STEPGCN: STEP-ORIENTED GRAPH CONVOLUTIONAL NETWORKS IN REPRESENTATION LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Graph convolutional networks (GCNs) are employed to address a number of tasks in the society through their representation learning approach. Nonetheless, despite their effectiveness and usefulness, the majority of GCN-oriented approaches face the over-smoothing problem. Over-smoothing is the problem of node representations converging into a certain value, making the nodes indistinguishable. To effectively address the over-smoothing problem, we introduce StepGCN, a GCN model that integrates step learning techniques with graph residual connection networks. As a result, we achieved significant performance improvements in multiple representations learning benchmark datasets, demonstrating that step learning can be expanded to other graph networks.

1 INTRODUCTION

Graph convolutional networks (GCNs) are one of the important research trends in a field of representation learning for graph-structured datasets (Kipf & Welling, 2017). Thus, a number of scholars have improved GCNs with unique techniques and approaches (Chen et al., 2018; Hamilton et al., 2017; Veličković et al., 2018). With these improvements, GCNs are employed in a number of research topics, which are represented by graph-oriented structures, such as social networks (Bian et al., 2020; Wu et al., 2019), recommendation systems (Ding et al., 2022; Yang et al., 2020; Zhang et al., 2019), and object detections (Li et al., 2020; Liu et al., 2020b; Wang et al., 2021).

However, despite these improvements, the majority of GCN-oriented approaches face the over-smoothing concern, which converges node representations to certain value, making them indistinguishable (Li et al., 2018; 2019; Liu et al., 2020a). Over-smoothing is reported as one of the common phenomena in deep GCNs (Chen et al., 2020a; Li et al., 2018). GCNs update node representations by aggregating messages passed from neighbor nodes. However, in this procedure, noise is presented during message passing, while a low information-to-noise ratio of received messages leads to over-smoothing problem (Chen et al., 2020a). In addition, due to layer stacking, Laplacian smoothing is repeated, which is one reason for over-smoothing problems (Li et al., 2018).

Considering that a number of recent deep neural network frameworks achieved powerful performance by increasing the depth of their architectures (He et al., 2016; Szegedy et al., 2015), deeper architectures can also be helpful for GCNs. Since shallow GCNs could not extract favorable information from nodes far hops away, deeper GCNs are vital for conducting more accurate representation learning. Thus, to make GCNs deeper, it is necessary to overcome over-smoothing. To alleviate over-smoothing, several unique techniques and methods are proposed (e.g., the integrated method of co-training and self-training procedures (Li et al., 2018), normalization layers (Zhao & Akoglu, 2019), application of transformation, and propagation (Liu et al., 2020a)).

He et al. (2016) addressed a comparable problem by suggesting residual connections considering visual-oriented tasks. Kipf & Welling (2017) attempted to apply residual connections to GCNs but indicated that examining residual connections is not one of the absolute solutions for the over-smoothing problem. Xu et al. (2018) introduced jumping knowledge network (JKNet), which aggregates the outputs of each layer at the last layer and then combines them by concatenating, max-pooling, and using LSTM-attention mechanism that computes the importance of the features. Li et al. (2019) applied residual connections at GCNs using both vertex-wise addition and vertex-wise concatenation, and they used dilated aggregation to enlarge the receptive fields of deep GCNs. Graph residual network (GResNet) was proposed by examining the correlations between graph

nodes (Zhang & Meng, 2019). Zhang & Meng (2019) focused on the correlations between nodes and pointed out that the residual connections of CNNs assumed the independence of data. However, when the graph nodes were closely correlated, they applied a normalized adjacency matrix to residual connections. Li & King (2020) proposed GCNII, using two techniques, namely, initial residual connection and identity mapping, to alleviate the over-smoothing problem. They used the information from the initial layer, not the previous one, to apply residual connection, and added identity matrix to weight matrix, allowing regularization and application to semi-supervised learning.

A number of scholars introduced several solutions to address the over-smoothing problems in utilizing GCNs; however, there is still room to effectively and simply address these issues. Therefore, we propose new concise and compatible approaches to stack additional layers in GCNs, called step graph convolutional network (StepGCN). We consider several traditional graph residual connection networks (GRCNs) as our base architectures. We then introduce a step learning technique, that is, a method to improve model performance considering the hops of neighbor nodes. At each step, ResBlock, an additional block extracting information from multi-hop, is attached to the prior model, and the learning rate, which controls the volume of information that is passed from neighbors, is decreased by a certain amount.

