Graph-Revised Convolutional Network

Donghan Yu , **Ruohong Zhang** , **Zhengbao Jiang** , **Yuexin Wu** and **Yiming Yang**
Carnegie Mellon University
{dyu2,ruohongz,zhengbaj,yuexinw,yiming}@cs.cmu.edu

Abstract

Graph Convolutional Networks (GCNs) have received increasing attention in the machine learning community for effectively leveraging both the content features of nodes and the linkage patterns across graphs in various applications. As real-world graphs are often incomplete and noisy, treating them as ground-truth information, which is a common practice in most GCNs, unavoidably leads to sub-optimal solutions. Existing efforts for addressing this problem either involve an over-parameterized model which is difficult to scale, or simply re-weight observed edges without dealing with the missing-edge issue. This paper proposes a novel framework called Graph-Revised Convolutional Network (GRCN), which avoids both extremes. Specifically, a GCN-based graph revision module is introduced for predicting missing edges and revising edge weights w.r.t. downstream tasks via joint optimization. A theoretical analysis reveals the connection between GRCN and previous work on multigraph belief propagation. Experiments on six benchmark datasets show that GRCN consistently outperforms strong baseline methods, especially when the original graphs are severely incomplete or the labeled instances for model training are highly sparse[1].

1 Introduction

Graph Convolutional Networks (GCNs) have received increasing attention in recent years as they are highly effective in graph-based node feature induction and belief propagation, and widely applicable to many real-world problems, including computer vision [Wang *et al.*, 2018; Landrieu and Simonovsky, 2018], natural language processing [Kipf and Welling, 2016; Marcheggiani and Titov, 2017], recommender systems [Monti *et al.*, 2017; Ying *et al.*, 2018], epidemiological forecasting [Wu *et al.*, 2018], and more.

However, the power of GCNs has not been fully exploited as most of the models assume that the given graph perfectly depicts the ground-truth of the relationship between nodes. Such assumptions are bound to yield sub-optimal results as real-world graphs are usually highly noisy, incomplete (with many missing edges), and not necessarily ideal for different downstream tasks. Ignoring these issues is a fundamental weakness of many existing GCN methods.

Recent methods that attempt to modify the original graph can be split into two major streams: 1) Edge reweighting: GAT [Veličković *et al.*, 2017] and GLCN [Jiang *et al.*, 2019] use attention mechanism or feature similarity to reweight the existing edges of the given graph. Since the topological structure of the graph is not changed, the model is prone to be affected by noisy data when edges are sparse. 2) Full graph parameterization: LDS [Franceschi *et al.*, 2019], on the other hand, allows every possible node pairs in a graph to be parameterized. Although this design is more flexible, the memory cost is intractable for large datasets, since the number of parameters increases quadratically with the number of nodes. Therefore, finding a balance between model expressiveness and memory consumption remains an open challenge.

To enable flexible edge editing while maintaining scalability, we develop a GCN-based graph revision module that performs edge addition and edge reweighting. In each iteration, we calculate an adjacency matrix via GCN-based node embeddings, and select the edges with high confidence to be added. Our method permits a gradient-based training of an end-to-end neural model that can predict unseen edges. Our theoretical analysis demonstrates the effectiveness of our model from the perspective of multigraph [Balakrishnan, 1997], which allows more than one edges from different sources between a pair of vertices. To the best of our knowledge, we are the first to reveal the connection between graph convolutional networks and multigraph propagation. Our contributions can be summarized as follows:

- We introduce a novel structure that simultaneously learns both graph revision and node classification through different GCN modules.
- Through theoretical analysis, we show our model's advantages in the view of multigraph propagation.
- Comprehensive experiments on six benchmark datasets from different domains show that our proposed model achieves the best or highly competitive results, especially under the scenarios of highly incomplete graphs or sparse training labels.

[1]Our code is available in https://github.com/Maysir/GRCN.

2 Background

We first introduce some basics of graph theory. An undirected graph G can be represented as (V, E) where V denotes the set of vertices and E denotes the set of edges. Let N and M be the number of vertices and edges, respectively. Each graph can also be represented by an adjacency matrix A of size $N \times N$ where $A_{ij} = 1$ if there is an edge between v_i and v_j, and $A_{ij} = 0$ otherwise. We use A_i to denote the i-th row of the adjacency matrix. A graph with adjacency matrix A is denoted as G_A. Usually each node i has its own feature $x_i \in \mathbb{R}^F$ where F is the feature dimension (for example, if nodes represent documents, the feature can be a bag-of-words vector). The node feature matrix of the whole graph is denoted as $X \in \mathbb{R}^{N \times F}$.

