.2 Experiment Results

The results of our comparative evaluation experiments are summarized in Tables 2 and 3.

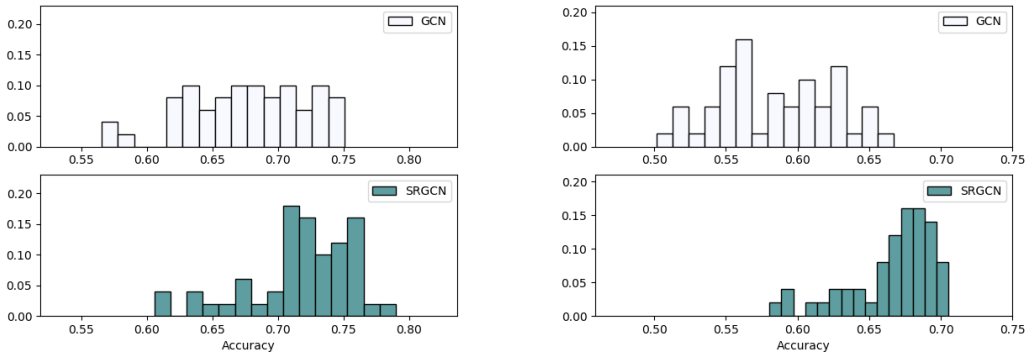
For task (1), we report the mean classification accuracy on the test nodes of all the methods. The empirical results clearly demonstrate state-of-the-art performance being achieved by SRGCN across all three datasets in the few-shot regime. More specifically, we are able to improve upon GCNs by a margin of 6% and 10% on Cora and Citeseer, respectively. From our results about GAT, we see that attention mechanism does not help when the training set is very small.

In the few-shot setting, we observe the classification accuracy is sensitive to the selection of labeled examples. In our experiments, the results w.r.t. 50 randomly picked training sets are recorded. We present the histograms and box-plots of the accuracy distributions of GCN and SRGCN on the 50 training sets in Figure 1 and Figure 2. From histograms and box-plot, we clearly see that SRGCN has a better overall performance and it is also more robust against the choice of the training set.

For task (2), compared to all unsupervised and semi-supervised methods, SRGCN achieves state-of-the-art performance under almost all experimental settings. We observe that the recent G2G method has impressive performance especially in the one-shot experiments. Unlike in the task (1), when

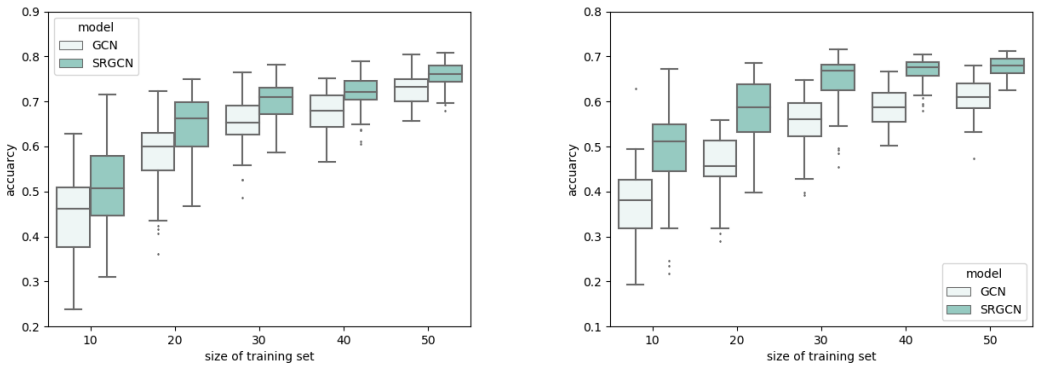
Table 2: Experimental results on task (1). The numbers in the first row indicates sizes of the training set, the node in which is randomly sampled from the entire graph.