We empirically demonstrate that StepGCN successfully conducts representation learning by extracting distinctive information from similar receptive fields according to position in graph-structured data. In general, the model performance of StepGCN is better compared to those of other GCN-based models. We also indicate that step learning techniques can simply be applied to other GCN-based models. Our implementation code is publicly available[1].

2 PRELIMINARIES

2.1 GRAPH CONVOLUTIONAL NETWORK

GCN successfully employed convolutional operation (LeCun et al., 1995) to graph-structured data (Kipf & Welling, 2017). For a given undirected graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ with vertex set $\mathcal{V} = \{v_i\}_{i=1}^{N}$, which contains feature $X \in \mathbb{R}^{N \times d}$ and edge set $\mathcal{E} = \{e_i\}_{i=1}^{E}$, the graph convolutional operation can be described as follows:

$$, H^{(l+1)} = \sigma(\tilde{A}H^{(l)}W^{(l)}). \tag{1}$$

Note that $H^{(l+1)}$ is the output of the l-th layer and $H^{(0)} = X$ is input features. $\tilde{A} = \hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}$ is a normalized adjacency matrix, where $\hat{A} = A + I$ is an adjacency matrix of graph $\mathcal{G}$ with self-loop, and $\hat{D}$ denotes its degree matrix with $\hat{D_{ii}} = \sum_j \hat{a_{ij}}$. $W^{(l)} \in \mathbb{R}^{d_{in}^{(l)} \times d_{out}^{(l)}}$ contains trainable parameters of the l-th layer. σ is a nonlinear activation function such as ReLU.

The graph convolutional operation can be decomposed into two steps, namely, aggregation step (or message passing step) and feature extraction step. $\tilde{A}H^{(l)}$ in Equation 1 is an aggregation step, which aggregates information from neighbor nodes in single hop, while $H^{(l)}W^{(l)}$ is a feature extraction step, which extracts a feature of a target node from aggregated information in the prior step.

2.2 RESIDUAL CONNECTION

He et al. (2016) advanced the depth of deep convolutional neural networks (CNNs) by attaching shortcuts from the inputs to outputs of each layer, called residual connection. A layer $H(x)$ with residual connection can be represented as follows:

$$H(x) = F(x) + R(x), \tag{2}$$

where $F(x)$ denotes the original computational layer and the residual term $R(x)$ connects the input to the output, where He et al. (2016) used $R(x) = x$. It showed great performance in computer vision tasks. However, that was not the case with graph-structured data. Zhang & Meng (2019) claimed

[1] https://anonymous.4open.science/r/StepGCN-7709/

that this naive residual connection could not consider connectivity among the nodes. Therefore, they proposed graph residual connection, which considers connectivity of the nodes with normalized adjacency matrix. GCNs with graph residual connection can be computed using

$$H^{(l+1)} = \sigma(\tilde{A}H^{(l)}W^{(l)}) + R(H^{(l)}, X; G), \tag{3}$$

where $R(H^{(l)}, X; G) = \tilde{A}H^{(l)}$ for the most simple graph residual connection, and nonlinearity σ can also be applied including $R(H^{(l)}, X; G)$.

3 Step Graph Convolutional Network

3.1 Graph Residual Connection Network

Based on the residual connections proposed by He et al. (2016) and Zhang & Meng (2019), we build GRCNs using naive residual connections and graph residual connections. Figure 1 shows the structure of GRCNs.

