

Effective Training Strategies for Deep Graph Neural Networks

Kuangqi Zhou^{1*}, Yanfei Dong^{1,2*}, Wee Sun Lee¹, Bryan Hooi¹, Huan Xu², Jiashi Feng¹,

¹National University of Singapore, ²Alibaba Group

{kzhou, dyanfei}@u.nus.edu, {leews, bhooi}@comp.nus.edu.sg,

{yanfei.dong, huan.xu}@alibaba-inc.com, elefjia@nus.edu.sg

Abstract

Graph Neural Networks (GNNs) tend to suffer performance degradation as model depth increases, which is usually attributed in previous works to the *oversmoothing* problem. However, we find that although oversmoothing is a contributing factor, the main reasons for this phenomenon are *training difficulty* and *overfitting*, which we study by experimentally investigating Graph Convolutional Networks (GCNs), a representative GNN architecture. We find that training difficulty is caused by gradient vanishing and can be solved by adding residual connections. More importantly, overfitting is the major obstacle for deep GCNs and cannot be effectively solved by existing regularization techniques. Deep GCNs also suffer *training instability*, which slows down the training process. To address overfitting and training instability, we propose Node Normalization (NodeNorm), which normalizes each node using its own statistics in model training. The proposed NodeNorm regularizes deep GCNs by discouraging feature-wise correlation of hidden embeddings and increasing model smoothness with respect to input node features, and thus effectively reduces overfitting. Additionally, it stabilizes the training process and hence speeds up the training. Extensive experiments demonstrate that our NodeNorm method generalizes well to other GNN architectures, enabling deep GNNs to compete with and even outperform shallow ones. Code is publicly available¹.

1 Introduction

Graph Neural Network (GNN) architectures [16, 11, 3, 6] have been proposed and widely adopted to learn from graph-structured data. Many current GNNs [16, 11, 30, 33] are built under the framework of neural message passing [8]. These models perform neighbor propagation and transformation operations in each layer [12, 31, 36], and thus the number of layers corresponds to the range of interactions among nodes. However, they are generally shallow, consisting of only 2 or 3 layers, which means they can only utilize short range interactions between nodes. Though engaging longer range interactions is beneficial in many cases [17, 35, 7], it is non-trivial since the performance of GNNs has been found to decline drastically as they get deeper [18, 4, 20, 21].

In this work, we take Graph Convolutional Networks [16] (GCNs) as an example to experimentally analyze the obstacles of training deep GNNs. Previously such performance degradation is mainly attributed to the over-smoothing problem caused by stacked propagation operations [18, 4, 20]. By disentangling and investigating the effects of propagation and transformation operations, we find that although over-smoothing hurts the performance to some extent, stacked transformation operations contribute more to the performance drop. To further study the problems caused by these operations, we conduct more experiments to analyze the training process of GCNs. We observe that *training*

*Equal Contribution

¹ <https://github.com/miafei/NodeNorm>

difficulty caused by gradient vanishing poses great impact upon the performance, which can be easily solved by adding residual connections [14]. Furthermore, we observe that deep GCNs with residual connections (GCN-res) still fail to eliminate the performance drop problem. We find this is because they tend to *overfit* to the training set and have difficulty generalizing to the test set — the loss and accuracy gaps between the training and validation sets become increasingly large as a GCN-res model becomes deeper. We also find that this problem cannot be addressed by existing regularization techniques such as Dropout [26]. Additionally, we observe that they require many more epochs to converge than the shallow ones due to *training instability*.

The above problems, i.e. overfitting and training instability, are conventionally tackled by normalization techniques [15, 22], as they regularize deep neural networks implicitly and stabilize the training process. Among them, Batch Normalization (BatchNorm) [15] is the most widely applied [14, 23]. Although it helps stabilize training and improve performance for deep GCN-res models, we observe that deep GCN-res with BatchNorm still cannot compete with shallow GCN-res and often performs badly on nodes with large variance in their features. This is possibly because BatchNorm normalizes all nodes in a feature-wise manner and does not adjust node-wise variance.

Based on these observations, we propose Node Normalization (NodeNorm) that normalizes each node using its own statistics. This ensures that all nodes have unit variance, and thus enables deep GCNs to achieve good overall performance across all nodes. Additionally, it helps speed up the training by stabilizing the training process and thus allowing a larger learning rate to be used. Furthermore, we find empirically that NodeNorm has better regularization effects and thus reduces overfitting more effectively than BatchNorm. Firstly, deep models trained with NodeNorm produce less correlated hidden features, which implies less overfitting according to [5]. Secondly, we define an empirical Graph Lipschitz Constant (GL Const) to assess the smoothness of a GNN w.r.t. its input node features, and find that deep models become smoother when equipped with NodeNorm. This suggests that NodeNorm acts as a better implicit regularizer in the training process of deep models.

We apply our proposed NodeNorm to various GNN backbones, including GCN [16], GAT [28] and GraphSage [11]. We conduct extensive experiments on popular semi-supervised node classification benchmark datasets [24, 25], and find that our method can successfully address the degradation problem. To further show the power of NodeNorm, we also identify a scenario where longer range interactions are particularly beneficial, namely a low label rate setting in semi-supervised node classification. In this case, deep models with NodeNorm significantly outperform the shallow ones.

The contributions of this paper are four-fold. First, we are among the first to analyze and show that the main reasons for performance degradation of deep GNN models with increasing model depth are training difficulty and overfitting. Second, we propose a Node Normalization technique termed NodeNorm that stabilizes the training process and reduces overfitting. Third, we identify a scenario where deeper models are more advantageous than shallow ones, i.e. the low label rate setting in semi-supervised tasks. Last but not least, experiments across various popular architectures and public benchmark datasets demonstrate that our method establishes a new state-of-the-art on deep GNNs. To the best of our knowledge, this is the first time that the performance of deep models is on par with or even better than that of shallow ones on these benchmarks.

2 Why do deep GCNs fail?

Neural networks usually perform better with increasing depth [14, 27]. For GCNs, greater depth enables the capability to make use of longer range interactions, which can be beneficial in many scenarios. However, recent works [18, 4, 20] find that GCNs tend to suffer from performance degradation as they get deeper. In this section, we investigate this phenomenon with experiments.

We conduct ablation experiments of different operations in a GCN layer, and examine the effects of existing regularization and normalization techniques on the accuracy of the semi-supervised node classification task. Our experiments are conducted on three benchmark datasets from [24], i.e. Cora, Citeseer and Pubmed, but we only report results on Cora and refer interested readers to Appendix for results on the other two datasets, due to space limitations. We use the same dataset split as [16]. Note we do not use regularization unless stated otherwise to exclude their distraction upon results.

2.1 Notations

Given an undirected graph G with n nodes, let the adjacency matrix and the degree matrix of G be denoted as $A \in \{0, 1\}^{n \times n}$ and $D = \text{diag}(A\mathbf{1})$, where $\mathbf{1}$ is an n -dimension all-ones column vector. An L -layer GCN model [16] is composed of L cascaded feed-forward Graph Convolution (GC) layers. Formally, the l -th GC layer can be represented as

$$H^{(l+1)} = \text{ReLU}(\hat{A}H^{(l)}W^{(l)}), \quad (1)$$

where $H^{(l)} \in \mathbb{R}^{n \times d_l}$ and $W^{(l)} \in \mathbb{R}^{d_l \times d_{l+1}}$ denote the matrix of node embeddings and the matrix of learnable parameters of this layer. The i -th row ($i \in \{1, \dots, n\}$) of $H^{(l)}$, i.e. $\mathbf{h}_i^{\top(l)}$, denotes the input embedding vector of node i of the l -th layer. In this formula, \hat{A} is the re-normalized adjacency matrix and $\hat{A} = \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}}$, where $\tilde{A} = A + I$ and $\tilde{D} = \text{diag}(\tilde{A}\mathbf{1})$. According to Eqn. (1), a GC layer consists of a propagation operation and a transformation operation:

$$\bar{H}^{(l)} = \hat{A}H^{(l)} \text{ (propagation)}, \quad H^{(l+1)} = \text{ReLU}(\bar{H}^{(l)}W^{(l)}) \text{ (transformation)}. \quad (2)$$

The first operation propagates information from the 1-hop neighbors to each single node, while the latter transforms propagated embeddings via a linear transformation followed by a ReLU activation function. We examine each of them to see how the performance is influenced.

2.2 Is over-smoothing the main reason?

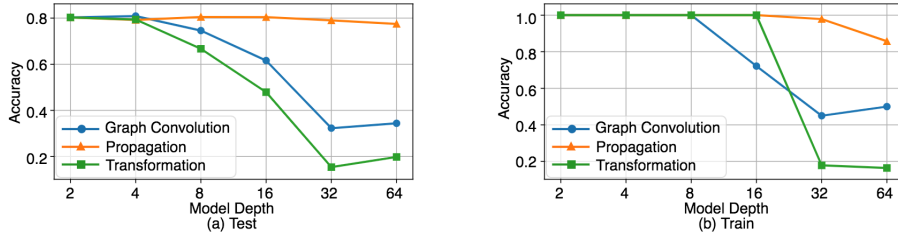


Figure 1: Accuracy w.r.t. model depth by stacking different modules.

Some works [18, 4, 20] attribute the classification performance degradation of deep GCNs to *over-smoothing*, i.e. node embeddings are pushed to be indistinguishable from each other due to propagation operations in stacked GC layers.

To verify this, we build three variants of L -layer GCNs by inserting different $(L-2)$ stacked modules between two GC layers, namely: 1) GC layers; 2) propagation operations; and 3) transformation operations. Formally, denote a GC layer as $\text{GC}(\cdot)$ and a transformation operation as $\text{T}(\cdot)$, and let $X \in \mathbb{R}^{n \times d}$ be the input feature matrix. Then, inserting a stack of GC layers yields a vanilla GCN model [16]; inserting the other two operations yields respectively

$$H^{(L)} = \text{GC}(\hat{A}^{L-2} \text{GC}(X)) \quad \text{and} \quad H^{(L)} = \text{GC}(\underbrace{\text{T} \circ \dots \circ \text{T}}_{L-2}(\text{GC}(X))). \quad (3)$$

Note that the parameters in different layers are not shared. We plot the test accuracy of the three variants with varying depths L in Fig. 1 (a). It can be seen that the variant with stacked transformation operations suffers accuracy decrease more drastically w.r.t. increasing depth, compared to the variant with stacked propagation operations. Our interpretation is that the over-smoothing problem is a contributing factor but is not the main reason. Instead, the stack of transformation operations causes more performance decline.

2.3 Main reason I: training difficulty caused by gradient vanishing

We then conduct further experiments to investigate how transformation operations in GC layers impact the performance w.r.t. model depth. Fig. 1 (b) shows that the training accuracy of deep GCNs is very low, which implies some difficulty for training GCNs. We check the gradients in the training process of GCNs, and find that gradient vanishing causes severe *training difficulty*. We use the

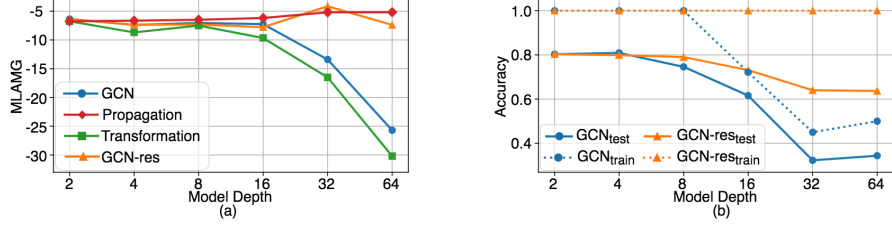


Figure 2: (a) MLAMG of different models. (b) Accuracy of GCN and GCN-res models. Solid, dotted lines denote test, training respectively.