Graph convolutional networks generalize the convolution operation on images to graph structure data, performing layer-wise propagation of node features [Bruna *et al.*, 2013; Kipf and Welling, 2016; Hamilton *et al.*, 2017; Veličković *et al.*, 2017]. Suppose we are given a graph with adjacency matrix A and node features $H^{(0)} = X$. An L-layer Graph Convolution Network (GCN) [Kipf and Welling, 2016] conducts the following inductive layer-wise propagation:

$$H^{(l+1)} = \sigma\left(\widetilde{D}^{-\frac{1}{2}}\widetilde{A}\widetilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right), \tag{1}$$

where $l = 0, 1, \cdots, L-1$, $\tilde{A} = A + I$ and $\widetilde{D}$ is a diagonal matrix with $D_{ii} = \sum_j \widetilde{A}_{ij}$. $\{W^{(0)}, \cdots, W^{(L-1)}\}$ are the model parameters and $\sigma(\cdot)$ is the activation function. The node embedding $H^{(L)}$ can be used for downsteam tasks. For node classification, GCN defines the final output as:

$$\widehat{Y} = \text{softmax}\left(H^{(L)}W^{(L)}\right). \tag{2}$$

where $\widehat{Y} \in \mathbb{R}^{N \times C}$ and C denotes the number of classes. We note that in the GCN computation, A is directly used as the underlining graph without any modification. Additionally, in each layer, GCN only updates node representations as a degree-normalized aggregation of neighbor nodes.

To allow for an adaptive aggregation paradigm, GLCN [Jiang *et al.*, 2019] learns to reweight the existing edges by node feature embeddings. The reweighted adjacancy matrix $\widetilde{A}$ is calculated by:

$$\widetilde{A}_{ij} = \frac{A_{ij}\exp\left(\text{ReLU}\left(a^T\left|x_iP - x_jP\right|\right)\right)}{\sum_{k=1}^{n} A_{ik}\exp\left(\text{ReLU}\left(a^T\left|x_iP - x_kP\right|\right)\right)}, \tag{3}$$

where x_i denotes the feature vector of node i and a, P are model parameters. Another model GAT [Veličković *et al.*, 2017] reweights edges by a layer-wise self-attention across node-neighbor pairs to compute hidden representations. For each layer l, the reweighted edge is computed by:

$$\widetilde{A}_{ij}^{(l)} = \frac{A_{ij}\exp\left(a(W^{(l)}H_i^{(l)}, W^{(l)}H_j^{(l)})\right)}{\sum_{k=1}^{n} A_{ik}\exp\left(a(W^{(l)}H_i^{(l)}, W^{(l)}H_k^{(l)})\right)} \tag{4}$$

where $a(\cdot,\cdot)$ is a shared attention function to compute the attention coefficients. Compared with GLCN, GAT uses different layer-wise maskings to allow for more flexible representation. However, neither of the methods has the ability to add edges since the revised edge $\widetilde{A}_{ij}$ or $\widetilde{A}_{ij}^{(l)} \neq 0$ only if the original edge $A_{ij} \neq 0$.

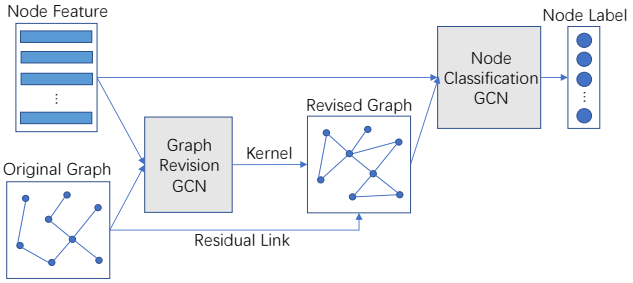


Figure 1: Architecture of the proposed GRCN model for semi-supervised node classification. The node classification GCN is enhanced with a revised graph constructed by the graph revision GCN module.

In order to add new edges into the original graph, LDS [Franceschi *et al.*, 2019] makes the entire adjacency matrix parameterizable. Then it jointly learns the graph structure θ and the GCN parameters W by approximately solving a bilevel program as follows:

$$\begin{aligned} &\min_{\theta \in \overline{\mathcal{H}}_N} \mathbb{E}_{A \sim \text{Ber}(\theta)}\left[\zeta_{val}\left(W_\theta, A\right)\right], \\ &\text{such that } W_\theta = \arg\min_W \mathbb{E}_{A \sim \text{Ber}(\theta)}[\zeta_{train}(W, A)], \end{aligned} \tag{5}$$

where $A \sim \text{Ber}(\theta)$ means sampling adjacency matrix $A \in \mathbb{R}^{N \times N}$ from Bernoulli distribution under parameter $\theta \in \mathbb{R}^{N \times N}$. $\overline{\mathcal{H}}_N$ is the convex hull of the set of all adjecency matrices for N nodes. ζ_{train} and ζ_{val} denote the node classification loss on training and validation data respectively. However, this method can hardly scale to large graphs since the parameter size of θ is N^2 where N is the number of nodes. In the next section, we'll present our method which resolves the issues in previous work.