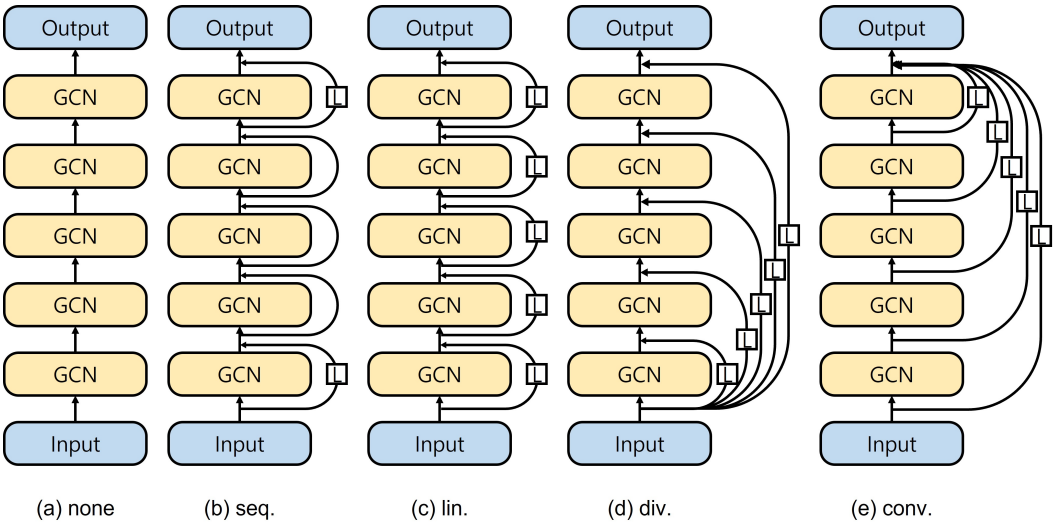

Figure 1: Graph residual connection networks (GRCNs). (a) is a vanilla GCN model; (b), (c), (d), and (e) are GRCNs with naive residual connections. (b) and (c) connect the input and the output of each layer. (d) propagates the initial input feature to every layer. (e) converges each layer to the output of the last layer. Note that $\boxed{\text{L}}$ denotes linear layers

Basically, we construct a GCN structure, introduced by Kipf & Welling (2017) (Figure 1 (a)). The initial node feature and normalized adjacency matrix are given as inputs of the model. It then transforms them to objective space, which means the labels of the nodes. In addition, we consider four types of GRCNs with naive residual connections and four types of GRCNs with graph residual connections. Figure 1 (b) employs residual connections to create a linkage between the input and output of each layer. The linear layers are employed in the first and last layers to correspond to the dimensional sizes. Figure 1 (c) has the same architecture as Figure 1 (b). Meanwhile, linear layers are applied at every residual connection. Figure 1 (d) propagates the initial node features, the input of the model, to every output of layers. Finally, in Figure 1 (e), every input fed into each layer is converged to the output of the last layer. Linear layers are also applied to every residual connection of Figure 1 (d) and (e).

There are four types of GRCNs with naive residual connections. Multiplying normalized adjacency matrix to every residual connections, we also construct GRCNs with graph residual connections and call them graph-seq., graph-lin., graph-div., and graph-conv. type. ReLU is used as an activation function, except the last layer, which used softmax to specify labels. Both the depth of the models and the dimensions in the hidden layers are hyperparameters.

3.2 STEP LEARNING

GCNs aggregate the information of the neighbor nodes within several hops, which are the same as the depth of the model, to compute the embedding of the target node. The set of neighbor nodes is called a receptive field. If the depth of the model increases, its receptive fields of different nodes will overlap a lot, blurring their embedding and causing over-smoothing problems. Therefore, different nodes should aggregate distinctive information to prevent this. To achieve it within similar receptive fields, different nodes should reference different amounts of information from neighbor nodes depending on their location. To implement it with graph-structured data, we attempt to extract such information from near nodes, with less information from far nodes. Step learning accomplishes it with additional ResBlocks and model tuning procedure. Figure 2 shows the entire process of step learning. At every step, we attach another ResBlock and perform model tuning, while the number of steps and layers of each ResBlock are the hyperparameters of step learning.