Minimum Log Absolute value of the Mean Gradients (MLAMG) to illustrate the gradient vanishing problem, which is calculated as below with the gradient matrix of the first layer in the t -th epoch denoted as $G^t = \nabla_{W^{(1)}} \mathcal{L}^t \in \mathbb{R}^{d_l \times d_{l+1}}$:

$$\text{MLAMG} = \min_t \lg \left(\frac{1}{d_l d_{l+1}} \sum_{i=1}^{d_l} \sum_{j=1}^{d_{l+1}} |G_{ij}^t| \right), \quad (4)$$

where $|G_{ij}^t|$ is the absolute value of the (i, j) -th element of G^t . Smaller MLAMG means smaller gradients are propagated to the very bottom layer of the model. We plot the MLAMG of GCNs with different depths in Fig. 2 (a). We can see that deep GCNs suffer from gradient vanishing, and consequently fail to fit training data (Fig. 1 (b)). Some works [17, 21] point to over-smoothing as the cause for gradient vanishing. However, as shown in Fig. 2 (a), the variant built by inserting propagation operations does not suffer this issue. Instead, gradient vanishing is observed in the variant built by inserting stacked transformation operations. Therefore, we argue that gradient vanishing in GCNs is caused by the transformation operations in GCN layers rather than over-smoothing.

To address gradient vanishing, we add residual connections [14, 16] to every hidden layer of a GCN model, resulting in the GCN-res model, where

$$H^{(l+1)} = \text{ReLU}(\hat{A}H^{(l)}W^{(l)}) + H^{(l)}. \quad (5)$$

Fig. 2 shows that GCN-res resolves the gradient vanishing problem and fits well to the training data. However, we still observe performance degradation of GCN-res with increasing depth (Fig. 2 (b)).

2.4 Main reason II: overfitting and training instability

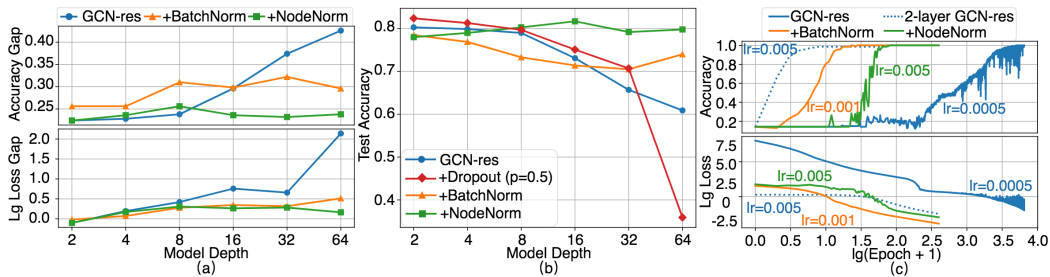


Figure 3: (a) Accuracy gap and loss gap of GCN-res between training and validation set. Note that the loss gap is shown in a log scale with base of 10. (b) Test accuracy of different models. Dropout rate is 0.5. (c) Training accuracy and loss across the training process of different models. Solid, dotted lines denote 64-layer models, 2-layer model respectively. Note that x-axis shows $\lg(\text{Epoch} + 1)$ to better compare models.

As shown in Fig. 3 (a), the loss gap and accuracy gap between training and validation set rise quickly as the GCN-res model gets deeper. This implies increasingly severe overfitting, which cannot be addressed by existing regularization techniques such as Dropout (Fig. 3 (b)). Besides, as shown in Fig. 3 (c), training curves of deep GCN-res models oscillate rapidly at the later stage of training, showing severe training instability. As a result, we need to carefully tune a small learning rate to

train the model, which slows down the training process. Additionally, due to training instability the training process of deep GCN-res models is sensitive to regularization. As illustrated in Fig. 3 (b), a dropout rate of 0.5 improves performance of GCN-res models with shallow and medium depths, but hurts that of deep GCN-res models with 64 layers. Please refer to Appendix for more results of applying existing regularization techniques upon GCN-res models.

To address overfitting and training instability, we consider normalization methods which have been shown to regularize neural networks [23, 9] and accelerate training [32]. Among them, BatchNorm [15] is the most widely-adopted. Although it helps accelerate the training process (Fig. 3 (c)) and improve deep model performance compared to GCN-res, the performance of deep models with BatchNorm is still inferior to that of a 2-layer GCN-res model (Fig. 3 (b)).

2.5 Disadvantage of batch normalization

To investigate why the performance of BatchNorm is unsatisfactory, we analyze the characteristics of nodes that are incorrectly classified by deep models. We observe that GCN-res and GCN-res with BatchNorm both perform poorly on nodes with features of high variance. To see this, we define the following node-wise deviation to measure the feature variance of each single node:

$$\begin{aligned}\sigma_i^{(l)} &= \sqrt{\frac{1}{d_l} \sum_{j=1}^{d_l} (h_{ij}^{(l)} - \mu_i^{(l)})^2}, \\ \mu_i^{(l)} &= \frac{1}{d_l} \sum_{j=1}^{d_l} h_{ij}^{(l)},\end{aligned}\tag{6}$$

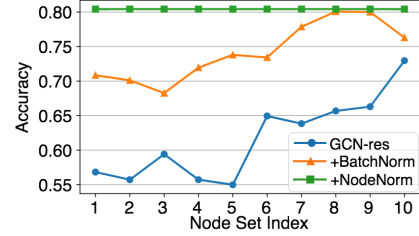


Figure 4: Classification accuracy of nodes in different sets.

where $h_{ij}^{(l)}$ denotes the j -th feature of hidden embedding of node i at the l -th layer, and μ_i is the mean of the embedding vector for node i . We then sort all nodes in Cora dataset by their node-wise deviation of the first layer in the descending order, and partition the sorted nodes evenly into ten sets. Formally, let S_1, \dots, S_{10} denote the ten sets. Then we have $\sigma_i^{(1)} \geq \sigma_j^{(1)}, \forall i \in S_p, j \in S_q, 1 \leq p < q \leq 10$. Please refer to Appendix for details on the partition.

Fig. 4 shows the performance on the ten sets. We can see that deep GCN-res models perform worse on the nodes with higher node-wise deviation than on other nodes, and that BatchNorm cannot solve this problem. A possible reason is that BatchNorm performs feature-wise normalization using the statistics of all nodes, thus cannot adjust the node-wise deviation of each individual node.

To address this problem, we propose a Node Normalization method that ensures all nodes have the same variance, thus giving good overall performance across all nodes, as shown in Fig. 4.

3 Proposed node normalization

3.1 Formulation

We propose Node Normalization (NodeNorm) to normalize the embedding of every node by its own mean and standard deviation. Let $\mathbf{h}_i^{(l)} \in \mathbb{R}^{d_l}$ denote the embedding vector of node i with d_l -dimensional features. The proposed NodeNorm operation is

$$\text{NodeNorm}(\mathbf{h}_i^{(l)}) = \frac{\mathbf{h}_i^{(l)} - \mu_i^{(l)}}{\sigma_i^{(l)}},\tag{7}$$

where $\mu_i^{(l)}$ and $\sigma_i^{(l)}$ are the node-wise mean and deviation of node i defined in Eqn. (6). No extra learnable parameters are introduced, which reduces the risk of overfitting. A GCN-res layer equipped with NodeNorm becomes

$$H^{(l+1)} = \text{ReLU}(\text{NodeNorm}(\hat{A}H^{(l)}W^{(l)})) + H^{(l)}.\tag{8}$$

We can see NodeNorm has a simple form and can be easily applied to existing GNN architectures.

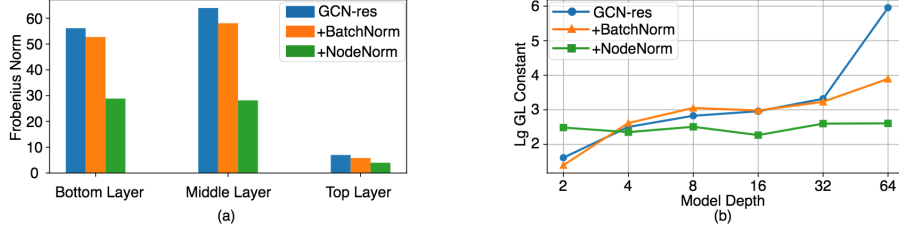


Figure 5: (a) Frobenius norm of feature correlation matrices of different hidden layers of 64-layer models. (b) GL Const of three models with different depths. Results are in log scale with base of 10.

3.2 How does NodeNorm help?

From Fig. 3, we can see the benefits of NodeNorm as compared to other variants. Firstly, Fig. 3 (c) shows that it stabilizes the training process, allowing for larger learning rate and thus speeding up the training process. Moreover, Fig. 3 (a) shows that GCN-res models with NodeNorm have smaller accuracy gap and loss gap than other variants, which implies less overfitting. As a result, NodeNorm successfully alleviates performance degradation on Cora (Fig. 3 (b)). In this subsection, to investigate how our NodeNorm reduces overfitting, we further conduct experiments to examine its influences on hidden embeddings of nodes and the trained models.

Reducing feature correlation Overfitting can be alleviated by decorrelating the hidden features[13]. We thus investigate whether NodeNorm can reduce feature correlation. We compare the correlation among hidden features of three 64-layer models: a GCN-res, a GCN-res with BatchNorm and a GCN-res with NodeNorm. Specifically, we compute the Frobenius norm of feature correlation matrices of a bottom layer (layer 22), a middle layer (layer 44) and a top layer (layer 64), as shown in Fig. 5 (a). It can be observed the model trained with NodeNorm has less correlated features than others, showing NodeNorm can effectively reduce overfitting.

Enhancing model smoothness Recently, some works [2, 10, 37] observe that networks enforced to have lower Lipschitz constant² tend to have better generalization ability and suffer less overfitting. Inspired by this, we investigate how NodeNorm affects model smoothness. We define the Graph Lipschitz Constant (GL Const) for GNNs to measure smoothness w.r.t. node features. Let $f(\mathbf{x}; G; \mathbf{w})$ denote a GNN, where \mathbf{x} is the input node feature vector, G is the input graph structure and \mathbf{w} is the model parameters. For a given graph G , the GL Const is

$$L_G = \max_{i,j \in V} \frac{\|f(\mathbf{x}_i; G, \mathbf{w}) - f(\mathbf{x}_j; G, \mathbf{w})\|}{\|\mathbf{x}_i - \mathbf{x}_j\|}. \quad (9)$$

From Eqn. 9, models with smaller L_G are less sensitive to disturbances in node features. We compare the L_G values of GCN-res and GCN-res with BatchNorm or NodeNorm, as shown in Fig. 5 (b). We can see that NodeNorm effectively reduces L_G and acts as an implicit regularizer in the training process of deep models. Moreover, under the common assumption in node classification that nodes with similar features belong to the same class, our method is advantageous since applying it increases the smoothness of the model w.r.t. node features. This kind of smoothness provides a good balance between feature information and structural information for deep GNNs since they generally focus more on the latter [33].

