Training Graph Neural Networks by Graphon Estimation

Ziqing Hu * zhu4@nd.edu
Yihao Fang * yfang5@nd.edu
Lizhen Lin * lizhen.lin@nd.edu

*Department of the Applied and Computational Mathematics and Statistics
University of Notre Dame
Notre Dame, IN, USA

2021 IEEE International Conference on Big Data (Big Data) | 978-1-6654-3902-2/21/$31.00 ©2021 IEEE | DOI: 10.1109/BigData52589.2021.9671996

Abstract

In this work, we propose to train a graph neural network via resampling from a graphon estimate obtained from the underlying network data. More specifically, the graphon or the link probability matrix of the underlying network is first obtained from which a new network will be resampled and used during the training process at each layer. Due to the uncertainty induced from the resampling, it helps mitigate the well-known issue of over-smoothing in a graph neural network (GNN) model. Our framework is general, computationally efficient, and conceptually simple. Another appealing feature of our method is that it requires minimal additional tuning during the training process. Extensive numerical results show that our approach is competitive with and in many cases outperform the other over-smoothing reducing GNN training methods.

Index Terms

Graph neural network, Graphon estimation, Oversmoothing, Resampling.

I. Introduction

This paper considers an approach for mitigating the well-known problem of over-fitting and over-smoothing in the training of Graph Neural Networks (GNN)s and lies at the intersection of graphon estimation and neural network models. GNNs, initially proposed to capture graph representations in neural networks [**scarselli2008graph**], have witnessed an upsurge for semi-supervised learning in a variety of tasks including node classification, link predictions, and many others. The goal of each GNN layer is to transform features while considering the graph structure

by aggregating information from connected or neighboring nodes. When there is only one graph, the goal of node classification is to predict node labels in a graph while only a portion of node labels are available (even though the model might have access to the features of all nodes). Inspired by the advance of convolutional neural networks [**lecun1998gradient**] in computer vision [**krizhevsky2012imagenet**], Graph Convolutional Network (GCN) [**kipf2016semi**] employs the spectra of graph Laplacian for filtering signals and the kernel can be approximated using Chebyshev polynomials or functions [**zhou2018graph**, **wu2020comprehensive**]. GCN has become a standard and popular tool in the emerging field of geometric deep learning [**bronstein2017geometric**]. However, the issue of over-fitting arises when an overparametrized model such as the deep neural network, is applied to a distribution with limited training data, where the learned fits the training data well but generalizes poorly to the testing data. This can be illustrated briefly by fitting a deep GNN (more than 4 layers) to small a graph data (e.g., the Cora dataset). On the other hand, the issue of over-smoothing introduced by [**li2018deeper**] towards the other extreme, bringing difficulties to deep GNN training. Further explained by [**wu2020comprehensive**], graph convolutions mix representations of adjacent nodes and result in all nodes' representations converging to a stationary subspace or point [**oono2019asymptotic**]. This phenomenon is called *over-smoothing* of node features [**rong2019dropedge**]. By way of illustration, GCN models with more than 8 layers are observed to converge poorly in our experiments.

To alleviate those two issues, inspired by [**zhang2017estimating**], we propose a new GNN structure with resampling the adjacency matrix in the feed forward propagation via graphon estimation. Graphon, a function that determines the matrix of edge probabilities, plays an important role in graph theory and statistics [**optimal-graphon**, **zhao2019change**]. The estimation of probabilities of network edges from the observed adjacent matrix, known as "graphon estimation", has a wide range of applications to predicting missing links and network denoising [**Chatterjee˙2015**, **10.1214/15-AOS1370**]. In our framework, we assume the observed adjacency matrix A is generated from an underlying probability matrix P so that for $i \leq j$, $A'_{ij}s$ are independent Bernoulli(P_{ij}) trails where P_{ij} are edge probabilities. Consequently, we resample the adjacency matrix A from the estimated distribution P in the feed forward propagation for each training epoch. There are several benefits in applying the resampling strategy for training GNN. First, resampling the adjacency matrix is one way for data augmentation to relieve the over-fitting. We obtain more graph samples from the underlying distribution under this method. Second, re-

sampling strategy can be considered as noise addition to the deterministic GNN and which avoids our nodes' representations converging to the stationary subspace [**oono2019asymptotic**], hence solving the over-smoothing phenomena. Finally, since we consider the underlying distribution of the graph, our method is able to achieve a stable result under noisy graphs.