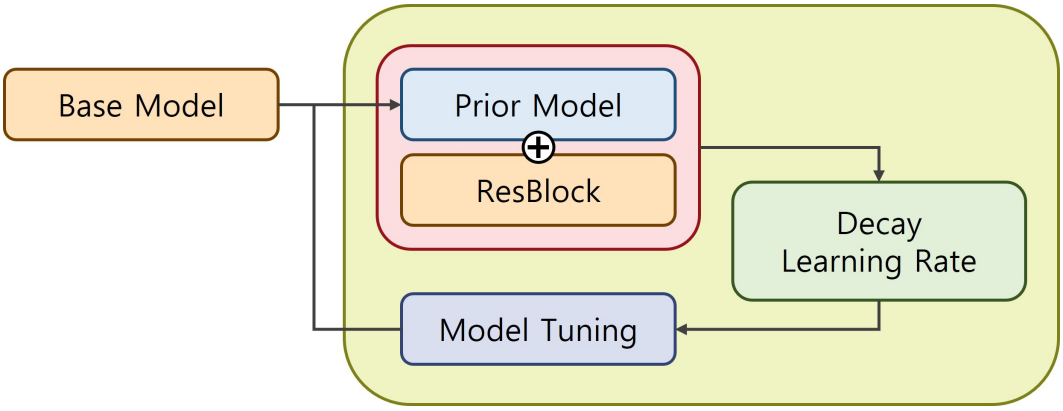

Figure 2: Step learning procedures

ResBlock. ResBlocks, which are organized by GCN layers and residual connections, are attached to a prior pretrained model. The input and the output dimensions of these blocks should be the same as the dimension of the input node feature, because the block is added at the front end of prior models. After attaching ResBlocks to prior models, the performance of prior model should be maintained. Thus, the initial state of ResBlocks should take a form of identity layers, delivering the input of layers to their output as it is. Each layer of ResBlocks $H(x)$ can be represented as $F(x) + R(x)$, the same as in Equation 2, and the initial form of the layer as $H(x) = x$ can be accomplished by setting initial value of $F(x) = 0$ and $R(x) = x$. By specifying all the parameters of GCN layers to 0, and those of residual connections to identity matrix, the initial state of ResBlocks could be set as the desired form. Figure 3 shows the four types of ResBlocks employed in this study.

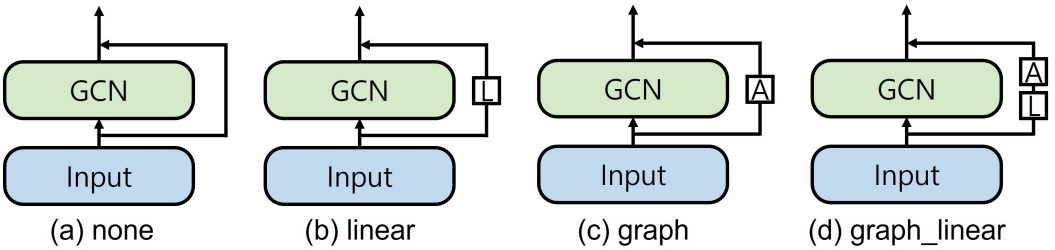

Figure 3: Types of ResBlocks. (a) is naive residual connection, (b) adjusts linear layer, while (c) and (d) multiply normalize adjacency matrix to (a) and (b) structure. L and A mean linear layer, whose initial parameters are set to identity matrix, and a normalized adjacency matrix, respectively

Model Tuning. The degree of learning should be distinctive for each layer to differentiate the amount of information propagated from neighbor nodes at different hops. It could be computed by tuning learning rates in training procedures, for example, training layers that extract information

from close neighbor nodes with a higher learning rate while layers that extract information from far neighbor nodes with a lower learning rate could differentiate the amount of information depending on the hops of each neighbor node. In each step, after appending another ResBlock, its training is conducted after decaying the learning rate by a certain level, making each layers trained with different learning rates.

4 EXPERIMENTS

We examined semi-supervised node classification experiments to demonstrate the performance of the StepGCN model. We employed three benchmark citation network datasets, namely, Cora, CiteSeer, and PubMed (Kipf & Welling, 2017; Sen et al., 2008). Nodes and edges are documents and citations, respectively. The features of nodes are bag-of-words representations of each document in these datasets. The task is to classify subjects of each document. Table 1 presents the statistics of the datasets.

Dataset	Nodes	Edges	Features	Classes	Label Rate
Cora	2,708	5,429	1,433	7	0.052
CiteSeer	3,327	4,732	3,703	6	0.036
Pubmed	19,717	44,338	500	3	0.003

Table 1: Dataset statistics

4.1 STEPGCN

Settings. Table 1 presents both classes and label rates to configure the training datasets. We employed 300 and 1,000 nodes in our validation and testing procedures. The employed GRCNs were evaluated with several depths, from 1 to 10, to decide which base models are fed into step learning. Step learning was then adjusted to demonstrate its capacity for extracting further information from data. We tested each model with every step (from the first to tenth steps).

