Understanding and Resolving Performance Degradation in Graph Convolutional Networks

Kuangqi Zhou*, Yanfei Dong*, Kaixin Wang, Wee Sun Lee, Bryan Hooi, Huan Xu, Jiashi Feng

Abstract—A Graph Convolutional Network (GCN) stacks several layers and in each layer performs a PROPagation operation (PROP) and a TRANsformation operation (TRAN) for learning node representations over graph-structured data. Though powerful, GCNs tend to suffer performance drop when the model gets deep. Previous works focus on PROPs to study and mitigate this issue, but the role of TRANs is barely investigated. In this work, we study performance degradation of GCNs by experimentally examining how stacking only TRANs or PROPs works. We find that *TRANs contribute significantly, or even more than PROPs*, to declining performance, and moreover that they tend to amplify node-wise feature variance in GCNs, causing *variance inflammation* that we identify as a key factor for causing performance drop. Motivated by such observations, we propose a variance-controlling technique termed Node Normalization (NodeNorm), which scales each node's features using its own standard deviation. Experimental results validate the effectiveness of NodeNorm on addressing performance degradation of GCNs. Specifically, it enables deep GCNs to achieve comparable results with shallow ones on 6 benchmark datasets, and to outperform shallow ones in cases where deep models are needed. NodeNorm is a generic plug-in and can well generalize to other GNN architectures.

Index Terms—Graph-structured data, deep graph convolutional networks, performance degradation.

1 Introduction

In recent years, data representation learning has been greatly advanced by the development of deep neural network models. However, they are mostly designed to learn over structured data such as images [1] or natural languages [2], while in the real world, unstructured data [3] (a.k.a. non-Euclidean data [4], [5]) widely exist, such as social networks [6] and interactions among proteins [7]. These data are generally represented as graphs [3], [4], [8], with nodes and edges denoting data points and their relations, and Graph Neural Networks (GNNs) are applied [6], [9], [10], [11] to learn and mine information from such graph-structured data. GNNs stack several layers, and in each layer perform a *PROPagation operation* (PROP) and a *TRANsformation operation* (TRAN) [4] to produce informative node representations (*e.g.* high-dimensional feature vectors) that can be used to facilitate downstream tasks like graph node classification [4], [7], [12], [13].

Though achieving remarkable success, GNNs suffer a model depth limitation—they tend to perform increasingly worse on classifying graph nodes as the model gets deeper [14], [15], [16]. This performance degradation problem has been widely explored in previous literature [14], [15], [16], [17], [18], but with the focus on effects of PROPs operations of the Graph Convolutional Networks (GCNs). GCNs are a representative GNN architecture that uses the 1-order approximation of the Chebyshev polynomials of the graph Laplacian matrix [19] to perform graph convolution [4], [9]. Recent works [14], [15], [18] observe that PROPs in deep GCNs overly mix the hidden features of different nodes, a.k.a. *oversmoothing*, and hence hurt the node classification performance. Moreover, PROPs are also believed to cause *gradient vanishing*, thus hindering the model training process and leading to poor performance [16], [17].

However, the role of TRANs, the other major operation of a GCN layer, in this problem is largely neglected and less understood. In this work, we investigate on the effects of TRANs upon the performance of GCNs w.r.t. model depth, to provide a more comprehensive understanding of the reasons for the performance degradation problem of GCNs. In particular, we carefully design a set of ablative experiments where we disentangle the PROPs and TRANs to check their respective effects on model performance. We design two variants of GCNs: one only performs TRANs in hidden layers, and the other only performs PROPs in hidden layers for learning the node representations. We observe that the former model, *i.e.* performing only TRANs, generally causes *more* significant performance drop than performing only PROPs. This surprising finding differs from the previous common belief that oversmoothing and gradient vanishing caused by PROPs are the main reasons for performance degradation. Actually, *TRANs contribute significantly, or even more*.

Such intriguing observations motivate us to dig deeper into the reasons behind performance degradation. We then investigate how TRANs hurt the performance by examining their influence on the node representations. We observe that TRANs tend to amplify node-wise feature variance (*i.e.*, the variance of each node's hidden features). Consequently, as a GCN gets deeper, the number of TRANs increases and the node-wise feature variance in general increases greatly. We refer to this phenomenon as *variance inflammation*. Moreover, we find that larger variance of node features