3 Proposed Method

3.1 Graph-Revised Convolutional Network

Our Graph-Revised Convolutional Network (GRCN) contains two modules: a *graph revision* module and a *node classification* module. The graph revision module adjusts the original graph by adding or reweighting edges, and the node classification module performs classification using the revised graph. Specifically, in our graph revision module, we choose to use a GCN to combine the node features and the original graph input, as GCNs are effective at fusing data from different sources [Wu *et al.*, 2019b]. We first learn the node embedding $Z \in \mathbb{R}^{N \times D}$ as follows:

$$Z = GCN_g(A, X) \tag{6}$$

where GCN_g denotes the graph convolutional network for graph revision, A is the original graph adjacency matrix and X is node feature. Then we calculate a similarity graph S based on node embedding using certain kernel function $k : \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}$:

$$S_{ij} = k(z_i, z_j). \tag{7}$$

The revised adjacency matrix is formed by an elementwise summation of the original adjacency matrix and the calculated similarity matrix: $\widetilde{A} = A+S$. Compared with the graph revision in GAT and GLCN which use entrywise product, we instead adopt the entrywise addition operator "+" in order for new edges to be considered. In this process, the original graph A is revised by the similarity graph S, which can insert new edges to A and potentially reweight or delete existing edges in A. In practice, we apply a *sparsification* technique on dense matrix S to reduce computational cost and memory usage, which will be introduced in the next section. Then the predicted labels are calculated by:

$$\widehat{Y} = GCN_c(\widetilde{A}, X) \tag{8}$$

where GCN_c denotes the graph convolutional network for the downstream node classification task. Note that to prevent numerical instabilities, renormalization trick [Kipf and Welling, 2016] is also applied on the revised adjacency matrix $\widetilde{A}$. Figure 1 provides an illustration of our model. Finally, we use cross-entropy loss as our objective function:

$$\zeta = -\sum_{i \in \mathcal{Y}_L} \sum_{j=1}^{C} Y_{ij} ln \widehat{Y}_{ij} \tag{9}$$

where $\mathcal{Y}_L$ is the set of node indices that have labels Y and C is the number of classes. It's worth emphasizing that our model does not need other loss functions to guide the graph revision process.

Overall, our model can be formulated as:

$$\begin{aligned} &GRCN(A, X) = GCN_c(\widetilde{A}, X), \\ &\widetilde{A} = A + K(GCN_g(A, X)), \end{aligned} \tag{10}$$

where $K(\cdot)$ is the kernel matrix computed from the node embeddings in Equation (6). In our implementation, we use dot product as kernel function for simplicity, and we use a two-layer GCN [Kipf and Welling, 2016] in both modules. We leave other kernel functions and graph convolutional networks for future exploration.

3.2 Sparsification

Since the adjacency matrix S of similarity graph is dense, directly applying it in the classification module is inefficient. Besides, we only want those edges with higher confidence to avoid introducing too much noise. Thus we conduct a K-nearest-neighbour (KNN) sparsification on the dense graph: for each node, we keep the edges with top-K prediction scores. The adjacancy matrix of the KNN-sparse graph, denoted as $S^{(K)}$, is computed as:

$$S_{ij}^{(K)} = \begin{cases} S_{ij}, & S_{ij} \in topK(S_i), \\ 0, & S_{ij} \notin topK(S_i). \end{cases} \tag{11}$$

where $topK(S_i)$ is the set of top-K values of vector S_i. Finally, in order to keep the symmetric property, the output sparse graph $\widehat{S}$ is calculated by:

$$\widehat{S}_{ij} = \begin{cases} \max(S_{ij}^{(K)}, S_{ji}^{(K)}), & S_{ij}^{(K)}, S_{ij}^{(K)} \geq 0 \\ \min(S_{ij}^{(K)}, S_{ji}^{(K)}), & S_{ij}^{(K)}, S_{ij}^{(K)} \leq 0 \end{cases} \tag{12}$$

Now since both original graph A and similarity graph $\widehat{S}$ are sparse, efficient matrix multiplication can be applied on both GCNs as in the training time, gradients will only backpropagate through the top-K values. By sparsification, the memory cost of similarity matrix is reduced from $O(N^2)$ to $O(NK)$.

3.3 Fast-GRCN

To further reduce the training time, we introduce a faster version of the propose model: Fast-GRCN. Note that GRCN needs to compute the dense adjacency matrix S in every epoch, where the computational time complexity is $O(N^2)$. While in Fast-GRCN, the whole matrix S is only calculated in the first epoch, and then the indices of the non-zero values of the KNN-sparse matrix $S^{(K)}$ are saved. For the remaining epochs, to obtain $S^{(K)}$, we only compute the values of the saved indices while directly setting zeros for other indices, which reduces the time complexity to $O(NK)$.

3.4 Theoretical Analysis

In this section, we show the effectiveness of our model in the view of Multigraph [Balakrishnan, 1997] propagation. The major observation is that for existing methods, the learned function from GCNs can be regarded as a linear combination of limited pre-defined kernels where the flexibility of kernels have a large influence on the final prediction accuracy.