Our work is organized as follows. Section II reviews some related work. Section III provides an overview of some background information such as GNN and GCN. The proposed algorithm is described in section IV and a series of experiments are performed in section V to evaluate our proposed method's efficiency and sensitivity to hyper-parameters. Finally, the work is concluded in section VI.

II. RELATED WORKS

A. GRAPH NEURAL NETWORK

Most graph neural networks, as mentioned above, are treating the related graph as ground-truth deterministic structure between nodes, but often the graph itself may be subjected to random perturbation or theoretical assumptions that might lead to unreliable results given the uncertain graph. [**zhang2019bayesian**] firstly propose a Bayesian version GCN (BGCN) to incorporate the potential uncertainty presented in the graph. Similarly, [**elinas2019variational**] extend the BGCN to include the node features and adopt the variational inference method to estimate the posterior distribution which achieve comparable result under adversarial attack setting. However, due to the computation complexity, it's not easy to apply the model on large datasets. Based on bilevel programming, [**franceschi2019learning**] proposes a method for jointly learning the graph structure and network parameter via constrained optimization. From over-smoothing alleviation perspective, [**hasanzadeh2020bayesian**] propose Graph DropConnect (GDC) method to alleviate the over-smoothing issue in GCN by resampling the graph for each node feature and show that DropOut [**srivastava2014dropout**], DropEdge [**rong2019dropedge**] and Node Sampling [**chen2018fastgcn**] are special cases of GDC with respect to different settings. However, there is no theoretical guarantee that GDC can reduce the over-smoothing issue. Finally, similar to our work, [**zhao2020data**] propose a two-step procedure for data augmentation in graph neural network. They firstly use graph auto-encoder (GAE) [**kipf2016variational**] to estimate the edge probability which is used for resampling in later procedure. Then, combining the resampled graph with original graph, they applied another graph neural network to learning the embedding

of nodes. However, their emphasis is very different from ours as we focus on reducing over-smoothing issue in a deep graph neural network.

B. GRAPHON ESTIMATION

Graphon estimation is an important component of our proposed procedure for training the GNNs. A prominent estimator of the graphon is the so-called USVT (Universal Singular Value Thresholding) estimator [**Chatterjee˙2015**]. USVT is a general procedure for estimating the entries of a large structured matrix, given a noisy realization of the matrix. This includes estimating the link probability matrices which is our case of interest. The key idea behind USVT is to threshold the singular values of the observed matrix at an universal threshold which essentially approximates the rank of the population matrix, and then compute an approximation of the population matrix using the top singular values and vectors. A recent work by [**zhang2017estimating**] proposes a statistically consistent and computationally efficient method for estimating the link probability matrix by neighborhood smoothing. More specifically, given an adjacent matrix A, the link probability P_{ij} between node i and j is estimated by

$$\hat{P}_{ij} = \frac{1}{2}\left(\frac{\sum_{i' \in \mathcal{N}(v_i)} A_{i'j}}{|\mathcal{N}(v_i)|} + \frac{\sum_{j' \in \mathcal{N}(v_j)} A_{ij'}}{|\mathcal{N}(v_j)|}\right), \tag{1}$$

where $\mathcal{N}(v_i)$ is a certain set of neighboring nodes of node v_i (which consists of the nodes that have similar connection patterns as node v_i). Rather than simply choosing connected node as neighbours, the neighbour is selected by the following criteria $\mathcal{N}(v_i) = \left\{i' \neq i : \tilde{d}\left(i, i'\right) \leq q_i(h)\right\}$ where distance $\tilde{d}^2\left(i, i'\right)$ is defined as $\tilde{d}^2\left(i, i'\right) = \max_{k \neq i, i'} \left|\langle A_i - A_{i'}, A_{k\cdot}\rangle\right| / n$ and $q_i(h)$ is the h-th sample quantile of the set $\left\{\tilde{d}\left(i, i'\right) : i' \neq i\right\}$.