* *Equal Contribution.*

- *Kuangqi Zhou, and Jiashi Feng are with the Department of Electrical and Computer Engineering, National University of Singapore.*
- *Kaixin Wang is with the NUS Graduate School for Integrative Sciences and Engineering, National University of Singapore.*
- *Yanfei Dong, Wee Sun Lee and Bryan Hooi are with the School of Computing, National University of Singapore.*
- *Yanfei Dong and Huan Xu are with Alibaba Group.*

leads to greater difficulty in classifying these nodes, and thus deep GCNs perform significantly worse than shallow ones. The effects of large feature variance on classification performance remind us of some early studies on Multi-Layer Perceptrons (MLPs) and Convolution Neural Networks (CNNs) [20], [21], [22]. Though those works similarly claim that the large feature variance would affect model performance, they do not establish the explicit relation to the difficulty of training a deeper model. Moreover, due to the complex entanglement between the PROPs and TRANs in GCNs, such a factor of large feature variance is hidden by the feature smoothing phenomenon among nodes (*i.e.*, oversmoothing).

We are motivated to mitigate *variance inflammation* to address the performance degradation issue for deep GCNs. To this end, we propose a plug-in variance-controlling technique termed as Node Normalization (NodeNorm) that can effectively alleviate *variance inflammation*. NodeNorm scales hidden features of each single node based on its standard deviation. To make the normalization effect controllable and collaborate well with GCNs in different scenarios, NodeNorm takes the p-th root of the standard deviation as the normalization factor. With a smaller p, NodeNorm controls node-wise feature variance more strictly. In the following part of our work, we use NodeNorm_p to denote NodeNorm with a specific p. We empirically find NodeNorm is effective for improving the performance of deep GCNs by well handling the *variance inflammation* problem. To further reveal the importance of variance controlling, we also investigate whether and why the existing Layer Normalization (LayerNorm) [22], which also performs a node-wise variance-scaling operation, helps reduce performance degradation. We find through experiments that LayerNorm also improves deep model performance by mitigating *variance inflammation*.

Extensive experiments on various types of graph datasets demonstrate that mitigating *variance inflammation* successfully relieves the performance degradation of deep GCNs. Specifically, we make following observations: 1) The NodeNorm_1, *i.e.*, NodeNorm with $p = 1$, a simple and strict variance-controlling technique, effectively resolves the performance degradation issue, enabling deep GCNs (*e.g.* 64-layer) to achieve comparable results with shallow ones (*e.g.* 2-layer) on six benchmark datasets [23]: three citation graphs [23] Cora, Citeseer and Pubmed, a co-authorship graph Coauthor-CS [12], a web-page graph Wiki-CS [24], and a product co-purchasing graph Amazon-photo [12]. 2) The NodeNorm is a simple and flexible method to control *variance inflammation* at different levels; with a smaller p, NodeNorm_p is able to mitigate performance degradation more effectively. (3) We reveal that the true contributing factor of LayerNorm' success to improving deep model performance is its variance scaling step, which is essentially our NodeNorm_1.

To further show the benefit of mitigating *variance inflammation*, we apply NodeNorm_1 to three exemplar cases where usually deep models are required to learn good node representations: citation graphs with missing features [18], citation graphs with low label rate [25], and geographical graphs [26] whose graph diameter is much larger than other commonly used graph datasets. We find NodeNorm_1 enables deep GCNs (*e.g.* 64-layer) to outperform shallow ones (*e.g.* 2-layer) in these cases significantly. We also compare NodeNorm_1 with two existing best-performance methods [17], [18] for addressing performance degradation in GNNs, and show that NodeNorm_1 outperforms them under the same setting. Furthermore, we show that applying variance-controlling techniques helps reduce overfitting in training deep GCNs, which is also a contributing factor of performance degradation [18].

Our proposed NodeNorm is generic and generalizable to other GNN architectures. To show this, we also perform several experiments with other GNN architectures. Again, we find that NodeNorm well resolves performance degradation for other GNNs, including GAT [10] and GraphSage [6]. It can also improve performance of recent GNN architectures that do not suffer performance degradation, including GCNII [27] and GEN [28].

The contributions of this paper are four-fold:

- We empirically find that TRANs make a significant cause of the performance degradation problem for deep GCNs, which is however under-explored in previous works.
- We figure out through experiments that TRANs cause *variance inflammation*, and that deep GCNs perform notably worse on nodes with relatively large variance as compared to shallow models. Based on these findings, we propose NodeNorm to mitigate *variance inflammation*.
- Our studies also show it is the ability of the LayerNorm to mitigate *variance inflammation* that makes it effective on enhancing performance of deep GCNs.
- Experiments across various benchmark datasets demonstrate the proposed NodeNorm well resolves performance degradation of GCNs, and enables deep GCNs (*e.g.* 64-layer) to achieve comparable results with shallow ones (*e.g.* 2-layer). Moreover, NodeNorm helps deep GCNs to outperform shallow ones in cases where often deep models are required to learn good node representations.