4 Evaluation of NodeNorm on training deeper GNNs

Experimental results in previous sections have shown the superiority of our method on one widely adopted split of Cora dataset. In this section, we conduct further experiments to demonstrate that our NodeNorm generalizes well to different GNN architectures and datasets.

²A function $f(\mathbf{x})$ is L -Lipschitz if $\|f(\mathbf{x}_1) - f(\mathbf{x}_2)\| \leq L\|\mathbf{x}_1 - \mathbf{x}_2\|, \forall \mathbf{x}_1, \mathbf{x}_2$. L is the Lipschitz constant.

4.1 Experiment setup

Datasets We evaluate our method on four widely used benchmark datasets: Cora, Citeseer, Pubmed [24] and CoauthorCS [25]. For the former three datasets, we follow the widely-adopted setting [16, 28, 33, 4] of the semi-supervised node classification task, namely 20 training nodes per class, 500 validation nodes and 1,000 test nodes. For CoauthorCS, we split it following the rules in [29]. With such low label rates, there is substantial randomness in the data for each split [25]. Therefore, we run each of the experiments on 50 random splits and report the average performance and standard deviation for more comprehensive evaluation in Tab. 1 and Tab. 3. However, in Tab. 2 we show only a single split as used in [16] to fairly compare against other reported results.

Implementation We apply NodeNorm to three well-known GNN models: GCN [16], GAT [28] and GraphSage [11]. To avoid gradient vanishing, we use GCN-res, GraphSage-res. However, for GAT baseline we use the vanilla version since gradient vanishing is not observed, but we add residual connections for our method as we find it slightly improves performance. We run experiments with depth $\{2, 4, 8, 16, 32, 64\}$ respectively. Please refer to Appendix for implementation details.

4.2 Effectiveness in resolving performance degradation

We first investigate whether our proposed NodeNorm can alleviate the performance degradation issue. As reported in Tab. 1, deep models augmented with NodeNorm significantly outperform the baselines. Moreover, this is the first time that the performance of deep GNNs is on par with or even better than shallow models on these benchmark datasets (see both Tab. 1 and 2).

Table 1: Test accuracy w.r.t. model depth. We only report results for models with 2, 16, 64 layers here due to space limitations and refer interested readers to Appendix for results of other layers.

Dataset	# layers	GCN-res	w/ NodeNorm	GAT	w/ NodeNorm	Sage-res	w/ NodeNorm
Cora	2	80.94 ± 1.32	80.15 ± 1.25	80.23 ± 1.56	80.21 ± 1.50	79.47 ± 1.38	78.56 ± 1.40
	16	75.03 ± 3.36	81.03 ± 1.39	78.98 ± 1.52	79.13 ± 1.44	73.46 ± 6.31	80.53 ± 1.59
	64	60.71 ± 4.17	80.68 ± 2.20	70.96 ± 2.67	79.26 ± 1.93	49.62 ± 9.11	81.09 ± 1.55
Citeseer	2	69.79 ± 1.48	68.56 ± 1.73	70.42 ± 1.34	68.53 ± 1.46	69.13 ± 1.63	68.40 ± 1.69
	16	59.23 ± 3.94	68.52 ± 1.76	65.06 ± 2.06	68.67 ± 2.44	61.19 ± 4.12	68.18 ± 1.77
	64	39.66 ± 3.92	68.14 ± 1.58	50.64 ± 3.20	67.88 ± 2.37	29.33 ± 6.42	68.38 ± 1.84
Pubmed	2	78.65 ± 2.05	79.42 ± 1.99	77.12 ± 2.47	77.44 ± 2.39	77.48 ± 1.96	76.98 ± 1.90
	16	75.24 ± 6.17	79.34 ± 2.20	77.33 ± 1.78	76.79 ± 2.08	76.66 ± 2.37	78.33 ± 2.63
	64	68.00 ± 3.40	79.20 ± 2.10	75.17 ± 2.05	76.15 ± 2.60	55.21 ± 9.82	78.38 ± 2.18
CoauthorCS	2	91.00 ± 0.76	91.15 ± 0.70	90.83 ± 0.60	90.29 ± 0.56	92.25 ± 0.63	92.18 ± 0.53
	16	83.44 ± 4.60	90.57 ± 0.72	85.01 ± 1.08	88.57 ± 0.97	77.50 ± 8.98	91.85 ± 0.64
	64	61.41 ± 3.15	90.45 ± 0.67	81.10 ± 1.25	88.33 ± 1.42	29.24 ± 17.13	91.74 ± 0.72

We further compare our method with two state-of-the-art techniques for improving deep GNNs: DropEdge [21] that reduces oversmoothing and overfitting by randomly dropping edges of the input graph structure, and Pairnorm [35] that addresses over-smoothing by inserting a normalization layer to keep total pairwise distance constant for all layers. Results on GCN backbone are given in Tab. 2. Note the results for DropEdge are from their github repository, while for Pairnorm, 2-layer results are reported in their paper, and we reproduce other results using their reported settings. Our method is substantially better in all cases, especially on deep models. Results in Tab. 1 and Tab. 2 show the effectiveness of our method in resolving the performance degradation issue of deep models.

4.3 Favorable setting for deeper GNNs

To further show the power of NodeNorm, we identify a scenario where deep models are particularly beneficial. Consider the semi-supervised node classification task: for some graphs where the label rate is very low, more layers may be needed to reach the supervised information located farther away. Tab. 3 shows the results of Pubmed with low label rates. We can see that deeper models achieve better performance than shallow ones for both our method and the baselines. In addition, as the label rate decreases, best performance is achieved by increasingly deeper models with NodeNorm, from which we can see a clear trend of needing more layers as the label rate decreases. In fact, in all the three cases, our 64-layer model outperforms the 2-layer model. Our method enables deep models to

Table 2: Performance comparison with state-of-the-art methods.

Dataset	Model	2-Layer	4-Layer	8-Layer	16-Layer	32-Layer	64-Layer
Cora	DropEdge	82.8	83.3	82.8	82.7	81.1	78.9
	PairNorm	78.3	77.5	76.8	78.8	75.9	77.8
	NodeNorm (ours)	83.3	83.9	83.4	83.8	82.5	83.4
Citeseer	DropEdge	72.3	72.2	71.6	70.1	70.0	65.1
	PairNorm	64.8	65.9	63.2	61.7	61.5	61.4
	NodeNorm (ours)	74.1	73.2	73.3	73.2	74.6	73.8
Pubmed	DropEdge	79.6	78.8	78.9	78.0	78.2	76.9
	PairNorm	75.6	77.0	76.2	78.1	76.8	73.7
	NodeNorm (ours)	80.7	80.5	80.7	79.7	80.8	80.4

achieve much better performance especially when there are only 5 labels per class. Please refer to Appendix for results on other datasets. In addition, from both Tab. 1 and Tab. 3 we can see that our method has smaller variance in performance than the baselines.

Table 3: Test accuracy w.r.t. model depth in low label rate cases. The shadowed numbers indicate the best performance across three models for each label rate case.

Label per class	Model	2-Layer	4-Layer	8-Layer	16-Layer	32-Layer	64-Layer
15	GCN-res	77.02 \pm 2.35	77.39 \pm 2.77	76.72 \pm 2.68	74.39 \pm 3.16	69.09 \pm 4.43	67.06 \pm 3.87
	PairNorm	74.64 \pm 2.36	75.60 \pm 3.47	75.54 \pm 2.92	75.57 \pm 2.30	74.82 \pm 3.05	74.66 \pm 2.41
	NodeNorm	77.41 \pm 2.29	77.86 \pm 2.47	77.86 \pm 2.90	78.35 \pm 2.11	77.77 \pm 2.97	78.11 \pm 2.76
10	GCN-res	75.17 \pm 2.88	75.63 \pm 3.54	74.89 \pm 3.30	73.07 \pm 3.61	67.21 \pm 4.51	64.62 \pm 5.45
	PairNorm	73.33 \pm 3.11	72.98 \pm 3.94	73.45 \pm 3.40	73.12 \pm 3.43	73.57 \pm 2.92	73.81 \pm 2.82
	NodeNorm	76.00 \pm 2.66	75.67 \pm 3.36	75.56 \pm 3.45	75.76 \pm 2.51	76.67 \pm 2.68	76.37 \pm 2.56
5	GCN-res	68.81 \pm 5.06	71.41 \pm 4.65	70.80 \pm 4.49	69.55 \pm 4.98	63.42 \pm 5.76	58.14 \pm 6.69
	PairNorm	66.84 \pm 5.6	67.99 \pm 4.53	68.47 \pm 4.54	70.12 \pm 4.29	69.35 \pm 4.37	69.43 \pm 4.78
	NodeNorm	70.26 \pm 4.76	70.64 \pm 5.51	71.46 \pm 4.22	70.98 \pm 4.38	72.06 \pm 4.22	73.09 \pm 4.72

5 Related works

Multiple works in GNNs have observed that existing GNNs tend to suffer performance degradation as their depth increases [16, 4]. The main reason is considered to be oversmoothing [18, 20]. To improve performance of deep GNNs, many methods have been proposed [4, 21, 35]. Chen et al. [4] introduce MAD gap to measure the degree of oversmoothing and propose an additional loss to alleviate performance decline. DropEdge [21] randomly removes edges during training to reduce oversmoothing and overfitting. PairNorm [35] introduces a normalization layer to fix total pairwise feature distances and thus alleviates oversmoothing. Recently, Yang et al. [34] claim GCNs are naturally anti-oversmoothing and overfitting is the cause of performance drop, and propose a mean-subtraction technique to alleviate the problem.

Normalization techniques [15, 1, 22] are widely applied in training deep neural networks. Batch Normalization (BatchNorm) [15] is the most widely adopted. Santurkar et al. [23] show BatchNorm smooths the optimization landscape, and encourages stronger Lipschitz continuity in networks. Additionally, some works [9, 19] show BatchNorm implicitly regularizes training. Layer Normalization (LayerNorm) [1] is proposed to overcome the difficulty of applying BatchNorm to recurrent neural networks, shown to outperform BatchNorm in natural language processing tasks. Node normalization is technically similar to LayerNorm, in that LayerNorm also normalizes each single datum by the mean and deviation of its own. However, NodeNorm differs from LayerNorm in two aspects. First, LayerNorm is for normalizing recurrent neural networks where applying BatchNorm is difficult, while ours aims to standardize all nodes so that they have a unit node-wise deviation. Second, NodeNorm does not need affine transformation after the normalization operation, and thus no extra parameters are introduced. This helps reduce the risk of overfitting.