Typically for large networks USVT is more scalable than the neighborhood-smoothing approach. There are several other methods for graphon estimations, e.g., by fitting a stochastic blockmodel [**wolfe2013nonparametric**]. These methods can also be used in our proposed GNN training algorithm.

III. NOTATION AND BACKGROUND

A. Notation

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ represent the input graph with node set $\mathcal{V}$ of size N and edge set $\mathcal{E}$ where $v_i \in \mathcal{V}$ and $(v_i, v_j) \in \mathcal{E}$. $N(v_i)$ denotes all the neighbours connected to node v_i. We denote the $\boldsymbol{X} = \{\boldsymbol{x}_1, \cdots, \boldsymbol{x}_N\} \in \mathbb{R}^{N \times f}$ as the node feature matrix and $\mathbf{A} \in \{0, 1\}^{N \times N}$ as the adjacency

matrix. Let D be the diagonal matrix with node degrees $d_i = \sum_{i=1}^{N(v_i)} A_{i,j}$ as its entries. I is the identity matrix.

B. Graph Neural Networks

The Graph Neural Networks (GNN) can be seen as an extension of NN that learns the embedding of the data in graph domains [**scarselli2008graph**]. The basic idea can be written by a local transition function as, for each node $v_i, \ldots, v_n$,

$$\boldsymbol{x}_{v_i}^{l} = \mathbf{f}_l(\boldsymbol{x}_{v_i}^{l-1}, \underline{\boldsymbol{x}}_{N(v_i)}^{l-1}; W_l) \tag{2}$$

where $\underline{\boldsymbol{x}}_{N(v_i)}^{l-1}$ represents all the neighbouring information of node v_i at the lth layer. The $\boldsymbol{x}_{v_i}^{l}$ and W_l are the embedding of node v_i and the model parameters at l-th layer, respectively.

The Graph Convolutional Network (GCN) developed in [**kipf2016semi**] is one of the variants of GNN with the message passing mechanism as the graph signal filter in graph Fourier space, which can be written in matrix form as:

$$\boldsymbol{X}^{l} = \sigma\left(\tilde{A}\boldsymbol{X}^{l-1}W^{l}\right), \tag{3}$$

where σ is a element-wise nonlinear activation function such as $\text{ReLU}(x) = \max(x, 0)$, W^l is a $f^l \times f^{l-1}$ parameter matrix that needs to be estimated. $\tilde{A}$ denotes the normalized adjacency matrix defined by $\tilde{A} = (D + I)^{-1/2}(A + I)(D + I)^{-1/2}$.

IV. Method

In this section, we introduce the methodology of graphon estimation in training of generic GNNs. Moreover, we also propose and implement its layer-wise variant where we resample the adjacent matrix A from the estimated distribution P for each layer in the model. We also illustrate how our graphon estimation technique can alleviate over-smoothing and over-fitting issues.

A. Resampling strategy

For the given graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with adjacency matrix A, we apply the neighbouring smoothing method (NBS) [**zhang2017estimating**] that was described in Section II to estimate the underlying link probability matrix P, denoted as $\hat{P}$ (see equation (1)). Other graphon estimation methods can also be used. At each training epoch, we resample a new adjacency matrix $\hat{A}$ from the estimated link probability matrix $\hat{P}$ element-wisely following Bernoulli distribution:

$$\hat{A}_{ij} \sim Bern(\hat{P}_{ij}), \quad 1 \leq i, j \leq n. \tag{4}$$

We replace A with $\hat{A}$ in equation (4) during training. The original A is utilized for validation and test.

B. Layer-wise variant

Besides resampling the adjacency matrix $\hat{A}$ for the whole propagation, we can resample $\hat{A}^l$ independently from Equation 4 for each l-th layer. In particular, different l-layer could have different matrix $\hat{A}^l$ and additional randomness and augmentation of the original data could be brought to our training process. We compare its performance with the vanilla resampling strategy in Section V.