The rest of the paper is organized as follows. In Sec. 2 we review related works of GNNs and research on the performance degradation problem. Then, in Sec. 3, we experimentally show that TRANs contribute significantly to performance degradation and that this is because they cause *variance inflammation*. We also propose a variance-controlling technique to address *variance inflammation* in Sec. 3. Then we conduct experiments to demonstrate it helps resolve performance degradation by addressing *variance inflammation* in Sec. 4. In Sec. 5, we discuss where to put the variance-controlling technique in a GCN model, and also other benefits of these techniques. Sec. 6 concludes the paper.

2 Related Works

2.1 Graph Neural Networks (GNNs)

Graph neural networks (GNNs) are widely applied to learn graph node representations over graph-structured data. Current GNNs are generally built based on a neural message passing framework [29] where one PROP operation and one TRAN operation are performed in each layer. The Graph

Convolutional Network (GCN) is one most representative GNN architecture, which performs a graph convolution in the graph spectral domain per layer, with kernels approximated by a first-order Chebyshev polynomials [19] of the normalized graph Laplacian matrix. GCNs have achieved high performance in the node classification task on various datasets [7], [23], and are widely used as GNN backbones in lots of other graph learning tasks [30], [31], [32]. In addition to GCNs, the GraphSage architecture [6] and Graph ATtention networks (GATs) [10] are also popular GNNs. They are widely adopted for node classification, achieving comparable performance with GCNs on many benchmark datasets [7], [12]. GraphSage learn node representations by aggregating and transforming information from randomly sampled neighbors. This sampling-based aggregation strategy makes GraphSage very suitable for learning from large-scale graph datasets [6], [33]. GAT performs a learnable and flexible PROP by the attention mechanism [2] in each layer.

2.2 Performance degradation problem of GNNs

It has been observed that existing GNN architectures tend to suffer performance degradation as their model depth increases [9], [15]. This problem is first observed in GCNs [9], and later Chen *et al*. [15] find other popular GNN architectures such as GraphSage [6] and GAT [10] also suffer such performance degradation.

Most studies on this problem are based on GCNs. Existing works focus on how PROPs in GCNs affect node representations and cause performance degradation. Li *et al*. [14] show that a PROP in GCN is essentially a Laplacian smoothing operation, and therefore PROPs push the node embeddings to be indistinguishable in deep GCNs, causing performance degradation, which is termed *oversmoothing*. To reduce oversmoothing, Chen *et al*. [15] introduce an additional loss to discourage similarity among distant nodes; DropEdge [17] randomly removes edges from the graph during the training process; PairNorm [18] fixes the total pairwise feature distances across different layers. In addition to oversmoothing, gradient vanishing is also identified by Li *et al*. [16] to be a reason for performance degradation, and is also widely believed to be caused by the smoothing effect of PROPs [16], [17], [27], [28].

Compared with PROPs, little attention has been paid to effects of TRANs in this problem. Though Klicpera *et al*. [34] and Zhao *et al*. [18] claim that TRANs also make a reason for performance degradation, they do not investigate and justify it, and their focus is still on PROPs. To the best of our knowledge, [35] is the only work that studies the role of TRANs in the performance degradation problem. However, their theoretical analysis is performed under the assumption that the input graph is sufficiently dense and the model depth goes to infinity, which is inapplicable to real world datasets or practical GCN models. In addition, their analysis is about how TRANs and PROPs collectively lead to oversmoothing, rather than revealing whether and how TRANs themselves contribute to performance degradation. Unlike [18], [34], [35], our focus is placed on the role of TRANs in performance degradation of GCNs.

In addition to the works investigating and addressing performance degradation in existing GNNs, there are also some works trying to design new GNN architectures that can naturally go deep without incurring severe performance drop. A representative architectures among them is JKNet [11]. However, as shown in [17], though a 8-layer JKNet can compete with a 2-layer one, JKNet still suffers performance degradation when the model goes very deep (*e.g*. 32- or 64-layer). Concurrent to our work, GCNII [27] and GEN [28] show good performance even when they go very deep. GCNII addresses oversmoothing via initial residual connections and identity mappings, while GEN overcomes performance degradation with the help of several techniques including generalized aggregation functions, message normalization and layer normalization [22]. This line of works, *i.e*. JKNet, GCNII and GEN, are orthogonal and complementary to ours, as our work aims to better understand and resolve performance degradation based on the representative GCN architecture. Furthermore, as we will show in Sec. 4.4, applying our proposed variance-controlling technique to these architectures can also improve their performance.