6 Conclusion

In this work, we reveal that the main reasons for the performance decay problem of GCNs as model depth increases are training difficulty and overfitting. The former can be solved by applying residual connections, while the latter cannot be addressed by existing regularization techniques. Additionally, deep GCN-res suffers training instability. To address overfitting and training instability, we propose NodeNorm which normalizes each node using its own statistics. Empirical evaluations show that our method regularizes deep models by discouraging feature-wise correlation and increasing smoothness w.r.t. input node features. We also identify a scenario where deep GNN models outperform shallow ones, i.e. low label rates. Extensive experiments validate the effectiveness of our proposed method, and show that our method generalizes well to various popular GNN architectures and datasets.

References

- [1] Ba, J. L., Kiros, J. R., and Hinton, G. E. (2016). Layer normalization. *arXiv preprint arXiv:1607.06450*.
- [2] Bartlett, P. L., Foster, D. J., and Telgarsky, M. J. (2017). Spectrally-normalized margin bounds for neural networks. In *Advances in Neural Information Processing Systems*, pages 6240–6249.
- [3] Bruna, J., Zaremba, W., Szlam, A., and LeCun, Y. (2013). Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*.
- [4] Chen, D., Lin, Y., Li, W., Li, P., Zhou, J., and Sun, X. (2019). Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. *arXiv preprint arXiv:1909.03211*.
- [5] Cogswell, M., Ahmed, F., Girshick, R., Zitnick, L., and Batra, D. (2015). Reducing overfitting in deep networks by decorrelating representations. *arXiv preprint arXiv:1511.06068*.
- [6] Defferrard, M., Bresson, X., and Vandergheynst, P. (2016). Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in neural information processing systems*, pages 3844–3852.
- [7] Dehmamy, N., Barabási, A.-L., and Yu, R. (2019). Understanding the representation power of graph neural networks in learning graph topology. In *Advances in Neural Information Processing Systems*, pages 15387–15397.
- [8] Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. (2017). Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1263–1272. JMLR. org.
- [9] Gitman, I. and Ginsburg, B. (2017). Comparison of batch normalization and weight normalization algorithms for the large-scale image classification. *arXiv preprint arXiv:1709.08145*.
- [10] Gouk, H., Frank, E., Pfahringer, B., and Cree, M. (2018). Regularisation of neural networks by enforcing lipschitz continuity. *arXiv preprint arXiv:1804.04368*.
- [11] Hamilton, W., Ying, Z., and Leskovec, J. (2017a). Inductive representation learning on large graphs. In *Advances in neural information processing systems*, pages 1024–1034.
- [12] Hamilton, W. L., Ying, R., and Leskovec, J. (2017b). Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584*.
- [13] Hariharan, B. and Girshick, R. (2017). Low-shot visual recognition by shrinking and hallucinating features. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3018–3027.
- [14] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.
- [15] Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*.

- [16] Kipf, T. N. and Welling, M. (2016). Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.
- [17] Li, G., Muller, M., Thabet, A., and Ghanem, B. (2019). Deepgcns: Can gcns go as deep as cnns? In *Proceedings of the IEEE International Conference on Computer Vision*, pages 9267–9276.
- [18] Li, Q., Han, Z., and Wu, X.-M. (2018). Deeper insights into graph convolutional networks for semi-supervised learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*.
- [19] Luo, P., Wang, X., Shao, W., and Peng, Z. (2018). Towards understanding regularization in batch normalization. *arXiv preprint arXiv:1809.00846*.
- [20] Oono, K. and Suzuki, T. (2019). On asymptotic behaviors of graph cnns from dynamical systems perspective. *arXiv preprint arXiv:1905.10947*.
- [21] Rong, Y., Huang, W., Xu, T., and Huang, J. (2020). Dropedge: Towards deep graph convolutional networks on node classification. In *International Conference on Learning Representations*.
- [22] Salimans, T. and Kingma, D. P. (2016). Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Advances in neural information processing systems*, pages 901–909.
- [23] Santurkar, S., Tsipras, D., Ilyas, A., and Madry, A. (2018). How does batch normalization help optimization? In *Advances in Neural Information Processing Systems*, pages 2483–2493.
- [24] Sen, P., Namata, G., Bilgic, M., Getoor, L., Galligher, B., and Eliassi-Rad, T. (2008). Collective classification in network data. *AI magazine*, 29(3):93–93.
- [25] Shchur, O., Mumme, M., Bojchevski, A., and Günnemann, S. (2018). Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868*.
- [26] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958.
- [27] Telgarsky, M. (2016). Benefits of depth in neural networks. *arXiv preprint arXiv:1602.04485*.
- [28] Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., and Bengio, Y. (2017). Graph attention networks. *arXiv preprint arXiv:1710.10903*.
- [29] Verma, V., Qu, M., Lamb, A., Bengio, Y., Kannala, J., and Tang, J. (2019). Graphmix: Regularized training of graph neural networks for semi-supervised learning. *arXiv preprint arXiv:1909.11715*.
- [30] Wu, F., Zhang, T., Souza Jr, A. H. d., Fifty, C., Yu, T., and Weinberger, K. Q. (2019). Simplifying graph convolutional networks. *arXiv preprint arXiv:1902.07153*.
- [31] Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., and Philip, S. Y. (2020). A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*.
- [32] Xu, J., Sun, X., Zhang, Z., Zhao, G., and Lin, J. (2019). Understanding and improving layer normalization. In *Advances in Neural Information Processing Systems*, pages 4383–4393.
- [33] Xu, K., Li, C., Tian, Y., Sonobe, T., Kawarabayashi, K.-i., and Jegelka, S. (2018). Representation learning on graphs with jumping knowledge networks. *arXiv preprint arXiv:1806.03536*.
- [34] Yang, C., Wang, R., Yao, S., Liu, S., and Abdelzaher, T. (2020). Revisiting "over-smoothing" in deep gcns. *arXiv preprint arXiv:2003.13663*.
- [35] Zhao, L. and Akoglu, L. (2019). Pairnorm: Tackling oversmoothing in gnns. *arXiv preprint arXiv:1909.12223*.
- [36] Zhou, J., Cui, G., Zhang, Z., Yang, C., Liu, Z., Wang, L., Li, C., and Sun, M. (2018). Graph neural networks: A review of methods and applications. *arXiv preprint arXiv:1812.08434*.
- [37] Zou, D., Balan, R., and Singh, M. (2019). On lipschitz bounds of general convolutional neural networks. *IEEE Transactions on Information Theory*.

Appendix

In Appendix A, we show results of the analytical experiments on Citeseer and Pubmed, and some related details, to corroborate the discussions in Sec. 2 and Sec. 3. Then, in Appendix B, we show results of applying widely-adopted regularization techniques to GCN-res models. Moreover, implementation details of our experiments in Sec. 4 are given in Appendix C. Last but not least, we show some supplementary experiment results in Appendix D.

A Analytical Experiments on Citeseer and Pubmed

A.1 Is oversmoothing the main reason?

Fig. 6 and Fig. 7 show the test accuracy and training on Citeseer and Pubmed of the three variants of GCNs defined in Sec. 2.2.

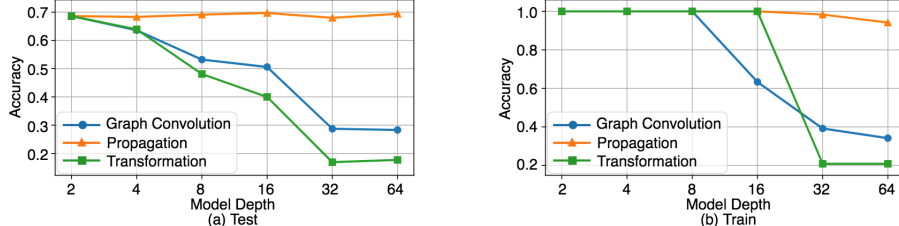


Figure 6: Accuracy w.r.t. model depth by stacking different modules (Citeseer).

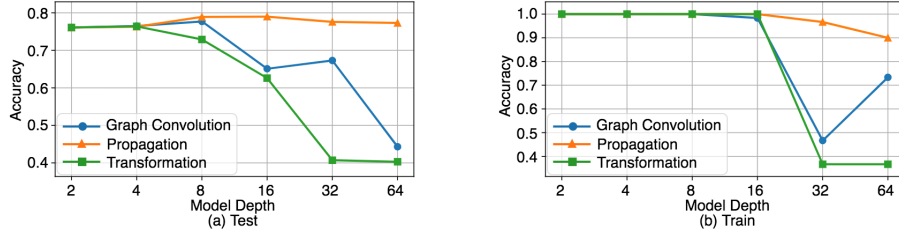


Figure 7: Accuracy w.r.t. model depth by stacking different modules (Pubmed).

A.2 Main reason I: training difficulty caused by gradient vanishing

In Sec. 2.3, we use the Minimum Log Absolute value of the Mean Gradients (MLAMG) of the first layer to illustrate the gradient vanishing problem, and plot the MLAMG on Cora in Fig. 2 (a). We further show the corresponding results on Citeseer and Pubmed in Fig 8 (a) and Fig 9 (a). The training and test accuracy of GCN and GCN-res models are shown in Fig 8 (b) and Fig 9 (b).

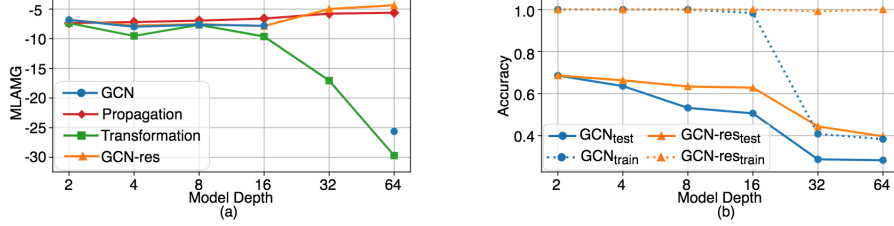


Figure 8: (a) MLAMG of different models. Note that the curve of GCN has no datum point when model depth is equal to 32, because the MLAMG goes to negative infinity for 32-layer GCN model. (b) Accuracy of GCN and GCN-res models. Solid, dotted lines denote test, training respectively. The results are on Citeseer.

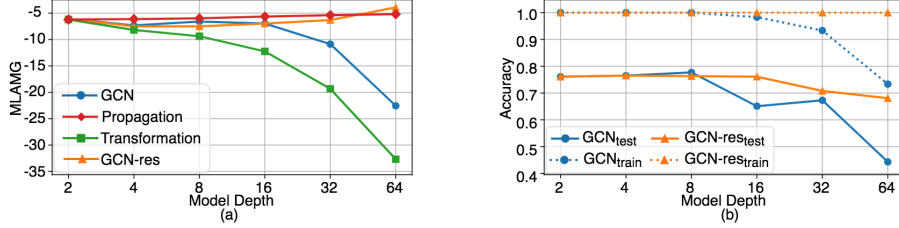


Figure 9: (a) MLAMG of different models. (b) Accuracy of GCN and GCN-res models. Solid, dotted lines denote test, training respectively. The results are on Pubmed.