C. Alleviating over-smoothing and over-fitting

Over-fitting occurs when an overparametrized model is utilized to fit a distribution with limited training data. To prevent this issue, we first estimate the underlying graphon of the input graph. Our resample strategy works as a data augmentation technique by generating different realizations of the input data from the underlying distribution. On the other hand, the over-smoothing phenomenon indicates that the node features would converge to the fixed point as the network depth increases [**li2018deeper**]. Furthermore, [**oono2019asymptotic**] has extended the original explanation to a more general framework by considering the non-linear activation function in the GCN propagation. Instead of converging to the fixed point, the node features will converge to a subspace related to the eigenspace of the graph adjacency matrix. The key point of the theory illustrates that when the same adjacency matrix is utilized for all layers, the whole dynamic system will go closer to the corresponding eigenspace as the number of layers increases under specific assumptions. To avoid the phenomena, our proposed method draws random adjacency matrices from the underlying graphon in training, which helps the dynamic system escape from the subspace. Different from other random sampling methods like DropEdge and DropNode, the graphon estimation method is able to detect the underlying graphon and provide a robust estimator with statistical bounds. Consequently, our proposed method enables us to train deep GNNs more effectively, notably when the input graphs are noisy.

V. EXPERIMENTS

In this section, we evaluate the proposed resampling algorithm on several datasets through different network architectures. A summary of datasets and their splitting settings are provided.

All the experiments are conducted by Pytorch [**paszke2019pytorch**] and Pytorch Geometric [**fey2019fast**].

A. Datasets

The summary statistics of the data are shown in Table I. We follow three different data-splitting settings for semi-supervised tasks on these datasets. The first setting comes from [**yang2016revisiting**], named 'public', in which 20 samples for each cluster are randomly drawn for training, 500 for validation, and 1000 for the test. For the next split in [**chen2018fastgcn**], named 'complete', 1708 samples are selected for training, 500 for validation and 500 for test. The last setting comes from [**levie2018cayleynets**], named 'full', which chooses all of the samples for training except for 500 nodes for the validation and 500 nodes for the test. For graphon estimation, we use the whole dataset and pre-compute it before running the network model.

TABLE I

CITATION NETWORK DATASETS SUMMARY

Dataset	Nodes	Edges	Classes	Features
Citeseer	3,327	4732	6	3,703
Cora	2,708	5,429	7	1,433
Pubmed	19,717	44,338	3	500

B. Architectures

We employ 3 different widely used GNN architectures in our experiment: GCN[**kipf2016semi**], GraphSAGE[**hamilton2017inductive**] and JK-NET[**xu2018representation**] with layers ranging from 2 to 16. Note that, for JK-NET, the number of layers doesn't include the concatenation and output layer. For the hidden layer dimension, we follow the same 64-dimension setting with [**kipf2016semi**]. We choose ReLU function as our activation function between each layer and the cross entropy as our loss function.

C. Optimization

We initialize the weight parameters through Xavier uniform initialization. All of the data are row-wise normalized accordingly [**glorot2010understanding**]. The model is trained for 1000 epochs with a learning rate start from 0.001 and decreased at epoch, 300 and 600 with decay rate 0.5. The Adam optimizer is used without any penalty term.

D. Results

Due to limited space, we only attach the result of 'Public' splitting setting, which is given in Table II. We let the 'Resampling' represents our original algorithm while the 'Layerwise' represents the layer-wise variant of our method. We pick the best result of 'Dropedge' in each setting where the dropping rate ranging from 0.2 to 0.8. The reported value are the average and stand deviation over 10 runs in Table II. Also, we apply the early stopping to keep track of the validation loss, if the loss stops decreasing for several epochs. As shown by the numerical results in the table, our Resampling method or its layerwise variant performs the best in most of the settings for all three datasets and three different GNN architectures considered. Precisely, we consider a 8-layer GCN with/without Resampling (Layerwise) on the Citation dateset. In term of the loss evolution among different methods, our Resampling method or the layerwise variant is able to alleviate both overfitting and oversmoothing issues as shown in Figure 1. Similar patterns are observed in other splitting settings. The results demonstrate the effectiveness of our proposed methods in comparing with other state-of-art methods. In comparing with other methods, like Dropedge, which requires multiple comparisons to determine the appropriate Dropedge rate, our approach requires minimal additional tuning. Once the estimation of graphon is completed, we can re-use it without any further modification.