3 Understanding and Resolving Performance Degradation of GCNs

Neural networks usually perform better with increasing depth [1], [36]. However, GCNs perform increasingly poorly with larger model depth. Existing works mainly study how PROPs contribute to this problem, and pay little attention to the role of TRANs, the other important operator that constitutes a GCN layer.

In this section, we study this problem from a new perspective. We start with ablative experiments to investigate the roles of PROPs and TRANs in causing performance degradation. Based on the attained observations, we identify that the *variance inflammation* issue introduced by TRANs is the critical contributing factor. Finally we develop a variance-controlling technique to alleviate this problem and improve performance of deep GCNs.

3.1 Preliminaries

We first introduce the preliminaries on graph convolution network (GCN) models. 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 $D = \mathrm{diag}(A\mathbf{1})$, where $\mathbf{1}$ is an n-dimensional all-ones column vector. An L-layer GCN model [9] is composed of L cascaded feed-forward Graph Convolution (GC) layers. Formally, the l-th GC layer can be represented as

$$H^{(l)} = \mathrm{ReLU}(\hat{A}H^{(l-1)}W^{(l)}), \tag{1}$$

where $H^{(l)} \in \mathbb{R}^{n\times d_l}$ and $W^{(l)} \in \mathbb{R}^{d_l \times d_{l+1}}$ denote the node feature matrix and the learnable weight matrix of this layer. The i-th row $(i \in \{1,\cdots,n\})$ of $H^{(l)}$, denoted as $\mathbf{h}_i^{\top(l)}$, represents the input embedding vector of node i of the l-th layer. In this formula, $\hat{A}$ is the re-normalized adjacency matrix defined as $\hat{A} = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}$, where $\tilde{A} = A + I$ and $\tilde{D} = \mathrm{diag}(\tilde{A}\mathbf{1})$. According to Eqn. (1), one GCN layer

performs the following two basic operations: the PROPagation operation (PROP) and the TRANsformation operation (TRAN) [4], [37]:

$$\begin{aligned} \bar{H}^{(l-1)} &= \hat{A}H^{(l-1)} \ \texttt{(PROP)}, \\ H^{(l)} &= \mathrm{ReLU}(\bar{H}^{(l-1)}W^{(l)}) \ \texttt{(TRAN)}. \end{aligned} \tag{2}$$

The first operation propagates and aggregates information from the 1-hop neighbors of each single node, while the latter transforms the aggregated embeddings via a linear transformation followed by a non-linear ReLU [38] activation function.

A GCN model is built by stacking multiple GCN layers as above. Generally, with more GCN layers, the node feature information can be propagated to farther nodes [4], [29]. This is helpful for aggregating information from distant nodes, and hence improves the performance of GCNs [16], [18], [27]. However, some practical observations [14], [15], [16], [17], [18] are contradictory to this intuition—stacking more layers would incur severe performance drop for GCNs. This is known as the performance degradation problem for deep GCNs. To address this problem, previous studies focus on PROPs, which are believed to cause the oversmoothing issue, *i.e.*, node features in deep GCNs being pushed by PROPs to be indistinguishable from each other. Meanwhile, the role of TRANs is largely overlooked in the previous studies. However, TRANs are also critical for the performance of GCNs since they transform the aggregated information progressively and hence greatly influence the learned node representations.

3.2 Transformation operations contribute significantly to performance degradation

To investigate the role of TRANs in performance degradation, we need to exclude the influence of PROPs. To this end, we disentangle the two operations and build two variants of GCNs: 1) T-GCNs that only perform TRAN in each hidden layer; 2) P-GCNs that only perform PROP in each hidden layer.

Formally, denote a GC layer as $\mathrm{GC}(\cdot)$ and a TRAN as $\mathrm{T}(\cdot)$, and let $X \in \mathbb{R}^{n\times d}$ be the input feature matrix. Then, an L-layer T-GCN and P-GCN can be represented as

$$\begin{aligned} H^{(L)} &= \mathrm{GC}(\underbrace{\mathrm{T}\circ\cdots\circ\mathrm{T}}_{L-2}(\mathrm{GC}(X))) \quad \text{(T-GCN)}, \\ H^{(L)} &= \mathrm{GC}(\hat{A}^{L-2}\mathrm{GC}(X)) \quad \text{(P-GCN)}. \end{aligned} \tag{3}$$

Note the parameters of the two models in different layers are not shared.