In this subsection, to further illustrate the gradient vanishing problem across the training process, we define the Log Absolute value of Mean Gradient (LAMG) of various different hidden layers. Formally, the LAMG of layer l at the t -th epoch is calculated as:

$$\text{LAMG}_l^t = \lg \left(\frac{1}{d_l d_{l+1}} \sum_{i=1}^{d_l} \sum_{j=1}^{d_{l+1}} |(G_l^t)_{ij}| \right), \quad (10)$$

where G_l^t is the gradient matrix of $W^{(l)}$, and $|(G_l^t)_{ij}|$ is the absolute value of the (i, j) -th element of G_l^t . Let t_k denote the epoch at which $k\%$ of the whole training process has progressed. We then plot $\text{LAMG}_l^{t_k}$ of different hidden layers of 64-layer GCN and GCN-res models for $k = 0, 1, \dots, 100$ in Fig. 10, Fig. 11 and Fig. 12. We can observe that in a 64-layer GCN model, the learnable parameters of the top layers can be updated with sufficiently large gradients; however, those of the bottom layers can hardly be updated, as the gradients propagated to these layers are too small. It can also be seen that residual connections address this issue by enabling stable and sufficiently large gradients to be propagated to bottom layers.

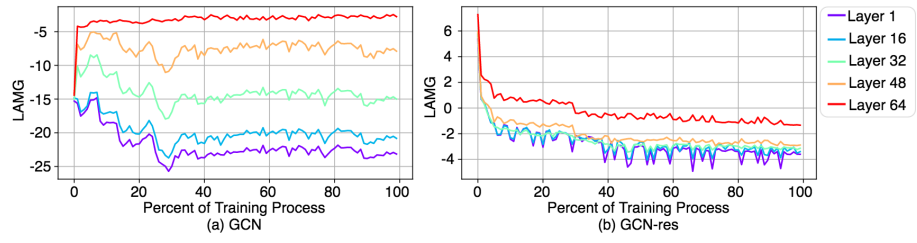


Figure 10: LAMG of different layers across the training process (Cora).

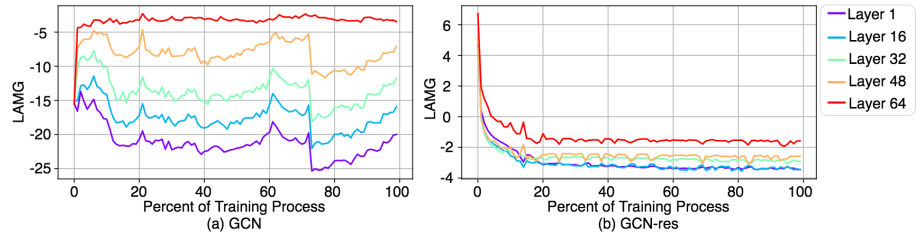


Figure 11: LAMG of different layers across the training process (Citeseer).

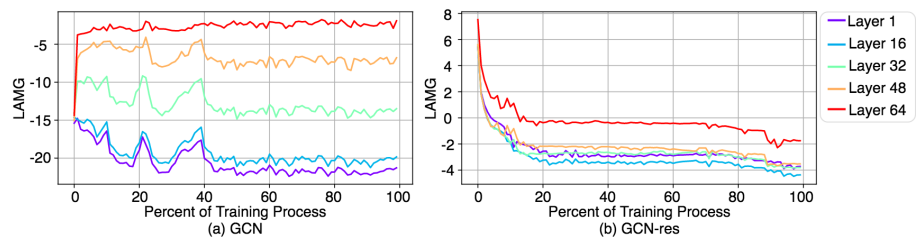


Figure 12: LAMG of different layers across the training process (Pubmed).

A.3 Main reason II: overfitting and training instability

Fig. 13 and Fig. 14 illustrate the overfitting and training instability problems of GCN-res models on Citeseer and Pubmed.

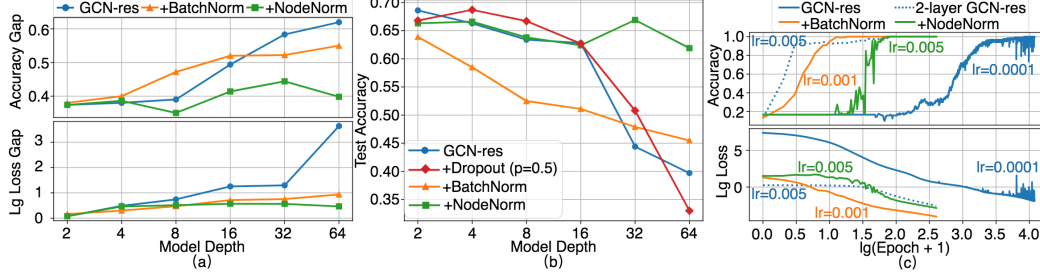


Figure 13: (a) Accuracy gap and loss gap of GCN-res between training and validation set. Note that the loss gap is shown in a log scale with base of 10. (b) Test accuracy of different models. Dropout rate is 0.5. (c) Training accuracy and loss across the training process of different models. Solid, dotted lines denote 64-layer models, 2-layer model respectively. Note that x-axis shows $\lg(\text{Epoch} + 1)$ to better compare models. Results are on Citeseer.

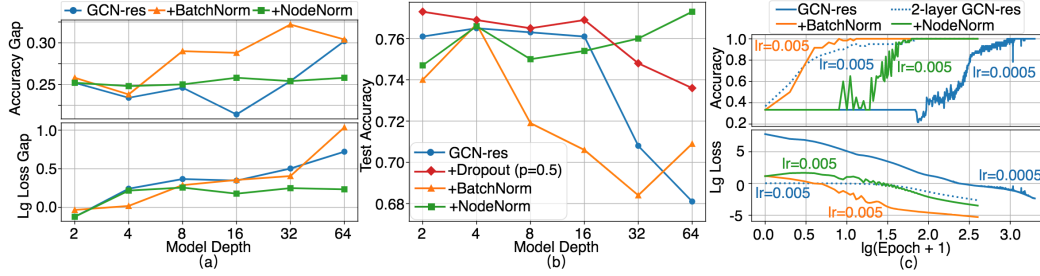


Figure 14: (a) Accuracy gap and loss gap of GCN-res between training and validation set. Note that the loss gap is shown in a log scale with base of 10. (b) Test accuracy of different models. Dropout rate is 0.5. (c) Training accuracy and loss across the training process of different models. Solid, dotted lines denote 64-layer models, 2-layer model respectively. Note that x-axis shows $\lg(\text{Epoch} + 1)$ to better compare models. Results are on Pubmed.

A.4 Disadvantage of batch normalization

Results on Citeseer and Pubmed Fig. 15 demonstrates that on Citeseer and Pubmed, GCN-res with BatchNorm performs poorly on nodes with large variance in features.

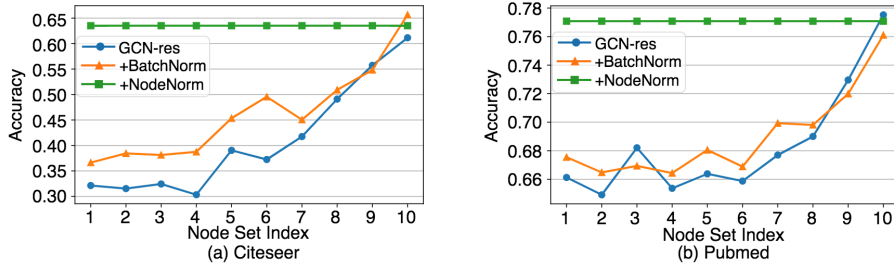


Figure 15: Classification accuracy of nodes in different sets. (a) Citeseer. (b) Pubmed.

Details about partition of nodes All n sorted nodes in a graph are partitioned into $(n \bmod 10)$ sets of size $\lfloor \frac{n}{10} \rfloor + 1$, and $(n - n \bmod 10)$ sets of size $\lfloor \frac{n}{10} \rfloor$.

A.5 How does NodeNorm help?

Fig. 16 and Fig. 17 show that NodeNorm regularizes deep GNNs effectively on Citeseer and Pubmed.

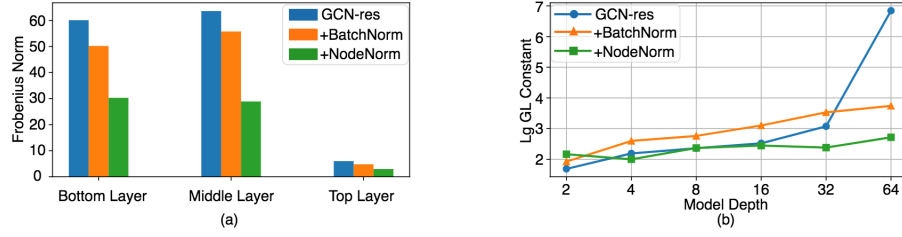


Figure 16: (a) Frobenius norm of feature correlation matrices of different hidden layers of 64-layer models. (b) GL Const of three models with different depths. Results are in log scale with base of 10. Results are on Citeseer.

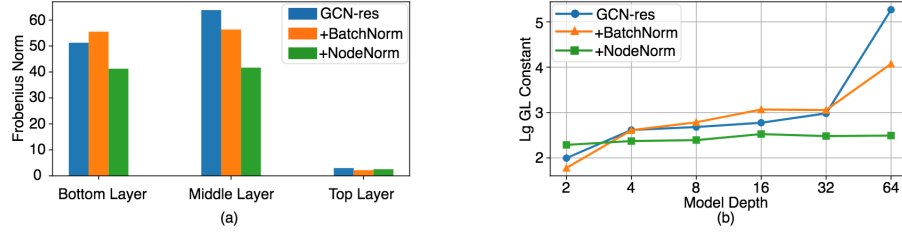


Figure 17: (a) Frobenius norm of feature correlation matrices of different hidden layers of 64-layer models. (b) GL Const of three models with different depths. Results are in log scale with base of 10. Results are on Pubmed.

B Effects of existing regularization techniques GCN-res training

In this subsection, we conduct experiments of applying three widely-adopted regularization techniques, i.e. dropout, weight decay, and ℓ_1 regularization to GCN-res models, and see how the performance is influenced.

Fig. 18 shows the results of applying dropout with a rate (dropping probability) of $\{0.1, 0.2, \dots, 0.9\}$ to GCN-res models. For each dataset, we show the results in three subplots, and also plot the result of GCN-res models without regularization (Baseline) in all subplots for better comparison. We also show the best result for each depth achieved by dropout.

Fig. 19 shows the results of applying weight decay with a factor of $\{10^{-5}, 5 \times 10^{-5}, 10^{-4}, 5 \times 10^{-4}, 10^{-3}, 5 \times 10^{-3}, 10^{-2}\}$ to GCN-res models. For each dataset, we show the results in three subplots, and also plot the result of GCN-res models without regularization (Baseline) in all subplots for better comparison. We also show the best result for each depth achieved by weight decay.

Fig. 20 shows the results of applying ℓ_1 regularization with a weight of $\{10^{-5}, 5 \times 10^{-5}, 10^{-4}, 5 \times 10^{-4}, 10^{-3}, 5 \times 10^{-3}, 10^{-2}\}$ to GCN-res models. For each dataset, we show the results in three subplots, and also plot the result of GCN-res models without regularization (Baseline) in all subplots for better comparison. We also show the best result for each depth achieved by ℓ_1 regularization.

We can see that 1) deep GCN-res models are sensitive to regularization techniques; with a fixed dropout rate (weight decay factor or ℓ_1 weight) that benefits shallow models, deep models may perform even worse than the baseline models without regularization; 2) these widely-adopted regularization techniques cannot resolve the problem of performance degradation, as the best performance achieved by the techniques also decreases as the models become deeper.