VI. Conclusion

In this work, we introduced a novel and efficient graphon estimation technique for training deep Graph Neural Networks. Our proposed method augments the input graph to alleviate over-fitting and over-smoothing by drawing random adjacency matrix from the estimated graphon. Considerable experiments on Cora, Citeseer and Pubmed on different splits have agreed that our graphon estimation method is able to promote the performance of several popular GNNs, like GCN, JKNet and GraphSAGE, in particular for the network with deep layers. To the best of our knowledge, this is the first work utilizing graphon estimation on Graph Neural Networks. We also aim to exploit the theoretical analysis and large scale graph training of GNNs, with a variety of graphon estimation methods in the future work.

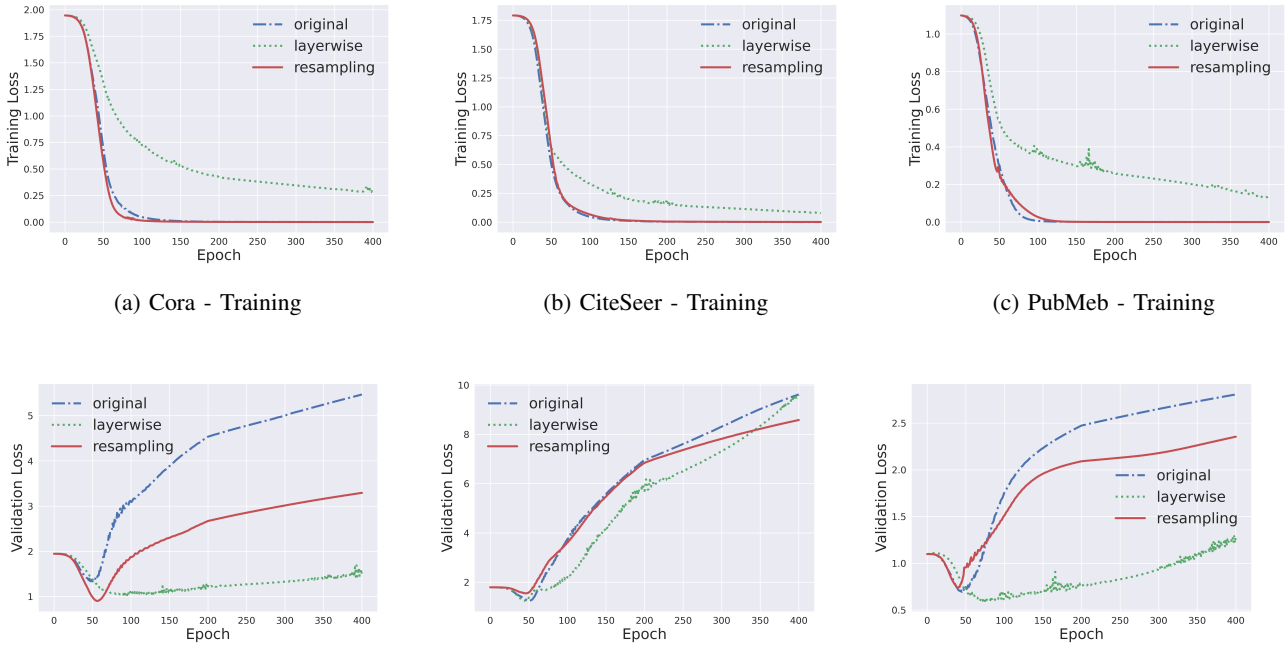
Acknowledgment

This research is partially supported by NSF grants DMS Career 1654579, DMS 1854779 and DMS 2113642.