	Cora				
	10	20	30	40	50
GCN	44.69	57.84	65.42	67.74	72.64
GAT	35.78	50.45	59.58	62.35	67.72
SRGCN	**50.04**	**64.18**	**70.13**	**71.63**	**76.00**
	Citeseer				
	10	20	30	40	50
GCN	37.14	46.19	55.28	58.40	60.77
GAT	30.69	36.04	50.88	51.14	54.10
SRGCN	**48.84**	**57.99**	**64.04**	**66.60**	**67.72**
	Pubmed				
	10	20	30	40	50
GCN	56.88	66.13	71.67	74.22	**76.64**
GAT	56.15	65.82	71.31	74.07	76.39
SRGCN	**56.98**	**66.39**	**72.42**	**74.33**	76.54



(a) Cora (b) Citeseer

Figure 1: Accuracy distributions of GCN and SRGCN tested on 50 randomly selected training sets, each of size 40. The upper and lower rows corresponds to GCN and SRGCN respectively.



(a) Cora (b) Citeseer

Figure 2: Box-plot of the accuracies of GCN and SRGCN tested on 50 randomly selected training sets, each of size 40.

training examples of each class are evenly distributed, GAT attains much better performance than before. On the other hand, as the number of labeled examples grows, the accuracy of GCN and GAT become closer to SRGCN. This is reasonable, since the effect of structural regularizations will be dominated by the supervised loss when the training set is large.

Table 3: Experimental results on task (2). The numbers in the first row indicates the number of training samples in each class; the node in each class is randomly sampled.

	Cora					Citeseer				
	1	3	5	7	10	1	3	5	7	10
DeepWalk	40.42	53.87	59.40	62.25	65.44	28.38	34.72	38.18	39.76	42.01
LINE	49.48	62.68	66.74	69.00	71.13	28.08	34.70	38.03	40.63	43.19
G2G	**54.56**	68.12	70.93	72.24	73.89	**45.10**	56.46	60.38	62.91	63.15
GCN	43.29	63.32	69.78	72.30	74.64	34.94	51.27	57.63	60.89	62.84
GAT	41.83	61.70	71.17	73.46	76.02	32.88	48.68	54.92	57.91	60.83
SRGCN	52.14	**68.22**	**72.84**	**74.79**	**76.60**	45.05	**59.32**	**64.35**	**66.39**	**67.03**

5.3 Network Visualization

Semi-supervised methods also learn a low-dimensional embedding for each node in the graph, which can be used to create meaningful visualizations of a network in 2D/3D. Moreover, the effectiveness of such methods can also be investigated qualitatively when the learned representations are further projected onto the 2D space. We provide such a visualization of the feature representation produced by GCN and SRGCN respectively on the Cora dataset (Figure 3) with t-SNE (Maaten & Hinton, 2008). The visualization results show that the clustering effect exhibited by SRGCN is more discernible than that by GCN, which is in align with our numerical results.

(a) GCN (b) SRGCN

Figure 3: Visualization: t-SNE plots of the computed feature representations of GCN's last layer and SRGCN's middle layer (used for classification) on the Citeseer dataset. The representations are trained using a labeled set of size 40.

6 Conclusion

We have introduced a novel regularized GCN architecture for few-shot classification problems on graph-structured data. Our SRGCN model combines the original GCN structure with an encoder-decoder architecture, which can be considered as a regularization method. What make this more powerful than standard methods is that the regularization term is a function parameterized by a trainable neural network, whose structure depends on the underlying graph topology. Experiments on a number of network datasets shows that SRGCN has matched state-of-the-art results and can significantly improve classification accuracy when there are very few examples from each class. This suggest that the proposed SRGCN model is capable of overcoming the classification accuracy degeneration caused by insufficient labeled information, while being computationally efficient on par with GCNs.

REFERENCES

James Atwood and Don Towsley. Diffusion-convolutional neural networks. In *Advances in Neural Information Processing Systems*, pp. 1993–2001, 2016.

Aleksandar Bojchevski and Stephan Günnemann. Deep gaussian embedding of graphs: Unsupervised inductive learning via ranking. *International Conference on Learning Representations (ICLR)*, 2018.

Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. *International Conference on Learning Representations (ICLR)*, 2014.

Jie Chen, Tengfei Ma, and Cao Xiao. Fastgcn: fast learning with graph convolutional networks via importance sampling. *International Conference on Learning Representations (ICLR)*, 2018.

Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*, pp. 3844–3852, 2016.

David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in neural information processing systems*, pp. 2224–2232, 2015.

Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 855–864. ACM, 2016.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pp. 1024–1034, 2017.

David K Hammond, Pierre Vandergheynst, and Rémi Gribonval. Wavelets on graphs via spectral graph theory. *Applied and Computational Harmonic Analysis*, 30(2):129–150, 2011.