We train three models, *i.e.*, vanilla GCN, P-GCN and T-GCN, of different depths on Cora dataset [23], and plot test accuracy in Fig. 1. Here the model depth varies from 2 to 64. It can be seen that T-GCN suffers even more severe performance degradation than GCNs. Specifically, the accuracy drops to 0.4 for 64-layer T-GCN, compared with the accuracy of 0.6 of 64-layer vanilla GCN. By contrast, although the performance of P-GCNs also drops when the the model goes very deep (*e.g.* 64 layers), the accuracy only drops to 0.72. Such observations deviate from the conventional belief [14], [15] that more PROPs will hurt the model performance more severely. Instead, we find that stacking more TRANs introduces larger performance drop and TRANs contribute more than PROPs in this study.

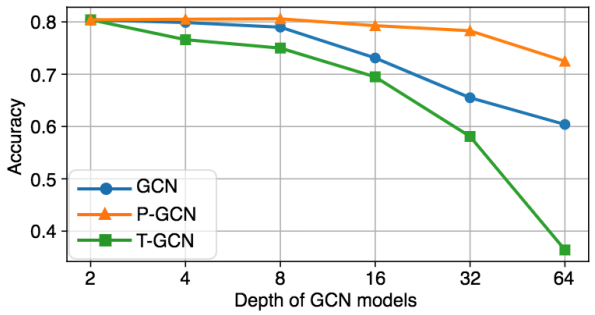


Fig. 1: Performance degradation of GCNs, P-GCNs and T-GCNs. A T-GCN performs only TRANs in hidden layers, while a P-GCN performs only PROPs in hidden layers. Results are obtained on the test split of Cora. Please refer to Supplementary Materials for results on other datasets.

3.3 Transformation operations cause variance inflammation

We then investigate why stacking more TRANs would hurt the performance of deep models. We experimentally examine their effects on node representations, and find that TRANs tend to amplify the node-wise feature variance. As a result, as a GCN model becomes deeper, it contains more TRANs and hence its output node-wise feature variance becomes increasingly large in general. We refer to this phenomenon as *variance inflammation*. Here the node-wise feature variance refers to the variance of each node's features. Formally, the feature variance of node i in the l-th layer is

$$\mathrm{var}_i^{(l)} = \frac{1}{d_l}\sum_{j=1}^{d_l}(h_{ij}^{(l)} - \mu_i^{(l)})^2, \tag{4}$$

where h_{ij} is the the j-th feature of node i, $\mu_i^{(l)} = \frac{1}{d_l}\sum_{j=1}^{d_l} h_{ij}^{(l)}$ is the mean of the features, and d_l denotes the feature dimension.

We plot how $\mathrm{var}_i^{(l)}$ changes with the layer index l in a 64-layer T-GCN in Fig. 2 (a). Results in 64-layer P-GCN are also included for comparison. As we can see, within a 64-layer T-GCN, the node-wise feature variance in general rises drastically (note that the y-axis is shown in log scale). By contrast, $\mathrm{var}_i^{(l)}$ in P-GCN does not show an increasing trend with larger l. This observation demonstrates that TRANs tend to amplify node-wise feature variance.

We then plot the node-wise feature variance of the last layer of GCNs with different depths, *i.e.*, $\mathrm{var}_i^{(L)}$ with different L, in Fig. 2 (b). We can see *variance inflammation* from the drastic rise in $\log(\mathrm{var}_i^{(L)})$ from $L = 2$ to $L = 64$, *i.e.* such amplification of node feature variance leading to *variance inflammation* in GCNs.

Moreover, we find that nodes with large feature variance are difficult to classify. We observe this by sorting all the nodes in the graph based on their node-wise feature variance of the last layer, *i.e.*, $\mathrm{var}_i^{(64)}$ of a 64-layer GCN, and partitioning the sorted nodes evenly into 5 bins: $S_1, \cdots, S_5$.