TABLE II

THE TEST ACCURACY OF DIFFERENT METHODS IN 'PUBLIC' SETTING

			Cora		
GCN	Original	Resampling	Layerwise	Dropedge	Dropout 0.2
2-Layers	79.74 ± 0.006	$\mathbf{80.30 \pm 0.004}$	76.53 ± 0.015	80.16 ± 0.005	80.56 ± 0.006
4-Layers	76.91 ± 0.020	77.15 ± 0.016	74.46 ± 0.013	78.93 ± 0.014	$\mathbf{78.95 \pm 0.012}$
8-Layers	68.76 ± 0.060	70.93 ± 0.040	69.16 ± 0.020	71.38 ± 0.031	$\mathbf{71.52 \pm 0.028}$
16-Layers	59.36 ± 0.039	59.43 ± 0.025	$\mathbf{65.31 \pm 0.037}$	60.50 ± 0.027	64.48 ± 0.026
GraphSage	Original	Resampling	Layerwise	Dropedge	Dropout 0.2
2-Layers	76.80 ± 0.007	$\mathbf{78.16 \pm 0.006}$	78.08 ± 0.005	52.06 ± 0.034	77.05 ± 0.010
4-Layers	78.41 ± 0.012	78.51 ± 0.013	77.01 ± 0.014	43.21 ± 0.067	$\mathbf{79.34 \pm 0.010}$
8-Layers	75.79 ± 0.014	74.45 ± 0.021	74.58 ± 0.025	52.62 ± 0.075	$\mathbf{76.90 \pm 0.012}$
16-Layers	$\mathbf{72.61 \pm 0.020}$	71.63 ± 0.016	69.84 ± 0.028	34.21 ± 0.095	67.18 ± 0.022
JK-Net	Original	Resampling	Layerwise	Dropedge	Dropout 0.2
2-Layers	47.83 ± 0.032	65.32 ± 0.007	$\mathbf{65.89 \pm 0.023}$	52.52 ± 0.041	48.34 ± 0.021
4-Layers	49.58 ± 0.037	$\mathbf{63.56 \pm 0.019}$	63.49 ± 0.016	44.66 ± 0.060	51.37 ± 0.025
8-Layers	49.77 ± 0.017	$\mathbf{62.89 \pm 0.025}$	60.88 ± 0.016	32.31 ± 0.077	55.94 ± 0.029
16-Layers	56.72 ± 0.027	$\mathbf{59.02 \pm 0.046}$	52.89 ± 0.027	27.46 ± 0.075	57.46 ± 0.025
			CiteSeer		
GCN	Original	Resampling	Layerwise	Dropedge	Dropout 0.2
2-Layers	68.12 ± 0.008	68.40 ± 0.006	67.66 ± 0.010	$\mathbf{68.69 \pm 0.005}$	68.62 ± 0.009
4-Layers	$\mathbf{65.61 \pm 0.024}$	64.92 ± 0.018	65.26 ± 0.016	65.48 ± 0.015	65.46 ± 0.016
8-Layers	55.29 ± 0.029	52.36 ± 0.041	$\mathbf{60.63 \pm 0.030}$	55.89 ± 0.037	55.83 ± 0.043
16-Layers	48.06 ± 0.027	47.46 ± 0.025	$\mathbf{51.39 \pm 0.015}$	48.39 ± 0.030	51.41 ± 0.024
GraphSage	Original	Resampling	Layerwise	Dropedge	Dropout 0.2
2-Layers	65.21 ± 0.013	67.74 ± 0.012	$\mathbf{68.20 \pm 0.012}$	50.64 ± 0.037	66.19 ± 0.007
4-Layers	64.80 ± 0.017	67.01 ± 0.017	$\mathbf{67.28 \pm 0.012}$	38.25 ± 0.085	64.96 ± 0.017
8-Layers	57.55 ± 0.048	62.06 ± 0.026	$\mathbf{63.29 \pm 0.023}$	44.23 ± 0.037	58.53 ± 0.043
16-Layers	50.08 ± 0.053	56.56 ± 0.032	$\mathbf{56.94 \pm 0.028}$	30.59 ± 0.043	44.30 ± 0.031
JK-Net	Original	Resampling	Layerwise	Dropedge	Dropout 0.2
2-Layers	38.94 ± 0.019	51.71 ± 0.018	$\mathbf{52.83 \pm 0.026}$	45.40 ± 0.032	40.04 ± 0.023
4-Layers	39.83 ± 0.020	50.86 ± 0.022	$\mathbf{51.16 \pm 0.023}$	38.51 ± 0.049	41.74 ± 0.021