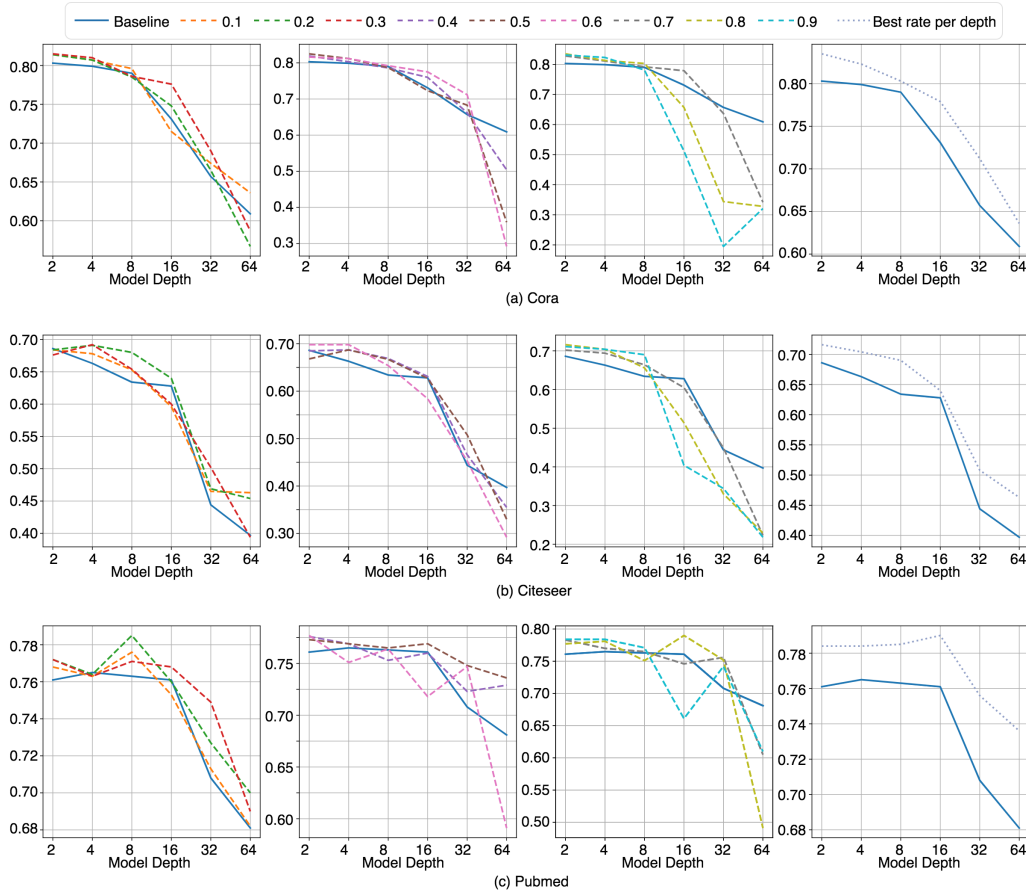


Figure 18: Test accuracy of applying dropout to GCN-res models. Each row corresponds to a single dataset. The first three columns show the results given by different dropout rate. The last column shows the best result achieved by dropout for each layer.

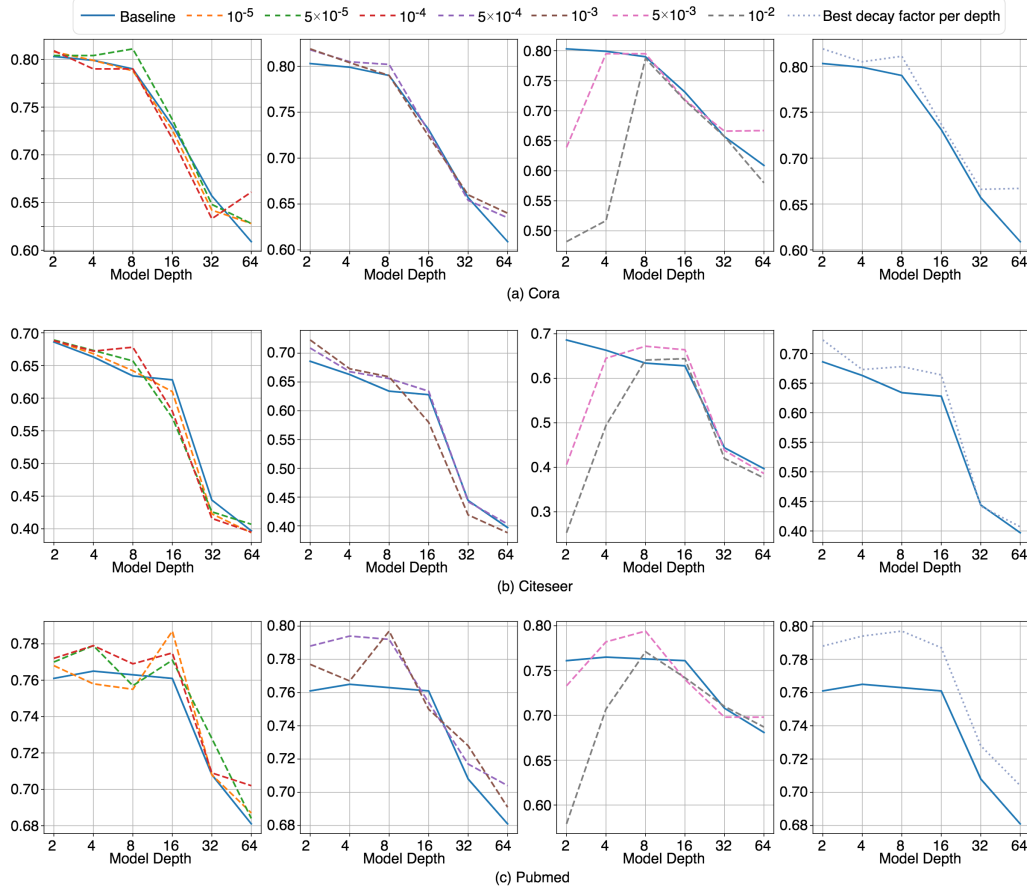


Figure 19: Test accuracy of applying weight decay to GCN-res models. Each row corresponds to a single dataset. The first three columns show the results given by different weight decay factor. The last column shows the best result achieved by weight decay for each layer.

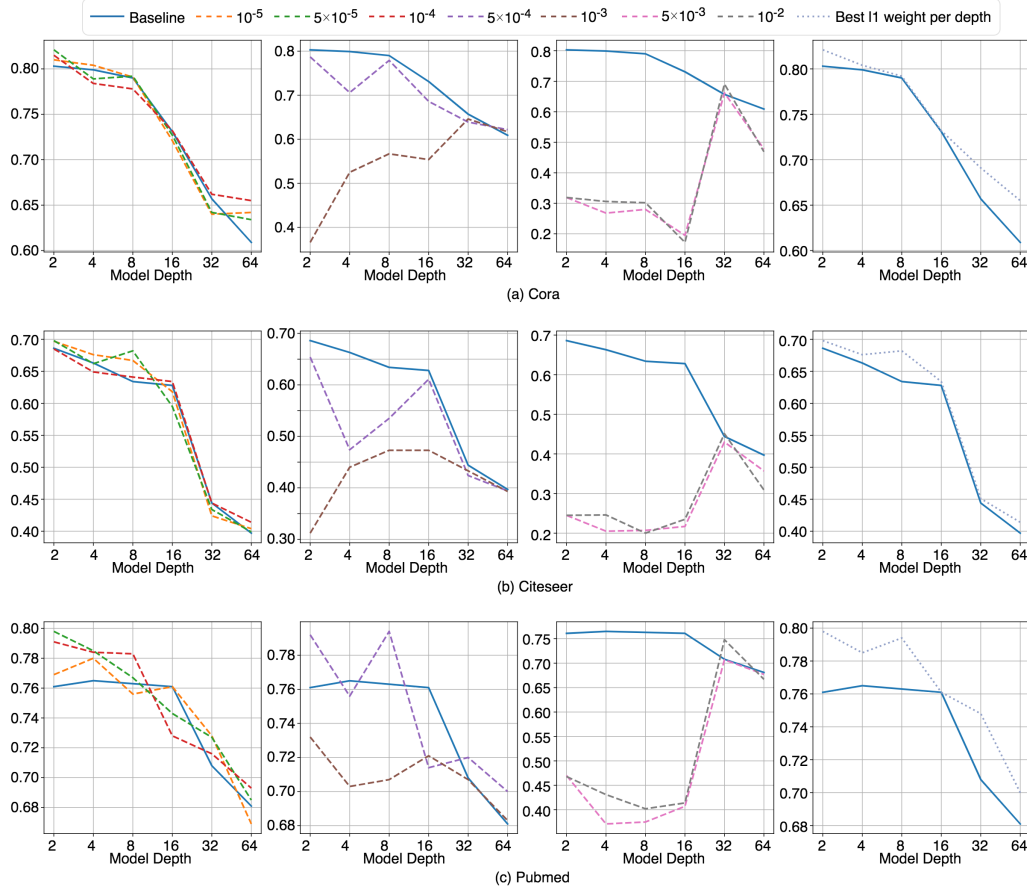


Figure 20: Test accuracy of applying ℓ_1 regularization to GCN-res models. Each row corresponds to a single dataset. The first three columns show the results given by different ℓ_1 weight. The last column shows the best result achieved by ℓ_1 regularization for each layer.

C Implementation Details

C.1 Hyperparameters

In this subsection, we specify the hyperparameters used for our method. Note that we choose different hyperparameters for baselines that are more suitable for them. In all the experiments with NodeNorm, we adopt the same learning rate: 0.005 and same number of epochs: 400. Following [16, 28, 11, 21, 35], we use dropout and weight decay as regularization during training. Moreover, ℓ_1 regularization is employed to further regularize GNN models. We use grid search to choose hyperparameters: dropout rate, ℓ_1 weight and weight decay factors, such that they satisfy the constraint that these hyperparameters in shallow models (with 2 or 4 layers), middle-depth models (with 8 or 16 layers) are shared separately as much as possible. The exact values of hyperparameters used in all tables except Tab. 2 are listed in Tab. 4 (for GCN-res with NodeNorm), Tab. 5 (for GAT-res with NodeNorm). Note that for experiments shown in Tab. 1 we also add residual connections to GAT models when applying NodeNorm to them, which we find will lead to slightly better performance than solely applying NodeNorm) and Tab. 6 (for Sage-res with NodeNorm). Following DropEdge [21], we enlarge the search space for Tab. 2 and allow separate hyperparameters for different depth, details are listed in Tab. 7.