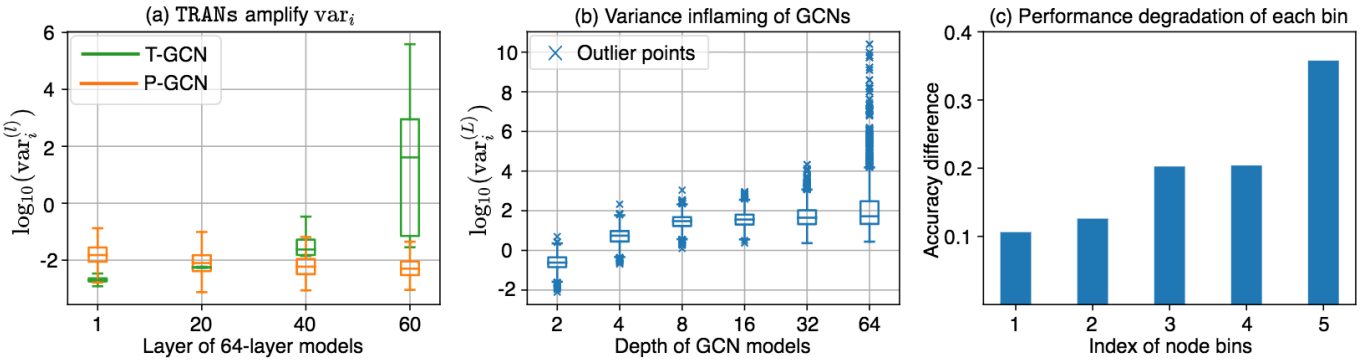


Fig. 2: (a) Node-wise feature variance, *i.e.*, $\mathrm{var}_i^{(l)}$ of all nodes in different layers ($l = 1, 20, 40, 60$) of 64-layer T-GCN and P-GCN models. Results are shown in log scale with a base of 10. Note that the two models are both 64-layer models. Outlier points are not shown for better comparison of the two models. (b) Node-wise feature variance of representations of the last layer, *i.e.*, $\mathrm{var}_i^{(L)}$ (in log scale with a base of 10) for $L = 2, 4, 68, 16, 32, 64$ of all nodes. Note that models in this sub-figure are GCNs of different model depths. (c) Performance difference between a 2-layer and a 64-layer model for each node bin. From S_1 to S_5, $\mathrm{var}_i^{(L)}$ increases. The results are obtained from Cora. Please refer to Supplementary Materials for results on other datasets.

For each bin, we illustrate the performance degradation by calculating the accuracy difference between this 64-layer model and a 2-layer model. The results are summarized in Fig. 2 (c). We observe that the 64-layer model has relatively larger variance and performs significantly worse than the 2-layer one.

Motivated by these findings, we hypothesize that mitigating *variance inflammation*, *i.e.*, preventing var_i from being too large, can help mitigate the performance degradation of deep GCNs.

3.4 Techniques to mitigate variance inflammation

To reduce *variance inflammation*, we propose a technique that scales each node's feature vector by the p-th root of the standard deviation of its features, with $p \geqslant 1$. As the operation is applied in a node-wise manner, we term it *Node Normalization* (NodeNorm), formally expressed as

$$\mathrm{NodeNorm}(\mathbf{h}_i) = \frac{\mathbf{h}_i}{(\sigma_i)^{\frac{1}{p}}}, \tag{5}$$

where $\sigma_i = \sqrt{\mathrm{var}_i}$ is the standard deviation of $\mathbf{h}_i$, and var_i is the variance defined in Eqn. (4). Here we omit the layer index l for clarity. In the following part, we use $\mathrm{NodeNorm}_p$ to denote NodeNorm with a specific p.

After normalized with $\mathrm{NodeNorm}_p$, the node i then has a standard deviation of $\sigma_i^{(1-\frac{1}{p})}$, which is smaller than σ_i if $\sigma_i > 1$. Therefore, a smaller p controls *variance inflammation* more strictly. We can adjust the value of p so that NodeNorm can collaborate well with GCNs in different scenarios. Specifically, when $p = 1$, the variance for all nodes is normalized to be 1.

Moreover, we note that the existing Layer Normalization (LayerNorm) [22] also performs a node-wise variance-scaling operation. We then also investigate whether (and why) LayerNorm is able to address performance degradation in GCNs. Formally, the formulation of LayerNorm is as below:

$$\mathrm{LayerNorm}(\mathbf{h}_i) = \boldsymbol{\alpha} \odot \frac{\mathbf{h}_i - \mu_i}{\sigma_i} + \boldsymbol{\beta}, \tag{6}$$

where $\odot$ denotes an element-wise multiplication. In Eqn. (6), $\boldsymbol{\alpha} = (\alpha_1, \cdots \alpha_d)$ and $\boldsymbol{\beta} = (\beta_1, \cdots \beta_d)$ are learnable parameters, where d denotes the dimension of $\mathbf{h}$. We can see that a LayerNorm consists of three operations: variance-scaling, mean-subtraction, and feature-wise linear transformation ($\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are the slopes and biases). In particular, the variance-scaling in LayerNorm is essentially our $\mathrm{NodeNorm}_1$.

One may argue that LayerNorm does not naturally mitigates *variance inflammation*, due to the linear transformations. However, we empirically find that LayerNorm operations in deep GCNs are trained to effectively reduce *variance inflammation* (see Sec. 4.1.2).