We consider the simplified graph convolution neural network GCN_s for the ease of analysis. That is, we remove feature transformation parameter W and non-linear activation function $\sigma(\cdot)$ as:

$$GCN_s(A, X) = A^k X \tag{13}$$

where k is the number of GCN layers. For simplicity we denote A as the adjacency matrix with self-loop after normalization. The final output can be acquired by applying a linear or logistic regression function $f(\cdot)$ on the node embeddings above:

$$\widehat{Y} = f(GCN_s(A, X)) = f(A^k X) \tag{14}$$

where $\widehat{Y}$ denotes the predicted labels of nodes. Then the following theorem shows that under certain conditions, the optimal function f^* can be expressed as a linear combination of kernel functions defined on training samples.

Theorem 1 (Representer Theorem [Schölkopf *et al.*, 2001])**.** *Consider a non-empty set $\mathcal{P}$ and a positive-definite real-valued kernel: $k : \mathcal{P} \times \mathcal{P} \to \mathbb{R}$ with a corresponding reproducing kernel Hilbert space H_k. If given: a. a set of training samples $\{(p_i, y_i) \in \mathcal{P} \times \mathbb{R} | i = 1, \cdots, n\}$; b. a strictly monotonically increasing real-valued function $g : [0, \infty) \to \mathbb{R}$; and c. an error function $E : (\mathcal{P} \times \mathbb{R}^2)^n \to \mathbb{R} \cup \{\infty\}$, which together define the following regularized empirical risk functional on H_k: $f \mapsto E((p_1, y_1, f(p_1)), \ldots, (p_n, y_n, f(p_n))) + g(\|f\|)$. Then, any minimizer of the empirical risk admits a representation of the form: $f^*(\cdot) = \sum_{i=1}^{n} \alpha_i k(\cdot, p_i)$ where $a_i \in \mathbb{R}\ \forall i = 1, \cdots, n$.*

In our case, $p_i \in \mathbb{R}^D$ is the embedding of node i. As shown in the theorem, the final optimized output is the linear combination of certain kernels on node embeddings. We assume the

kernel function to be dot product for simplicity, which means $k(p_i, p_j) = p_i^T p_j$. The corresponding kernel matrix can be written as:

$$K(GCN_s(A, X)) = A^k X X^T A^k = A^k B A^k \tag{15}$$

where $B = XX^T$ is the adjacency matrix of graph induced by node features. Now we have two graphs based on the same node set: original graph G_A (associated with adjacency matrix A) and feature graph G_B (associated with adjacency matrix B). They form a multigraph [Balakrishnan, 1997] where multiple edges is permitted between the same end nodes. Then the random-walk-like matrix $A^k B A^k$ can be regarded as one way to perform graph label/feature propagation on the multigraph. Its limitation is obvious: the propagation only happens once on the feature graph G_B, which lacks flexibility. However, for our method, we have:

$$\begin{aligned} GRCN(A, X) =& (A + K(GCN_s(A, X)))^k X \\ =& (A + A^m X X^T A^m)^k X \\ =& (A + A^m B A^m)^k X, \\ K(GRCN(A, X)) =& (A + A^m B A^m)^k B \\ & (A + A^m B A^m)^k, \end{aligned} \tag{16}$$

where labels/features can propagate multiple times on the feature graph G_B. Thus our model is more flexible and more effective especially when the original graph G_A is not reliable or cannot provide enough information for downstream tasks. In Equation (16), $A + A^m B A^m$ can be regarded as a combination of different edges in the multigraph. To reveal the connection between GRCN and GLCN [Jiang *et al.*, 2019], we first consider the special case of our model that $m = 0$: $GRCN(A, X) = (A + B)^k X$. The operator "+" is analogous to the operator OR which incorporates information from both graph A and B. While GLCN [Jiang *et al.*, 2019] takes another combination denoted as $A \circ B$ using Hadamard (entrywise) product "$\circ$", which can be analogous to AND operation.

We can further extend our model to a layer-wise version for comparison to GAT [Veličković *et al.*, 2017]. More specifically, for the l-th layer, we denote the input as X_l. The output X_{l+1} is then calculated by:

$$\begin{aligned} X_{l+1} =& (A + K(GCN_s(A, X_l)))X_l \\ =& (A + A^m X_l X_l^T A^m)X_l \\ =& (A + A^m B_l A^m)X_l, \end{aligned} \tag{17}$$

where $B_l = X_l X_l^T$. Similar to the analysis mentioned before, if we consider the special case of GRCN that $m = 0$ and change the edge combination operator from entrywise sum "+" to entrywise product "$\circ$", we have $X_{l+1} = (A \circ B_l)X_l$, which is the key idea behind GAT [Veličković *et al.*, 2017]. Due to the property of entrywise product, the combined edges of both GAT and GLCN are only the reweighted edges of A, which becomes ineffective when the original graph G_A is highly sparse. Through the analysis above, we see that our model is more general in combining different edges by varying the value of m, and also has more robust combination operator "+" compared to previous methods.