Table 4: Hyperparameters of GCN-res with NodeNorm

Dataset	# layers	dropout rate	ℓ_1 weight	weight decay factor
Cora	2	0.8	0.001	0.001
	4	0.8	0.001	0.001
	8	0.8	0.001	0.0005
	16	0.8	0.001	0.0005
	32	0.8	0.001	0.001
	64	0.8	0.001	0.0005
Citeseer	2	0.8	0.001	0.001
	4	0.8	0.001	0.001
	8	0.8	0.001	0.001
	16	0.8	0.001	0.001
	32	0.8	0.001	0.001
	64	0.7	0.005	0.0005
Pubmed	2	0.8	0.005	0.001
	4	0.8	0.005	0.001
	8	0.8	0.005	0.001
	16	0.8	0.005	0.001
	32	0.8	0.005	0.001
	64	0.8	0.005	0.001
CoauthorCS	2	0.8	0	0.001
	4	0.8	0	0.001
	8	0.8	0	0
	16	0.8	0	0
	32	0.8	0	0.001
	64	0.8	0	0

Table 5: Hyperparameters of GAT-res with NodeNorm

Dataset	# layers	dropout rate	ℓ_1 weight	weight decay factor
Cora	2	0.7	0.01	0.001
	4	0.7	0.01	0.001
	8	0.6	0.01	0.0005
	16	0.6	0.01	0.0005
	32	0.5	0.01	0.01
	64	0.7	0.01	0.0005
Citeseer	2	0.7	0.01	0
	4	0	0.005	0.001
	8	0.6	0.0001	0.001
	16	0.6	0	0.0005
	32	0.6	0.001	0.001
	64	0	0.0005	0.001
Pubmed	2	0.7	0.01	0.001
	4	0.7	0.01	0.001
	8	0	0.01	0
	16	0	0.01	0
	32	0.7	0.01	0.0005
	64	0.7	0.01	0.0005
CoauthorCS	2	0.5	0.005	0.001
	4	0.5	0.005	0.001
	8	0.5	0.0005	0
	16	0.5	0.0005	0
	32	0.8	0.001	0
	64	0.7	0.0001	0

Table 6: Hyperparameters of Sage-res with NodeNorm

Dataset	# layers	dropout rate	ℓ_1 weight	weight decay factor
Cora	2	0.8	0.0001	0.0005
	4	0.8	0.0001	0.0005
	8	0.8	0.0005	0.001
	16	0.8	0.0005	0.001
	32	0.8	0.001	0.001
	64	0.8	0.001	0
Citeseer	2	0.8	0.001	0
	4	0.8	0.001	0
	8	0.8	0.001	0
	16	0.8	0.001	0
	32	0.8	0.001	0.001
	64	0.8	0.005	0.001
Pubmed	2	0.8	0.001	0.001
	4	0.8	0.001	0.001
	8	0.8	0.005	0.0005
	16	0.8	0.005	0.0005
	32	0.8	0.01	0
	64	0.8	0.01	0.001
CoauthorCS	2	0.5	0	0.0005
	4	0.5	0	0.0005
	8	0.5	0	0.001
	16	0.5	0	0.001
	32	0.5	0	0.0005
	64	0.5	0	0.0005

Table 7: Hyperparameters used in Tab. 2

Dataset	# layers	dropout rate	ℓ_1 weight	weight decay factor
Cora	2	0.7	0.003	0.005
	4	0.7	0	0.0001
	8	0.9	0.0001	0.0005
	16	0.8	0.001	0.001
	32	0.8	0.001	0.0005
	64	0.9	0.0005	0.005
Citeseer	2	0.9	0.0005	0.0003
	4	0.5	0.003	0.0001
	8	0.7	0.005	0.0001
	16	0.7	0.0008	0.0008
	32	0.8	0.005	0.001
	64	0.9	0.0008	0.0005
Pubmed	2	0.8	0.005	0.0008
	4	0.7	0.008	0.0008
	8	0.9	0.003	0.0008
	16	0.9	0.001	0.001
	32	0.9	0.003	0.0003
	64	0.9	0.005	0.0008

C.2 Hardware devices

In Sec. 4, more than 95% of the experiments are run on NVIDIA Tesla V100 GPU, while the rest are run on NVIDIA GeForce GTX TITAN X GPU or NVIDIA GeForce RTX 2080 Ti GPU.

C.3 The gradient clipping trick

When training deep GCN-res models, we observe that the gradient flows can be unstable, which slows down the convergence. To see this, we plot the LAMG of all layers and the corresponding loss curve of a 64-layer GCN-res model across the whole training process in Fig. 21 (a). We can see that there are some sharp peaks of the LAMG curves, which means the gradients suddenly soar at some epochs during the training process. As a result, the loss curve also suddenly jumps at these epochs, which hurts the convergence of the training process.

To address this problem, we introduce a *gradient clipping* trick, which clips the gradient vector according to the norm of gradient vectors in a window of previous epochs. Formally, let $\mathbf{g}_t \in \mathbb{R}^N$ denote the gradient vector of all N learnable parameters of the model at the t -th epoch. Then, \mathbf{g}_t is clipped by

$$\mathbf{g}_t = \begin{cases} \mathbf{g}_t, & \text{if } \|\mathbf{g}_t\| < c; \\ c \frac{\mathbf{g}_t}{\|\mathbf{g}_t\|}, & \text{otherwise,} \end{cases} \quad (11)$$

where c is the mean of the gradient norms in the previous w epochs, and w is the window size, namely,

$$c = \frac{1}{w} \sum_{\tau=1}^w \|\mathbf{g}_{t-\tau}\| \quad (12)$$

Note that we do not clip the gradients in the first w epochs of the training process. From Eqn. 11 and Eqn. 12, we can see the norm of the gradient vector will be no larger than the mean of the gradient norms in the window. Therefore, the LAMG and loss curves would not suddenly soar, as shown in Fig. 21 (b).

We do not observe such an unstable gradient problem in models with NodeNorm (Fig. 22 (a)), which also suggests that NodeNorm helps to maintain steady gradient flows and thus stabilizes the training process. As such, applying the gradient clipping trick does not influence the training process of models with NodeNorm, as shown in Fig. 22 (b). However, for fair comparison, we applying this trick in all experiments shown in our paper.

C.4 The Cora Trick

We also introduce a trick for Cora dataset. We find that for GCN-res with NodeNorm on Cora, removing the NodeNorm operation at the first layer leads to slightly better performance. We do not observe this phenomenon on other GNN architectures or datasets. Note that we only use this trick in Sec. 4.

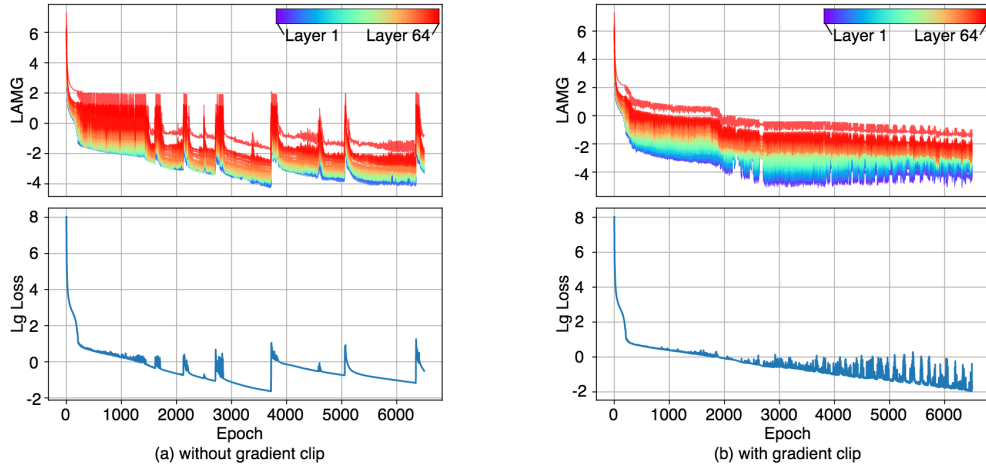


Figure 21: Upper: LAMG of different layers across the training process. Lower: Training loss across the training process, shown in log scale with base of 10. Results are obtained with a 64-layer GCN-res model. (a) With gradient clipping trick. (b) Without gradient clipping trick.

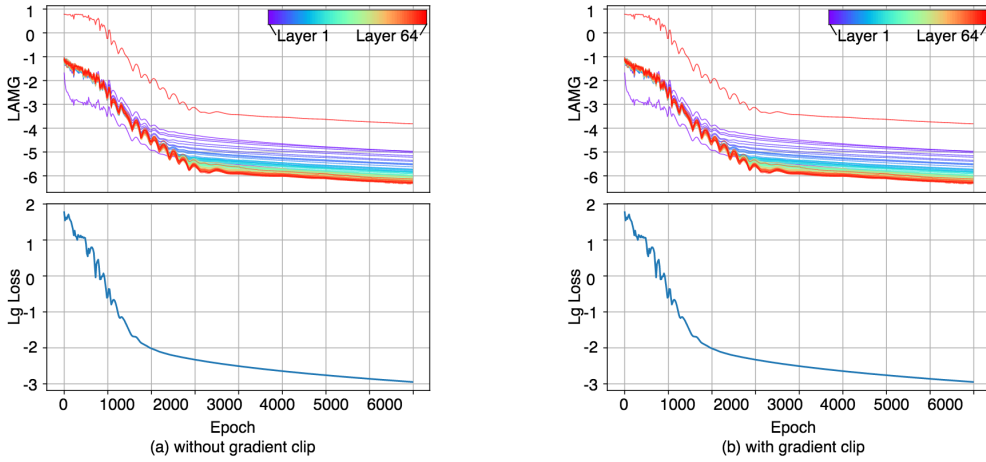


Figure 22: Upper: LAMG of different layers across the training process. Lower: Training loss across the training process, shown in log scale with base of 10. Results are obtained with a 64-layer GCN-res with NodeNorm model. (a) With gradient clipping trick. (b) Without gradient clipping trick.

D Supplementary experimental results

D.1 Results on more layers for Tab. 1

Tab. 8 shows test accuracy of GNNs of depth $\{4,8,32\}$ with and without NodeNorm. This supplements the results of Tab. 1.