4 Experiments

In this section, we first evaluate the effectiveness of variance-controlling techniques (*i.e.*, NodeNorm and LayerNorm) on alleviating performance degradation of GCNs on benchmark datasets in Sec. 4.1. Then, in Sec. 4.2, we validate whether the proposed NodeNorm can help deep GCNs to outperform shallow ones in cases requiring deep models. Then, we compare the proposed NodeNorm with the existing best methods for addressing performance degradation of GCNs in Sec. 4.3. Finally, in Sec. 4.4, we apply variance-controlling techniques (NodeNorm and LayerNorm) to other GNN architectures to study their effects upon the performance of different GNNs.

4.1 Evaluating effects of proposed NodeNorm on performance degradation of GCNs

To justify whether alleviating *variance inflammation* can help resolve performance degradation of GCNs, we evaluate the proposed NodeNorm on 6 benchmark datasets. We also experiment with LayerNorm to further validate our claim since LayerNorm takes effects on relieving performance drop of deep GCNs because it actually reduces *variance inflammation* (as explained in Sec. 3.4).

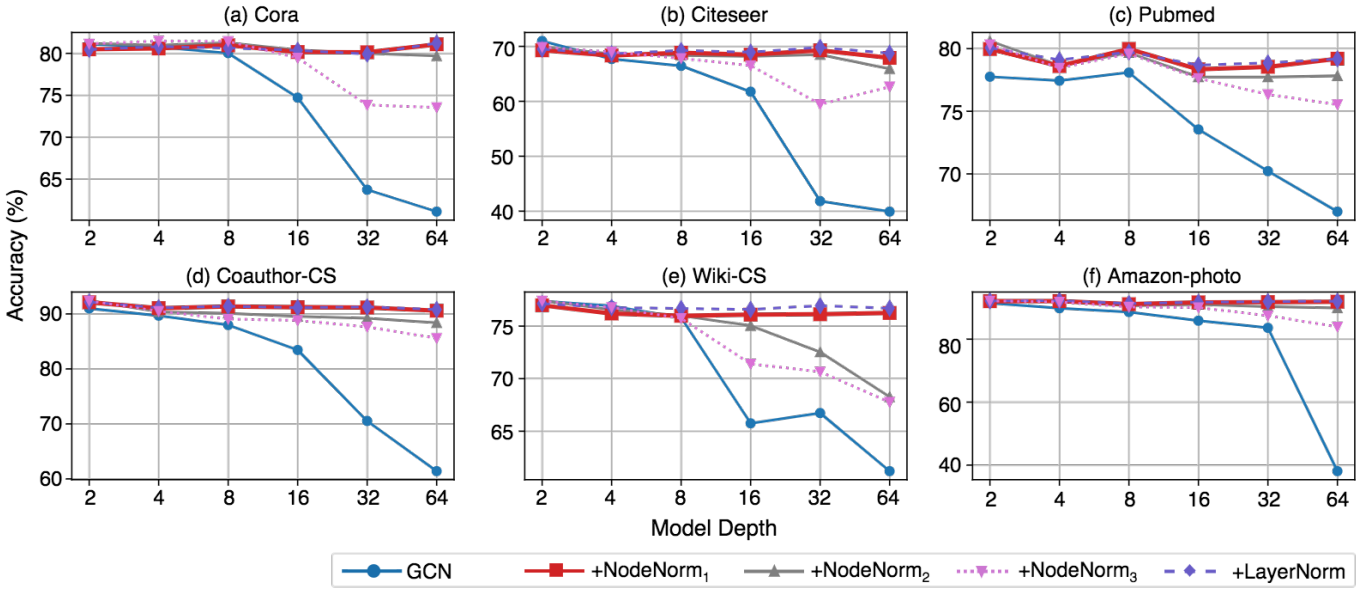


Fig. 3: Classification accuracy w.r.t. model depth. It can be seen all variance-controlling techniques mitigate performance degradation. Our $\mathrm{NodeNorm}_1$ enables deep models to compete with shallow ones by strictly controlling node-wise variance, *i.e.*, ensuring all var_i to be 1, though its operation is very simple, *i.e.* normalizing each node' features with its own standard deviation. Please refer to Supplementary Materials for numerical results, including standard deviation.

TABLE 1: Statistics of datasets used in this section. The task is multi-class classification on all the datasets except Ogbn-proteins dataset (marked with †) where multi-label classification is performed, *i.e.*, 112 independent binary classification tasks.