Mikael Henaff, Joan Bruna, and Yann LeCun. Deep convolutional networks on graph-structured data. *arXiv preprint arXiv:1506.05163*, 2015.

Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *International Conference on Learning Representations (ICLR)*, 2017.

Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodola, Jan Svoboda, and Michael M Bronstein. Geometric deep learning on graphs and manifolds using mixture model cnns. In *Proc. CVPR*, volume 1, pp. 3, 2017.

Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 701–710. ACM, 2014.

Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. *International Conference on Learning Representations (ICLR)*, 2017.

Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: Large-scale information network embedding. In *Proceedings of the 24th International Conference on World Wide Web*, pp. 1067–1077. International World Wide Web Conferences Steering Committee, 2015.

Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *International Conference on Learning Representations (ICLR)*, 2018.

Daixin Wang, Peng Cui, and Wenwu Zhu. Structural deep network embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 1225–1234. ACM, 2016.

Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. *International Conference on Machine Learning (ICML)*, 2018.

Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pp. 818–833. Springer, 2014.

Dawei Zhou, Jingrui He, Hongxia Yang, and Wei Fan. Sparc: Self-paced network representation for few-shot rare category characterization. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 2807–2816. ACM, 2018.

Daniel Zügner, Amir Akbarnejad, and Stephan Günnemann. Adversarial attacks on neural networks for graph data. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 2847–2856. ACM, 2018.

A Proof of Theorem 1

Assumption: Ignore the activation function, but this doesn't really matters(for sigmoid and tanh function it wont let the partial derivative be zero for any variables, but when it is relu the thing changes, so we just consider the situation of tanh and sigmoid etc.), just for the simplicity of our proof. In this case we have :

$$\hat{H} = H.$$

We treat W as the result parameter of the optimization process:

$$W = \arg\min_{\theta} L(\theta, D, labels, A),$$

where D is the input matrix, $labels$ denotes the set of labeled nodes in the semi-supervised task, and A just simply denotes the laplace matrix. In our GCN model, we use the simplest form of the architecture:

$$\hat{H}^{(l+1)} = AH^{(l)}W^{(l)}.$$

To complete part 1 of our proof, we just need to prove that for every node y and every labeled node z, if a node x satisfies $x \notin N_k(z)$ and $x \notin N_k(y)$ that we can get :

$$I_y(x) = 0.$$

First of all, because for every node y and every labelled node z, node x satisfies $x \notin N_k(z)$ and $x \notin N_k(y)$, we can know that there isn't any path shorter than k from x to y, nor from x to z. Therefore,

$$H_y^{(k)} = \sum_{i_{k-1}\in V} \sum_{i_{k-2}\in V} ... \sum_{i_0 \in V} A_{y,i_{k-1}} A_{i_{k-1},i_{k-2}} ... A_{i_1,i_0} H_{i_0}^{(0)} W^{(0)} W^{(1)} ... W^{(k-1)}.$$

Since $A_{x,y} \neq 0 \Leftrightarrow x \in N(y)$, we have

$$i_0 \notin N_k(i_k) \Longrightarrow \forall i_2, i_3, ..., i_{k-1} \in V, \prod_{t=1}^{k} A_{i_t, i_{t-1}} = 0. \tag{1}$$

Therefore,

$$H_z^{(k)} = H_z^{(k)}(\{H_t^{(0)}, t \in N_k(z)\} \cup \{W^{(i)} : i = 0, 1, 2, ..., k-1\}).$$

This represents that $H_z^{(k)}$ is a function of the elements in set $\{H_t^{(0)}, t \in N_k(z)\} \cup \{W^{(i)} : i = 0, 1, 2, ..., k-1\}$. Because

$$\begin{aligned} W &= \arg\min_{\theta} L \\ &= argmin \sum_{z \in Labeled} || H_z^{(k)} - label_z ||_2 \\ &= W(\{H_z^{(k)} : z \in Labeled\}). \end{aligned}$$

This represents that W is a function of the elements in set $\{H_z^{(k)} : z \in Labelled\}$. Because

$$W = W(\{H_z^{(k)} : z \in Labelled\}),$$

we have $W = W(\{H_t^{(0)} : t \in N_k(z), z \in Labelled\})$

Because, for every labeled node z, node x satisfies $x \notin N_k(z)$

$$\Rightarrow \frac{\partial W^{(i)}}{\partial H_x^{(0)}} = 0 \quad \text{for } i = 0, 1, 2, \cdots, k-1.$$