Dataset	#nodes	#edges	#feature	#class
Cora	2708	5429	1433	7
CiteSeer	3327	4732	3703	6
PubMed	19717	44338	500	3
CoraFull	19793	65311	8710	70
Amazon Computers	13381	245778	767	10
Coauthor CS	18333	81894	6805	15

Table 1: Data statistics

4 Experiments

We evaluate the proposed model on semi-supervised node classification tasks, and conduct extensive experimental analysis in the following sections.

4.1 Datasets and Baselines

We use six benchmark datasets for semi-supervised node classification evaluation. Among them, Cora, CiteSeer [Sen *et al.*, 2008] and PubMed [Namata *et al.*, 2012] are three commonly used datasets. The data split is conducted by two ways. The first is the fixed split originating from [Yang *et al.*, 2016]. In the second way, we conduct 10 random splits while keeping the same number of labels for training, validation and testing as previous work. This provides a more robust comparison of the model performance. To further test the scalability of our model, we utilize three other datasets: Cora-Full [Bojchevski and Günnemann, 2018], Amazon-Computers and Coauthor CS [Shchur *et al.*, 2018]. The first is an extended version of Cora, while the second and the third are co-purchase and co-authorship graphs respectively. On these three datasets, we follow the previous work [Shchur *et al.*, 2018] and take 20 labels of each classes for training, 30 for validation, and the rest for testing. We also delete the classes with less than 50 labels to make sure each class contains enough instances. The data statistics are shown in Table 1.

We compare the effectiveness of our GRCN model with several baselines, where the first two models are vanilla graph convolutional networks without any graph revision: GCN [Kipf and Welling, 2016], SGC [Wu *et al.*, 2019a], GAT [Veličković *et al.*, 2017], LDS [Franceschi *et al.*, 2019], and GLCN [Jiang *et al.*, 2019].

4.2 Implementation Details

Transductive setting is used for node classification on all the datasets. We train GRCN for 300 epochs using Adam [Kingma and Ba, 2014] and select the model with highest validation accuracy for test. We set learning rate as 1×10^{-3} for graph refinement module and 5×10^{-3} for label prediction module. Weight decay and sparsification parameter K are tuned by grid search on validation set, with the search space $[1 \times 10^{-4}, 5 \times 10^{-4}, 1 \times 10^{-3}, 5 \times 10^{-3}, 1 \times 10^{-2}, 5 \times 10^{-2}]$ and $[5, 10, 20, 30, 50, 100, 200]$ respectively. Our code is based on Pytorch [Paszke *et al.*, 2017] and one geometric deep learning extension library [Fey and Lenssen, 2019], which provides implementation for GCN, SGC and GAT. For LDS, the results were obtained using the publicly available code. Since an implementation for GLCN

	Cora (fix. split)	CiteSeer (fix. split)	PubMed (fix. split)
GCN	81.4 ± 0.5	70.9 ± 0.5	$\mathbf{79.0 \pm 0.3}$
SGC	81.0 ± 0.0	71.9 ± 0.1	$78.9 \pm 0.0^*$
GAT	83.2 ± 0.7	72.6 ± 0.6	78.8 ± 0.3
LDS	$84.0 \pm 0.4^*$	$\mathbf{74.8 \pm 0.5}$	N/A
GLCN	81.8 ± 0.6	70.8 ± 0.5	78.8 ± 0.4
Fast-GRCN	83.6 ± 0.4	72.9 ± 0.6	$\mathbf{79.0 \pm 0.2}$
GRCN	$\mathbf{84.2 \pm 0.4}$	$73.6 \pm 0.5^*$	$\mathbf{79.0 \pm 0.2}$
	Cora (rand. splits)	CiteSeer (rand. splits)	PubMed (rand. splits)
GCN	81.2 ± 1.9	69.8 ± 1.9	$77.7 \pm 2.9^*$
SGC	81.0 ± 1.7	68.9 ± 2.0	75.8 ± 3.0
GAT	81.7 ± 1.9	68.8 ± 1.8	$77.7 \pm 3.2^*$
LDS	81.6 ± 1.0	71.0 ± 0.9	N/A
GLCN	81.4 ± 1.9	69.8 ± 1.8	77.2 ± 3.2
Fast-GRCN	$\mathbf{83.8 \pm 1.6}$	$72.3 \pm 1.4^*$	77.6 ± 3.2
GRCN	$83.7 \pm 1.7^*$	$\mathbf{72.6 \pm 1.3}$	$\mathbf{77.9 \pm 3.2}$
	Cora-Full	Amazon Computers	Coauthor CS
GCN	$\mathbf{60.3 \pm 0.7}$	81.9 ± 1.7	$\mathbf{91.3 \pm 0.3}$
SGC	59.1 ± 0.7	81.8 ± 2.3	$\mathbf{91.3 \pm 0.2}$
GAT	59.9 ± 0.6	81.8 ± 2.0	89.5 ± 0.5
LDS	N/A	N/A	N/A
GLCN	59.1 ± 0.7	80.4 ± 1.9	90.1 ± 0.5
Fast-GRCN	$60.2 \pm 0.5^*$	$83.5 \pm 1.6^*$	$91.2 \pm 0.4^*$
GRCN	$\mathbf{60.3 \pm 0.4}$	$\mathbf{83.7 \pm 1.8}$	$\mathbf{91.3 \pm 0.3}$