8-Layers	37.76 ± 0.027	$\mathbf{48.91 \pm 0.023}$	48.26 ± 0.015	27.53 ± 0.046	41.59 ± 0.029
16-Layers	43.01 ± 0.027	44.04 ± 0.032	43.02 ± 0.027	25.55 ± 0.035	$\mathbf{44.35 \pm 0.015}$
			PubMed		
GCN	Original	Resampling	Layerwise	Dropedge	Dropout 0.2
2-Layers	76.37 ± 0.003	76.45 ± 0.003	74.88 ± 0.019	76.59 ± 0.003	$\mathbf{76.69 \pm 0.002}$
4-Layers	76.75 ± 0.004	76.76 ± 0.006	75.75 ± 0.011	$\mathbf{77.03 \pm 0.007}$	76.94 ± 0.006
8-Layers	74.26 ± 0.020	73.78 ± 0.027	$\mathbf{76.41 \pm 0.011}$	75.11 ± 0.033	75.28 ± 0.026
16-Layers	71.86 ± 0.022	$\mathbf{73.70 \pm 0.021}$	72.34 ± 0.021	73.15 ± 0.017	73.07 ± 0.021
GraphSage	Original	Resampling	Layerwise	Dropedge	Dropout 0.2
2-Layers	74.93 ± 0.006	$\mathbf{77.34 \pm 0.004}$	76.62 ± 0.007	69.21 ± 0.023	75.24 ± 0.006
4-Layers	74.60 ± 0.009	$\mathbf{76.38 \pm 0.010}$	74.67 ± 0.006	64.24 ± 0.046	75.56 ± 0.010
8-Layers	74.72 ± 0.013	74.17 ± 0.015	74.06 ± 0.015	69.36 ± 0.029	$\mathbf{76.24 \pm 0.012}$
16-Layers	73.20 ± 0.019	74.11 ± 0.018	73.64 ± 0.011	56.57 ± 0.061	$\mathbf{75.59 \pm 0.017}$
JK-Net	Original	Resampling	Layerwise	Dropedge	Dropout 0.2
2-Layers	58.43 ± 0.020	$\mathbf{70.88 \pm 0.010}$	69.92 ± 0.018	60.35 ± 0.037	59.61 ± 0.031
4-Layers	59.64 ± 0.043	$\mathbf{69.24 \pm 0.013}$	68.94 ± 0.020	55.36 ± 0.070	59.26 ± 0.020
8-Layers	57.49 ± 0.037	$\mathbf{69.09 \pm 0.017}$	65.10 ± 0.024	46.28 ± 0.057	58.59 ± 0.029
16-Layers	57.43 ± 0.034	$\mathbf{60.98 \pm 0.042}$	54.73 ± 0.042	46.55 ± 0.080	58.30 ± 0.025



(a) Cora - Training (b) CiteSeer - Training (c) PubMeb - Training

(d) Cora - Validation (e) CiteSeer - Validation (f) PubMed - Validation

Fig. 1. The training and validation loss of GCNs on the public split of Citation datasets. We implement the original 8-layer GCN (in blue), with resampling (in red) and with layerwise variant (in green). The original 8-layer GCN comes up with the overfitting issue in several epochs with low training loss but high validation loss on all datasets. Furthermore, the validation of the original 8-layer GCN diverges significantly in all cases due to over-smoothing issue. In contrast, our Resampling and layerwise variant (in green), alleviates both overfitting and oversmoothing issues and achieves smaller validation errors in notably for Cora and PubMed Fig. 1d and 1f. We utilize early stop technique in training around 100 epochs to achieve the best performance in Table II. Here we show the loss within 400 epochs for a complete comparison.