All the numbers of the hidden nodes in every model were set to 16. In Figure 3, ResBlocks with single GCN layer were utilized for step learning. Adam optimizer (Kingma & Ba, 2015), with a learning rate of 0.01, was used. In each step, we dropped a learning rate by a tenth of its previous step. The dropout rate was fixed to 0.5, and L_2 regularization was set to 0.0005 on model parameters. All the reported results were mean accuracy of 100 independent runs with early stopping with a patience of 100 epochs. All the experiments were conducted on a NVIDIA A100 40 GB GPU and implemented in Python 3.6.

Results. Table 2 presents the performance of the experiments. GRCNs with graph residual connections generally outperform the matched GRCNs with naive residual connections, indicating that using graph residual connections can benefit from neighbor information. However, vanilla GCN surpasses most of the GRCNs, while the differences are within one percent even if GRCNs perform better than vanilla GCN. In addition, the depth of the best models of GRCNs does not exceed three. It corresponds to the finding of Kipf & Welling (2017), which reported that residual connections have no effects on examining over-smoothing problems.

Table 3 presents the results of step learning applications on the models based on the results of base models, as presented in Table 2. The graph-lin.-type StepGCN with graph-type ResBlocks achieves the highest accuracy in the Cora dataset (0.84280) in six steps. None-type models with graph-type ResBlocks accomplish best results in both CiteSeer (0.69234) and PubMed (0.81041) datasets in ten steps.

The performances of models with none-type ResBlock are slightly improved, when lower than five steps are employed. Among the four ResBlock types, the graph-type ResBlock achieves the greatest performance in our experiment. The majority of the performances are examined in the last step (10th). The results of graph-linear-type ResBlock show the contrasting results in using step learning. Thus, we found that using a linear layer is ineffective in StepGCN.

Model	Cora		CiteSeer		Pubmed	
	Accuracy	Depth	Accuracy	Depth	Accuracy	Depth
none type	0.82411	2-layer	0.68268	2-layer	0.77318	2-layer
seq. type	0.80146	3-layer	0.66765	2-layer	0.75099	3-layer
lin. type	0.79108	3-layer	0.66684	2-layer	0.74575	3-layer
div. type	0.80970	3-layer	0.67458	2-layer	0.76261	3-layer
conv. type	0.82091	2-layer	0.68437	2-layer	0.77667	2-layer
graph-seq. type	0.82304	2-layer	0.67245	2-layer	0.76604	2-layer
graph-lin. type	0.82110	2-layer	0.67528	2-layer	0.76668	2-layer
graph-div. type	0.82407	2-layer	0.68054	2-layer	0.77089	2-layer
graph-conv. type	0.82344	2-layer	0.68329	2-layer	0.77308	2-layer

Table 2: The performance of GRCNs

Base Model	ResBlock Type	Cora		CiteSeer		Pubmed	
		Accuracy	Step	Accuracy	Step	Accuracy	Step
none type	none	0.82714	4-step	0.68647	3-step	0.77623	3-step
	linear	0.82410	0-step	0.68301	0-step	0.77445	0-step
	graph	0.83862	4-step	**0.69234**	**10-step**	**0.81041**	**10-step**
	graph-linear	0.82346	0-step	0.68276	0-step	0.79428	10-step
seq. type	none	0.81541	5-step	0.67118	4-step	0.75905	4-step
	linear	0.80372	0-step	0.66672	0-step	0.74891	0-step
	graph	0.83172	9-step	0.68662	10-step	0.80473	10-step
	graph-linear	0.80810	10-step	0.66760	0-step	0.79434	10-step
lin. type	none	0.80643	5-step	0.67075	2-step	0.75178	1-step
	linear	0.79220	0-step	0.66695	0-step	0.74294	0-step
	graph	0.82962	7-step	0.68671	10-step	0.80436	10-step
	graph-linear	0.80622	10-step	0.66807	0-step	0.79449	10-step
div. type	none	0.82055	3-step	0.67854	3-step	0.77143	2-step
	linear	0.81117	0-step	0.67502	0-step	0.75994	0-step
	graph	0.83186	8-step	0.68796	10-step	0.80637	10-step
	graph-linear	0.81083	0-step	0.67505	0-step	0.79684	10-step
conv. type	none	0.82412	3-step	0.68835	3-step	0.77790	2-step
	linear	0.81804	0-step	0.68484	0-step	0.77568	0-step
	graph	0.83773	4-step	0.69186	9-step	0.80905	10-step
	graph-linear	0.82049	10-step	0.68507	0-step	0.79223	10-step
graph-seq. type	none	0.82838	3-step	0.67802	3-step	0.76977	3-step
	linear	0.82215	0-step	0.67448	0-step	0.76729	0-step
	graph	0.84251	6-step	0.68524	10-step	0.80724	10-step
	graph-linear	0.82282	0-step	0.67454	0-step	0.79442	10-step
graph-lin. type	none	0.82745	3-step	0.67767	5-step	0.77006	2-step
	linear	0.82337	0-step	0.67482	0-step	0.76771	0-step
	graph	**0.84280**	**6-step**	0.68491	10-step	0.80706	10-step
	graph-linear	0.82318	0-step	0.67462	0-step	0.79351	10-step
graph-div. type	none	0.82925	4-step	0.68351	5-step	0.77464	2-step
	linear	0.82310	0-step	0.67910	0-step	0.77099	0-step
	graph	0.84084	5-step	0.68871	10-step	0.81014	10-step
	graph-linear	0.82377	0-step	0.67891	0-step	0.79433	10-step
graph-conv. type	none	0.82758	4-step	0.68634	4-step	0.77563	3-step
	linear	0.82154	0-step	0.68359	0-step	0.77306	0-step
	graph	0.83805	2-step	0.69141	10-step	0.81027	10-step
	graph-linear	0.82327	0-step	0.68263	0-step	0.79310	10-step