Table 2: Mean test classification accuracy and standard deviation in percent averaged for all models and all datasets. For each dataset, the highest accuracy score is marked in **bold**, and the second highest score is marked using $*$. N/A stands for the datasets that couldn't be processed by the full-batch version because of GPU RAM limitations.

was not available, we report the results based on our own implementation of the original paper.

4.3 Main Results

Table 2 shows the mean accuracy and the corresponding standard deviation for all models across the 6 datasets averaged over 10 different runs[2]. We see that our proposed models achieve the best or highly competitive results for all the datasets. The effectiveness of our model over the other baselines demonstrates that taking the original graph as input for GCN is not optimal for graph propagation in semi-supervised classification. It's also worth noting that Fast-GRCN, with much lower computational time complexity, performs nearly as well as GRCN .

To further test the superiority of our model, we consider the edge-sparse scenario when a certain fraction of edges in the given graph is randomly removed. Given an edge retaining ratio, we randomly sample the retained edges

[2]Note that the performance difference of baseline models between fixed split and random split is also observed in previous work [Shchur *et al.*, 2018], where they show that different splits can lead to different ranking of models, and suggest multiple random splits as a better choice. Our later experiments are based on the random split setting.

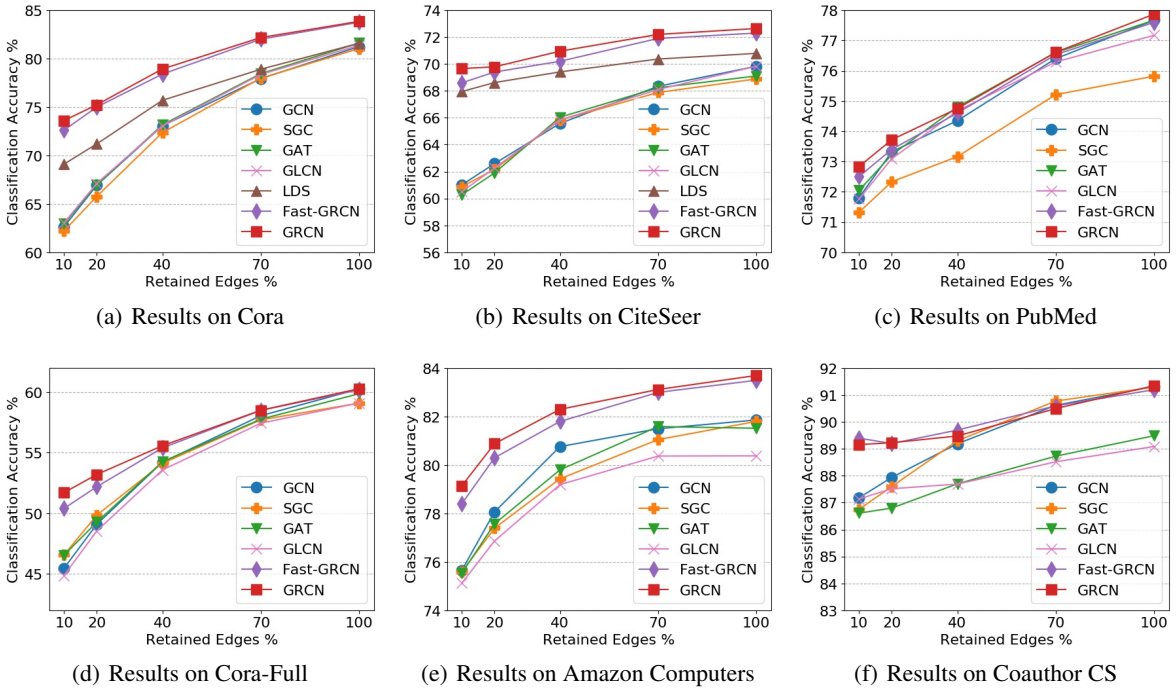
Cora-Full	5 labels	10 labels	15 labels
GCN	31.3 ± 1.5	41.1 ± 1.3	46.0 ± 1.1
SGC	31.5 ± 2.1	42.0 ± 1.5	46.8 ± 1.3
GAT	32.5 ± 2.1	41.2 ± 1.4	45.5 ± 1.2
GLCN	30.9 ± 1.9	41.0 ± 0.6	45.0 ± 0.9
Fast-GRCN	$41.2 \pm 1.2^*$	$47.7 \pm 0.8^*$	$50.6 \pm 1.0^*$
GRCN	$\mathbf{42.3 \pm 0.8}$	$\mathbf{48.2 \pm 0.7}$	$\mathbf{51.8 \pm 0.6}$
Amazon Computers			
GCN	70.5 ± 3.3	74.6 ± 2.3	77.2 ± 2.2
SGC	67.2 ± 5.0	74.6 ± 4.6	77.1 ± 1.6
GAT	64.6 ± 8.9	72.5 ± 4.5	74.2 ± 2.7
GLCN	66.9 ± 7.1	73.8 ± 3.6	75.8 ± 2.2
Fast-GRCN	$74.1 \pm 1.9^*$	$78.6 \pm 2.1^*$	$79.8 \pm 1.5^*$
GRCN	$\mathbf{75.3 \pm 1.2}$	$\mathbf{79.1 \pm 1.9}$	$\mathbf{79.9 \pm 1.6}$
Coauthor CS			
GCN	82.2 ± 1.5	86.1 ± 0.5	87.1 ± 0.9
SGC	81.5 ± 1.6	85.7 ± 0.9	86.7 ± 0.9
GAT	80.7 ± 1.1	84.8 ± 1.0	86.0 ± 0.7
GLCN	82.7 ± 0.7	85.7 ± 0.6	87.0 ± 0.8
Fast-GRCN	$\mathbf{86.2 \pm 2.1}$	$\mathbf{88.2 \pm 0.7}$	$\mathbf{88.5 \pm 0.6}$
GRCN	$86.1 \pm 0.7^*$	$87.9 \pm 0.4^*$	$88.2 \pm 0.5^*$