We have:

$$\begin{aligned} \frac{\partial H_y^{(k)}}{\partial H_x^{(0)}} &= \frac{\partial H_y^{(k)}}{\partial H^{(k-1)}} \frac{\partial H^{(k-1)}}{\partial H_x^{(0)}} + \frac{\partial H_y^{(k)}}{\partial W^{(k-1)}} \frac{\partial W^{(k-1)}}{\partial H_x^{(0)}} \\ &= \frac{\partial H_y^{(k)}}{\partial H^{(k-1)}} \frac{\partial H^{(k-1)}}{\partial H_x^{(0)}} + 0 \\ &= \sum_{i_{k-1} \in V} \frac{\partial H_y^{(k)}}{\partial H_{i_{k-1}}^{(k-1)}} \frac{\partial H_{i_{k-1}}^{(k-1)}}{\partial H_x^{(0)}}. \end{aligned}$$

With this recursion formula:

$$\begin{aligned} \frac{\partial H_y^{(k)}}{\partial H_x^{(0)}} &= \sum_{i_{k-1} \in V} \frac{\partial H_y^{(k)}}{\partial H_{i_{k-1}}^{(k-1)}} \frac{\partial H_{i_{k-1}}^{(k-1)}}{\partial H_x^{(0)}} \\ &= \sum_{i_{k-1} \in V} \sum_{i_{k-2} \in V} ... \sum_{i_1 \in V} \frac{\partial H_y^{(k)}}{\partial H_{i_{k-1}}^{(k-1)}} \frac{\partial H_{i_{k-1}}^{(k-1)}}{\partial H_{i_{k-2}}^{(k-2)}} ... \frac{\partial H_{i_1}^{(1)}}{\partial H_x^{(0)}}. \end{aligned}$$

We have

$$\begin{aligned} \frac{\partial H_{i_{l+1},s}^{(l+1)}}{\partial H_{i_l,t}^{(l)}} &= \frac{\partial (AH^{(l)}W^{(l)})_{i_{l+1},s}}{\partial H_{i_l,t}^{(l)}} \\ &= \frac{\partial (\sum_{p=1} \sum_{q=1} A_{i_{l+1},p} H_{p,q}^{(l)} W_{q,s}^{(l)})}{\partial H_{i_l,t}^{(l)}} \\ &= A_{i_{l+1},i_l} W_{t,s}^{(l)} \\ \Rightarrow \frac{\partial H_{i_{l+1}}^{(l+1)}}{\partial H_{i_l}^{(l)}} &= A_{i_{l+1},i_l} \cdot [W^{(l)}]^{\mathrm{T}}, \text{for l =0,1,2,...,k-1} \end{aligned}$$

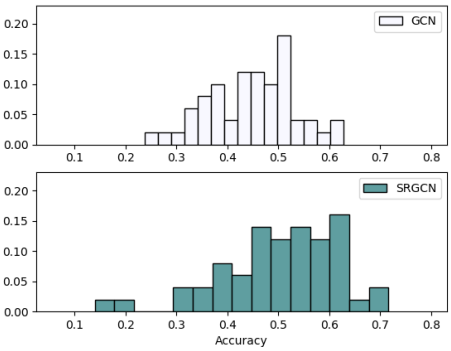
From (1), we get

$$\begin{aligned} \frac{\partial H_y^{(k)}}{\partial H_x^{(0)}} &= \sum_{i_{k-1} \in V} \sum_{i_{k-2} \in V} ... \sum_{i_1 \in V} A_{y,i_{k-1}} A_{i_{k-1},i_{k-2}} ... A_{i_1,x} [W^{(0)} W^{(1)} ... W^{(k-1)}]^{\mathrm{T}} \\ &= 0. \end{aligned}$$