Table 3: The performance of StepGCN

4.2 Comparison with Existing Models

Settings. Identical datasets and their settings with Section 4.1 were used to compare StepGCN with other GCN-based existing models. We employed JKNet (Xu et al., 2018) and GCNII (Chen et al., 2020b) as our baseline models. They were evaluated with several depths, from 1 to 10 for

JKNet, and from 1 to 100 for GCNII. Step learning was also applied to their best models found in the previous evaluation. The models were assessed until the tenth step, based on similar procedures in Section 4.1.

Except the depth of the models, we employed a set of hyperparameters used in Xu et al. (2018) and Chen et al. (2020b). In detail, Table 4 summarizes the employed hyperparameters for JKNet and GCNII. ResBlocks with single GCN layer were employed for step learning, and learning rate was decayed by a rate of 0.1 in each step. Adam optimizer (Kingma & Ba, 2015) was utilized in every training. The results of 100 independent runs with early stopping with a patience of 100 epochs were averaged to report the final results.

Model	Hyperparameter	Values
JKNet	hidden node	16
	learning rate	0.005
	dropout rate	0.5
	L_2 regularization	0.0005
	aggregation	max-pooling
GCNII	hidden node	64
	learning rate	0.01
	dropout rate	0.6
	convs L_2 regularization	0.01
	fcs L_2 regularization	0.0005
	α	0.1
	λ	0.5

Table 4: Hyperparameters of JKNet and GCNII

Results. Table 5 presents the results of baseline models. StepGCN refers to the best models of GCNs with residual connections in Table 2, as well as StepGCN* in Table 3. JKNet achieved their best performances with two layers, and much deeper depth was accomplished for GCNII. GCNII outperformed the other models before step learning in case of CiteSeer and PubMed datasets. The result also shows that step learning enhanced the accuracy of every model, regardless of the depth of the base model. From this, it can be concluded that step learning can be generalized to GCN-based models other than GRCNs. StepGCN* achieved the best accuracy in Cora (0.84280) and PubMed (0.81041) datasets. It proves that extracting information from nodes multi-hops away is favorable, and an effective learning procedure can be more powerful than complicated model architecture.

Model	Cora			CiteSeer			Pubmed		
	Accuracy	ResBlock	Depth & Step	Accuracy	ResBlock	Depth & Step	Accuracy	ResBlock	Depth & Step
JKNet	0.80807	-	2-layer	0.68179	-	2-layer	0.77572	-	2-layer
	0.81175	graph	3-step	0.69144	graph	3-step	0.80392	graph	9-step
GCNII	0.79903	-	26-layer	0.70274	-	37-layer	0.79292	-	96-layer
	0.82085	linear	1-step	0.70758	none	1-step	0.80513	graph	6-step
StepGCN	0.82411	-	2-layer	0.68437	-	2-layer	0.77667	-	2-layer
	0.83862	graph	4-step	0.69186	graph	9-step	0.80905	graph	10-step
StepGCN*	0.82110	-	2-layer	0.68268	-	2-layer	0.77318	-	2-layer
	0.84280	graph	6-step	0.69234	graph	10-step	0.81041	graph	10-step