Table 3: Mean test classification accuracy and standard deviation on Cora-Full, Amazon Computers and Coauthor CS datasets under different number of training labels for each class. The edge retaining ratio is 20% for all the results. For each dataset, the highest accuracy score is marked in **bold**, and the second highest score is marked using $*$.

10 times and report the mean classification accuracy and standard deviation. For the Cora, CiteSeer and PubMed dataset, we conduct experiments under random data split. Figure 2 shows the results under different ratios of retained edges. There are several observations from this figure. First, GRCN and Fast-GRCN achieve notable improvements on almost all the datasets, especially when edge retaining ratio is low. For instance, when edge retaining ratio is 10%, GRCN outperforms the second best baseline model by $6.5\%, 2.5\%, 1.1\%, 11.0\%, 4.6\%, 2.3\%$ on each dataset. Second, the GAT and GLCN models which reweight the existing edges do not perform well, indicating that such a reweighting mechanism is not enough when the original graph is highly incomplete. Third, our method also outperforms the over-parameterized model LDS in Cora and CiteSeer because of our restrained edge editing procedure. Though LDS achieves better performances than other baseline methods in these two datasets, its inability to scale prevents us from testing it on four of the larger datasets.

4.4 Robustness on Training Labels

We also show that the gains achieved by our model are very robust to the reduction in the number of training labels for each class, denoted by T. We compare all the models on the Cora-Full, Amazon Computers and Coauthor CS datasets and fix the edge sampling ratio to 20%. We reduce T from 15 to 5 and report the results in Table 3. While containing more parameters than vanilla GCN, our model still outperforms others. Moreover, it wins by a larger margin when T is smaller.



(a) Results on Cora (b) Results on CiteSeer (c) Results on PubMed

(d) Results on Cora-Full (e) Results on Amazon Computers (f) Results on Coauthor CS

Figure 2: Mean test classification accuracy on all the datasets under different ratios of retained edges over 10 different runs.

This demonstrates our model's capability to handle tasks with sparse training labels.

4.5 Ablation Study

To further examine the effectiveness of our GCN-based graph revision module, we conduct an ablation study by testing four different simplifications of the graph revision module:

- Truncated SVD Reconstruction [Golub and Reinsch, 1971]: $\widetilde{A} = \text{SVD}_k(A)$
- Feature-Only (FO): $\widetilde{A} = K(X)$
- Feature plus Graph (FG): $\widetilde{A} = A + K(X)$
- Random Walk Feature plus Graph (RWFG): $\widetilde{A} = A + K(A^2X)$

where SVD and FO only use the original graph structure or node features to construct the graph. They are followed by the FG method, which adds the original graph to the feature similarity graph used in FO. Our model is most closely related to the third method, RWFG, which constructs the feature graph with similarity of node features via graph propagation, but without feature learning.

We conduct the ablation experiment on Cora dataset with different edge retaining ratios and report the results in Figure 3. The performance of SVD method indicates that simply smoothing the original graph is insufficient. The comparison between FO and FG shows that adding original graph as residual links is helpful for all edge retaing ratios, especially when there are more known edges in the graph. Examining the results of FG and RWFG, we can also observe a large improvement brought by graph propagation on features. Finally, the performance of our model and RWFG underscores the importance of feature learning, especially in the cases of low edge retraining ratio.