Table 8: Test accuracy w.r.t model depth

Dataset	# layers	GCN-res	w/ NodeNorm	GAT	w/ NodeNorm	Sage-res	w/ NodeNorm
Cora	4	80.93 \pm 1.80	81.36 \pm 1.53	79.52 \pm 1.52	79.53 \pm 1.77	80.74 \pm 1.84	79.77 \pm 2.09
	8	78.91 \pm 1.74	81.32 \pm 1.53	80.58 \pm 1.45	79.17 \pm 1.68	78.48 \pm 1.72	79.62 \pm 1.61
	32	65.71 \pm 4.30	80.86 \pm 1.46	76.05 \pm 1.61	79.38 \pm 1.62	70.43 \pm 4.74	80.45 \pm 1.53
Citeseer	4	68.17 \pm 1.95	68.25 \pm 1.84	67.31 \pm 1.90	68.80 \pm 2.14	68.49 \pm 1.69	67.71 \pm 2.4
	8	64.99 \pm 2.18	68.87 \pm 1.81	67.29 \pm 1.94	68.60 \pm 2.52	65.05 \pm 2.60	68.45 \pm 1.88
	32	41.89 \pm 4.10	68.18 \pm 2.42	65.27 \pm 2.26	68.79 \pm 2.67	55.48 \pm 4.55	68.43 \pm 2.29
Pubmed	4	78.37 \pm 2.71	79.00 \pm 2.25	77.58 \pm 2.08	76.10 \pm 1.87	78.12 \pm 2.13	77.56 \pm 2.35
	8	78.24 \pm 1.76	79.24 \pm 2.05	78.35 \pm 1.96	76.97 \pm 2.21	76.53 \pm 3.12	77.94 \pm 2.08
	32	70.21 \pm 3.53	78.96 \pm 2.56	77.09 \pm 2.06	76.65 \pm 2.38	75.76 \pm 2.61	78.31 \pm 1.95
CoauthorCS	4	89.68 \pm 0.68	90.99 \pm 0.92	88.69 \pm 0.74	89.00 \pm 1.02	91.27 \pm 0.51	92.13 \pm 0.50
	8	87.98 \pm 0.85	90.74 \pm 0.63	86.58 \pm 0.96	88.75 \pm 1.14	89.04 \pm 0.88	91.93 \pm 0.60
	32	70.54 \pm 5.05	90.67 \pm 0.81	79.11 \pm 2.34	88.55 \pm 1.23	39.15 \pm 18.2	91.93 \pm 0.61

D.2 Low label rate

Tab. 10 and Tab. 11 shows the performance of low label rate setting for Cora and Citeseer dataset respectively. Similar to Pubmed, we can observe a trend of needing more layers when the label rate decreases. For Cora, the optimal model depth is 16 layers when there are only 5 labels per class, and this is obtained by our method. For Citeseer, the optimal is achieved by the baseline 4-layer GCN-res, it is still deeper than the 2-layer model used under the benchmark setting. The 64-layer model with NodeNorm achieves comparable performance as well. Note that the hyperparameters used here is the same as Tab. 1 and Tab. 8, we did not specifically tune them for the low label rate setting.

We observe that Pubmed benefits more from depth than the other two datasets. Moreover, we do not observe the advantage of deep models for CoauthorCS dataset (see Tab. 12). To understand why, we examine the difference in graph structure between these four datasets. We measure the degree of clustering tendency of a graph by clustering coefficient. Global clustering coefficient indicates the overall graph clustering tendency whereas local clustering coefficient measures single nodes' embeddedness. As shown in Tab. 9, Pubmed is the least clustered while CoauthorCS is the most clustered. Our conjecture is that less clustered graphs will benefit more from depth.

Table 9: Clustering coefficient of different datasets

Dataset	Global Clustering Coeff.	Average Clustering Coeff.
Cora	0.0935	0.2407
Citeseer	0.1301	0.1415
Pubmed	0.0537	0.0602
CoauthorCS	0.1826	0.3425

Table 10: Cora dataset: test accuracy w.r.t model depth in low label rate cases

Label per class	Model	2-Layer	4-Layer	8-Layer	16-Layer	32-Layer	64-Layer
15	GCN-res	79.50 \pm 1.99	79.68 \pm 1.68	77.82 \pm 2.28	72.7 \pm 4.87	63.03 \pm 4.08	57.39 \pm 5.59
	PairNorm	73.06 \pm 2.83	75.44 \pm 2.25	74.24 \pm 2.83	74.87 \pm 2.75	76.1 \pm 2.31	76.59 \pm 2.21
	NodeNorm	78.71 \pm 1.73	80.05 \pm 1.76	79.85 \pm 1.57	80.00 \pm 1.88	79.73 \pm 1.64	79.61 \pm 1.92
10	GCN-res	77.21 \pm 1.93	78.45 \pm 2.01	75.34 \pm 2.97	70.84 \pm 4.81	59.34 \pm 4.29	52.16 \pm 5.46
	PairNorm	70.07 \pm 2.30	72.47 \pm 2.89	72.42 \pm 3.51	74.03 \pm 2.63	74.01 \pm 3.04	74.35 \pm 2.67
	NodeNorm	76.2 \pm 1.98	78.80 \pm 2.01	78.35 \pm 2.18	78.05 \pm 2.04	77.62 \pm 2.0	77.48 \pm 2.16
5	GCN-res	72.26 \pm 3.45	73.92 \pm 3.18	70.42 \pm 5.35	67.68 \pm 5.66	53.04 \pm 6.80	43.75 \pm 5.41
	PairNorm	63.77 \pm 4.49	66.34 \pm 3.93	65.68 \pm 4.45	69.96 \pm 3.68	70.24 \pm 3.28	70.29 \pm 3.70
	NodeNorm	71.76 \pm 2.77	73.95 \pm 2.69	73.94 \pm 3.08	74.42 \pm 2.88	73.57 \pm 3.02	72.86 \pm 3.62

Table 11: Citeseer dataset: test accuracy w.r.t model depth in low label rate cases

Label per class	Model	2-Layer	4-Layer	8-Layer	16-Layer	32-Layer	64-Layer
15	GCN-res	68.68 \pm 2.14	67.53 \pm 1.87	63.93 \pm 2.59	58.35 \pm 4.13	39.85 \pm 4.22	37.40 \pm 3.71
	PairNorm	61.30 \pm 2.46	61.27 \pm 2.47	60.81 \pm 2.67	61.24 \pm 2.86	61.85 \pm 2.78	61.00 \pm 2.81
	NodeNorm	67.41 \pm 2.16	67.56 \pm 1.92	67.8 \pm 1.83	67.59 \pm 1.77	67.35 \pm 1.94	67.49 \pm 2.02
10	GCN-res	66.45 \pm 1.72	66.47 \pm 2.67	62.27 \pm 3.06	56.61 \pm 5.55	36.71 \pm 4.99	34.97 \pm 3.38
	PairNorm	57.64 \pm 2.68	59.20 \pm 2.83	58.55 \pm 3.19	59.11 \pm 3.40	59.15 \pm 2.95	58.94 \pm 2.94
	NodeNorm	65.71 \pm 1.96	65.90 \pm 2.41	65.64 \pm 2.23	65.28 \pm 2.69	64.80 \pm 2.27	65.71 \pm 1.92
5	GCN-res	59.56 \pm 3.66	61.65 \pm 3.81	57.74 \pm 5.30	54.0 \pm 5.25	34.51 \pm 4.55	29.81 \pm 3.43
	PairNorm	49.88 \pm 4.33	52.22 \pm 4.53	53.98 \pm 3.78	55.22 \pm 3.46	55.70 \pm 3.81	53.32 \pm 4.60
	NodeNorm	59.82 \pm 3.81	59.39 \pm 4.44	60.49 \pm 3.90	59.82 \pm 3.36	60.88 \pm 4.00	61.19 \pm 3.99

Table 12: CoauthorCS dataset: test accuracy w.r.t model depth in low label rate cases

Label per class	Model	2-Layer	4-Layer	8-Layer	16-Layer	32-Layer	64-Layer
15	GCN-res	90.75 \pm 0.64	89.08 \pm 0.99	87.39 \pm 1.10	83.17 \pm 3.76	71.16 \pm 5.47	58.33 \pm 4.71
	PairNorm	84.90 \pm 2.43	87.20 \pm 1.54	87.04 \pm 2.09	87.19 \pm 1.22	86.25 \pm 1.35	86.09 \pm 1.14
	NodeNorm	91.23 \pm 0.82	90.60 \pm 0.90	90.43 \pm 0.74	90.33 \pm 0.88	90.24 \pm 0.83	90.05 \pm 0.93
10	GCN-res	90.11 \pm 0.89	88.37 \pm 1.16	86.74 \pm 1.18	81.20 \pm 6.73	67.78 \pm 6.63	51.92 \pm 5.49
	PairNorm	83.25 \pm 3.07	86.16 \pm 1.61	85.77 \pm 2.16	85.69 \pm 1.83	85.51 \pm 1.60	84.75 \pm 1.56
	NodeNorm	90.92 \pm 0.73	89.86 \pm 1.34	89.79 \pm 0.89	89.76 \pm 0.89	89.72 \pm 0.91	89.43 \pm 1.09
5	GCN-res	88.18 \pm 1.59	86.50 \pm 1.87	84.83 \pm 1.93	78.60 \pm 4.28	59.82 \pm 7.74	39.71 \pm 5.15
	PairNorm	79.98 \pm 3.80	82.32 \pm 2.79	81.52 \pm 3.66	82.29 \pm 2.62	81.91 \pm 2.45	81.72 \pm 2.82
	NodeNorm	89.53 \pm 1.29	88.60 \pm 1.36	88.02 \pm 1.67	88.41 \pm 1.25	88.30 \pm 1.30	87.40 \pm 2.06

D.3 Ablation study

We then train GCN-res models with NodeNorm without using regularization techniques, i.e. dropout, ℓ_1 regularization and weight decay, to further demonstrate the effect of our proposed NodeNorm. The results are shown in Tab. 13.

Table 13: Test accuray of GCN-res models with NodeNorm without using regularization.

Backbone Model	Dataset	2-Layer	4-Layer	8-Layer	16-Layer	32-Layer	64-Layer
GCN-res	Cora	77.34 \pm 1.48	78.29 \pm 2.6	78.31 \pm 1.74	77.45 \pm 2.10	77.87 \pm 2.17	77.78 \pm 2.42
	Citeseer	64.24 \pm 2.21	64.14 \pm 2.25	63.46 \pm 2.18	63.24 \pm 2.72	62.62 \pm 2.91	63.04 \pm 2.30
	Pubmed	76.64 \pm 2.48	76.04 \pm 2.78	77.18 \pm 2.56	76.93 \pm 2.84	76.73 \pm 2.98	76.52 \pm 2.71
	CoauthorCS	92.00 \pm 0.60	90.73 \pm 0.98	90.54 \pm 0.64	90.46 \pm 0.78	90.55 \pm 0.65	90.38 \pm 0.81
GAT	Cora	77.50 \pm 2.02	77.70 \pm 2.01	77.35 \pm 2.02	77.47 \pm 2.03	77.35 \pm 2.37	77.66 \pm 1.89
	Citeseer	68.48 \pm 2.07	64.49 \pm 2.35	65.56 \pm 2.64	65.56 \pm 2.65	63.72 \pm 2.36	64.90 \pm 2.63
	Pubmed	75.41 \pm 2.54	75.58 \pm 2.18	76.08 \pm 2.21	75.63 \pm 2.28	75.99 \pm 2.54	75.15 \pm 3.09
	CoauthorCS	89.46 \pm 0.78	88.91 \pm 1.04	88.72 \pm 1.01	87.98 \pm 1.18	87.64 \pm 1.54	87.47 \pm 1.75
GraphSage-res	Cora	73.16 \pm 1.95	73.74 \pm 2.11	73.17 \pm 3.47	73.43 \pm 2.91	72.04 \pm 3.11	72.15 \pm 4.12
	Citeseer	62.36 \pm 1.87	62.54 \pm 2.41	61.46 \pm 3.11	60.17 \pm 3.15	58.46 \pm 3.96	57.28 \pm 3.55
	Pubmed	73.85 \pm 2.10	74.24 \pm 2.87	73.78 \pm 2.82	73.34 \pm 3.13	72.34 \pm 2.92	72.40 \pm 2.96
	CoauthorCS	91.56 \pm 0.93	91.39 \pm 0.82	91.26 \pm 0.82	91.43 \pm 0.61	90.81 \pm 0.92	90.33 \pm 1.04