Table 5: The performance of baseline

5 Concluding Remarks

We propose StepGCN with various types of residual connections as base model and step learning, extracting information from neighbor nodes, considering that their hops from target node are adjusted. The experiments demonstrate that StepGCN is effective for learning representations from graph-structured data, without suffering from an over-smoothing. Moreover, additional experiments show that step learning can be generalized to other GCN-based models. Interesting directions for future work include applying step learning to other models and expanding its effectiveness to inductive tasks.

ETHICS STATEMENT

This study does not have any ethical concerns. We do not address any human subjects, potential harmful insights, methodologies and potential conflicts of interest. Moreover, this study does not have any privacy, security, illegal, and research integrity concerns.

REPRODUCIBILITY STATEMENT

To make sure the reporducibility and readability of our experimental results, all code and instructions are available at `https://anonymous.4open.science/r/StepGCN-7709/`.

REFERENCES

Tian Bian, Xi Xiao, Tingyang Xu, Peilin Zhao, Wenbing Huang, Yu Rong, and Junzhou Huang. Rumor detection on social media with bi-directional graph convolutional networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pp. 549–556, 2020.

Deli Chen, Yankai Lin, Wei Li, Peng Li, Jie Zhou, and Xu Sun. Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 3438–3445, 2020a.

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

Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. Simple and deep graph convolutional networks. In *International Conference on Machine Learning*, pp. 1725–1735. PMLR, 2020b.

Sihao Ding, Fuli Feng, Xiangnan He, Yong Liao, Jun Shi, and Yongdong Zhang. Causal incremental graph convolution for recommender system retraining. *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (Poster)*, 2015.

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

Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

Guohao Li, Matthias Muller, Ali Thabet, and Bernard Ghanem. Deepgcns: Can gcns go as deep as cnns? In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 9267–9276, 2019.

Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *Thirty-Second AAAI conference on artificial intelligence*, 2018.

Yaoman Li and Irwin King. Autograph: Automated graph neural network. In *International Conference on Neural Information Processing*, pp. 189–201. Springer, 2020.

Zheng Li, Xiaocong Du, and Yu Cao. Gar: Graph assisted reasoning for object detection. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pp. 1295–1304, 2020.

Meng Liu, Hongyang Gao, and Shuiwang Ji. Towards deeper graph neural networks. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 338–348, 2020a.

Zheng Liu, Zidong Jiang, Wei Feng, and Hui Feng. Od-gcn: object detection boosted by knowledge gcn. In *2020 IEEE International Conference on Multimedia & Expo Workshops (ICMEW)*, pp. 1–6. IEEE, 2020b.

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.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

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

Li Wang, Chenfei Wang, Xinyu Zhang, Tianwei Lan, and Jun Li. S-at gcn: Spatial-attention graph convolution network based feature enhancement for 3d object detection. *arXiv preprint arXiv:2103.08439*, 2021.

Yongji Wu, Defu Lian, Shuowei Jin, and Enhong Chen. Graph convolutional networks on user mobility heterogeneous graphs for social relationship inference. In *International Joint Conferences on Artificial Intelligence*, pp. 3898–3904, 2019.

Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. In *International conference on machine learning*, pp. 5453–5462. PMLR, 2018.

Carl Yang, Aditya Pal, Andrew Zhai, Nikil Pancha, Jiawei Han, Charles Rosenberg, and Jure Leskovec. Multisage: Empowering gcn with contextualized multi-embeddings on web-scale multipartite networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 2434–2443, 2020.

Jiani Zhang, Xingjian Shi, Shenglin Zhao, and Irwin King. Star-gcn: Stacked and reconstructed graph convolutional networks for recommender systems. In *International Joint Conference on Artificial Intelligence*, pp. 4264, 2019.

Jiawei Zhang and Lin Meng. Gresnet: Graph residual network for reviving deep gnns from suspended animation. *International Conference on Learning Representations*, 2019.

Lingxiao Zhao and Leman Akoglu. Pairnorm: Tackling oversmoothing in gnns. In *International Conference on Learning Representations*, 2019.