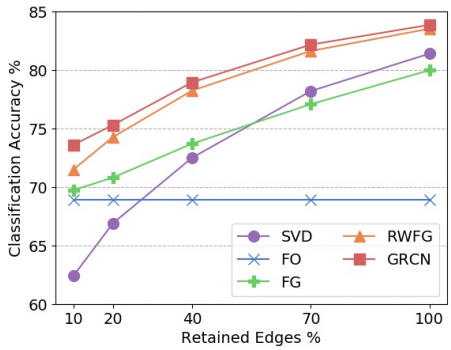


Figure 3: Results of GRCN and its simplified versions on Cora dataset with different ratios of retained edges

5 Conclusion

This paper presents Graph-Revised Convolutional Network, a novel framework for incorporating graph revision into graph convolution networks. We show both theoretically and experimentally that the proposed way of graph revision can significantly enhance the prediction accuracy for downstream tasks. GRCN overcomes two main drawbacks in previous approaches to graph revision, which either employ over-parameterized models and consequently face scaling issues,

or fail to consider missing edges. In our experiments with node classification tasks, the performance of GRCN stands out in particular when the input graphs are highly incomplete or if the labeled training instances are very sparse.

In the future, we plan to explore GRCN in a broader range of prediction tasks, such as knowledge base completion, epidemiological forecasting and aircraft anomaly detection based on sensor network data.

References

[Balakrishnan, 1997] VK Balakrishnan. Graph theory (schaum's outline), 1997.

[Bojchevski and Günnemann, 2018] Aleksandar Bojchevski and Stephan Günnemann. Deep gaussian embedding of graphs: Unsupervised inductive learning via ranking. In *International Conference on Learning Representations*, pages 1–13, 2018.

[Bruna *et al.*, 2013] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*, 2013.

[Fey and Lenssen, 2019] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*, 2019.

[Franceschi *et al.*, 2019] Luca Franceschi, Mathias Niepert, Massimiliano Pontil, and Xiao He. Learning discrete structures for graph neural networks. *arXiv preprint arXiv:1903.11960*, 2019.

[Golub and Reinsch, 1971] Gene H Golub and Christian Reinsch. Singular value decomposition and least squares solutions. In *Linear Algebra*, pages 134–151. Springer, 1971.

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

[Jiang *et al.*, 2019] Bo Jiang, Ziyan Zhang, Doudou Lin, Jin Tang, and Bin Luo. Semi-supervised learning with graph learning-convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 11313–11320, 2019.

[Kingma and Ba, 2014] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[Kipf and Welling, 2016] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[Landrieu and Simonovsky, 2018] Loic Landrieu and Martin Simonovsky. Large-scale point cloud semantic segmentation with superpoint graphs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4558–4567, 2018.

[Marcheggiani and Titov, 2017] Diego Marcheggiani and Ivan Titov. Encoding sentences with graph convolutional networks for semantic role labeling. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1506–1515, 2017.

[Monti *et al.*, 2017] 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 *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5115–5124, 2017.

[Namata *et al.*, 2012] Galileo Namata, Ben London, Lise Getoor, Bert Huang, and UMD EDU. Query-driven active surveying for collective classification. In *10th International Workshop on Mining and Learning with Graphs*, page 8, 2012.

[Paszke *et al.*, 2017] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. In *NIPS Autodiff Workshop*, 2017.

[Schölkopf *et al.*, 2001] Bernhard Schölkopf, Ralf Herbrich, and Alex J Smola. A generalized representer theorem. In *International conference on computational learning theory*, pages 416–426. Springer, 2001.

[Sen *et al.*, 2008] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008.

[Shchur *et al.*, 2018] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868*, 2018.

[Veličković *et al.*, 2017] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[Wang *et al.*, 2018] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic graph cnn for learning on point clouds. *arXiv preprint arXiv:1801.07829*, 2018.

[Wu *et al.*, 2018] Yuexin Wu, Yiming Yang, Hiroshi Nishiura, and Masaya Saitoh. Deep learning for epidemiological predictions. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, pages 1085–1088. ACM, 2018.

[Wu *et al.*, 2019a] Felix Wu, Tianyi Zhang, Amauri Holanda de Souza Jr, Christopher Fifty, Tao Yu, and Kilian Q Weinberger. Simplifying graph convolutional networks. *arXiv preprint arXiv:1902.07153*, 2019.

[Wu *et al.*, 2019b] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S Yu. A comprehensive survey on graph neural networks. *arXiv preprint arXiv:1901.00596*, 2019.

[Yang *et al.*, 2016] Zhilin Yang, William W Cohen, and Ruslan Salakhutdinov. Revisiting semi-supervised learning

with graph embeddings. *arXiv preprint arXiv:1603.08861*, 2016.

[Ying *et al.*, 2018] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 974–983. ACM, 2018.