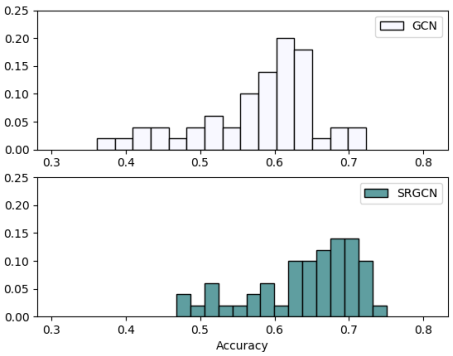
Therefore, $|| \frac{\partial H_y^{(k)}}{\partial H_x^{(0)}} ||_F = 0$. If the maximal degree of the nodes in graph is m, we have:

$$| N_k(z) | \leq m^k \text{and } | N_k(y) | \leq m^k. \qquad \square$$

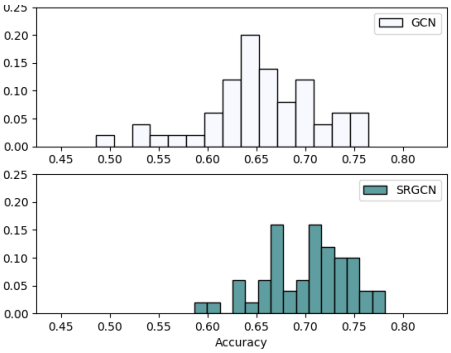
B MORE EXPERIMENTAL RESULTS



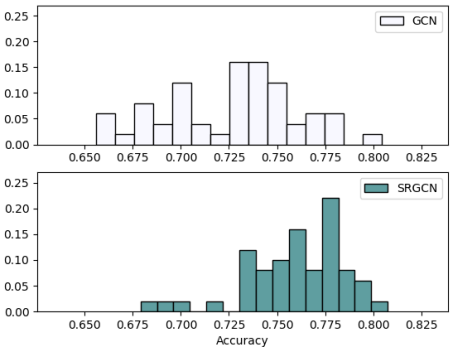
(a) Cora, sample size 10



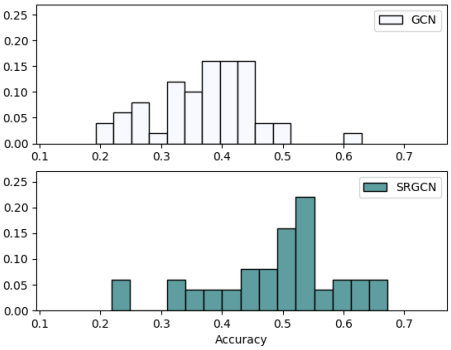
(b) Cora, sample size 20



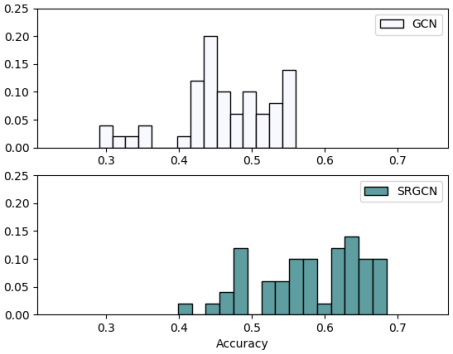
(c) Cora, sample size 30



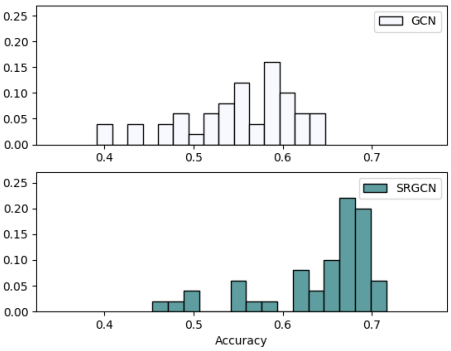
(d) Cora, sample size 50



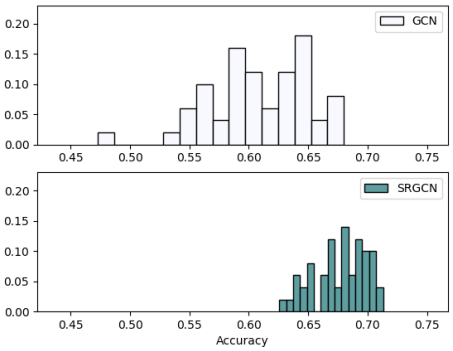
(e) Citeseer, sample size 10



(f) Citeseer, sample size 20



(g) Citeseer, sample size 30



(h) Citeseer, sample size 50