Datasets	#Nodes	#Edges	#Node features	#Edge features	#Classes
Cora	2,708	5,278	1,433	-	7
Citeseer	3,327	4,551	3,703	-	6
Pubmed	19,717	44,324	500	-	3
Coauthor-CS	18,333	81,894	6,805	-	15
Amazon-photo	7650	119,081	767	-	10
Wiki-CS	11,701	216,123	300	-	10
US-election12	3,234	12,717	6	-	2
US-election16	3,234	12,717	6	-	2
Ogbn-proteins	132,534	39,561,252	-	8	112†
Cornell	183	295	1,703	-	5
Texas	183	309	1,703	-	5
Wisconsin	251	499	1,703	-	5

4.1.1 Experiment settings

We run experiments on 6 node classification datasets of various graph types: three benchmark citation datasets [23] Cora, Citeseer and Pubmed, a co-authorship dataset Coauthor-CS [12], a web-page dataset Wiki-CS [24], and a product co-purchasing dataset Amazon-photo [12]. Please refer to Tab. 1 for statistics of these datasets.

For the three citation networks Cora, Citeseer and Pubmed, we follow the widely adopted 20-label-per-class setting [9], [10], [11], [15], namely 20 labeled training nodes per class, 500 validation nodes, 1,000 test nodes. To make the results more reliable, we run each experiment with 10 different random splits of the dataset and report the mean result, as suggested by [12]. For Coauthor-CS, we also run experiments with 10 splits that are randomly generated based on splitting rules in [39]. For Wiki-CS, we use the 20 splits provided in [24]. For Amazon-photo, following [12] we also adopt the same split setting as the three citation networks. The evaluation metric for these datasets is classification accuracy.

We run experiments of GCNs with {2,4,8,16,32,64} layers. We add residual connections [1], [14] in each layer to avoid training difficulty caused by gradient vanishing. For $\mathrm{NodeNorm}$, we run experiments with $p = 1, 2, 3$, since we find that $p \geqslant 4$ does not help much.

4.1.2 Results

As shown in Fig. 3, compared with the baseline models (vanilla GCNs), models with variance-controlling techniques suffer much less accuracy drop when they get deep. The results well demonstrate that mitigating *variance inflammation* indeed helps address performance degradation.

This is further justified by comparing $\mathrm{NodeNorm}_p$ with varying p. For deep models (*e.g.* a model with 32 or 64 layers), as p decreases from 3 to 1, the model performance increases. The comparison shows that techniques that control *variance inflammation* more strictly (with a smaller p) can address performance degradation more effectively for very deep models. In some scenarios where less strict constraint on variance is desired, a larger p brings better performance. Please refer to Supplementary Materials for details.

In particular, we find that the most strict variance-controlling technique in our evaluation (*i.e.*, $\mathrm{NodeNorm}_1$) can effectively resolve performance degradation, and is among the techniques that perform the best on all datasets for all model depths. Additionally, $\mathrm{NodeNorm}_1$ is also the most efficient among the techniques, because it simply normalizes each node's features with its standard deviation. In comparison, $\mathrm{NodeNorm}_2$ or $\mathrm{NodeNorm}_3$ needs extra calculation for the 2nd or 3rd root of the node features' standard deviation, and LayerNorm performs two more steps than $\mathrm{NodeNorm}_1$ (see Sec. 3.4).

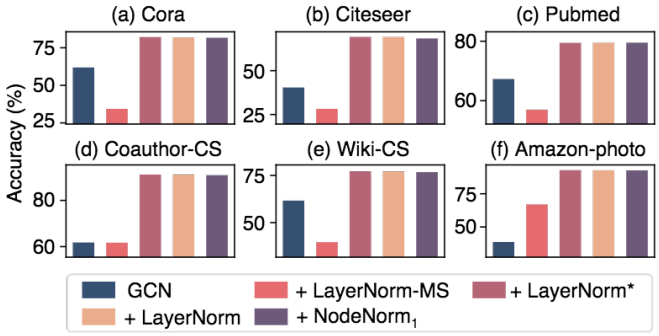


Fig. 4: Classification accuracy of 64-layer GCNs with LayerNorm-MS and LayerNorm*. We also include results of the baseline GCNs, GCNs with $\mathrm{NodeNorm}_1$ or LayerNorm for clearer comparison.

We can also see from Fig. 3 that LayerNorm achieves comparable results with $\mathrm{NodeNorm}_1$ in most of the experiments. However, since LayerNorm performs three operations, more investigation is needed to demonstrate reducing *variance inflammation* is the key to the effectiveness of LayerNorm in resolving performance degradation of deep GCNs. We then ablatively study the effects of the three kinds of operations in LayerNorm on addressing performance degradation. Specifically, we study two variants of LayerNorm:

$$\mathrm{LayerNorm}^*(\mathbf{h}_i) = \frac{\mathbf{h}_i - \mu_i}{\sigma_i}, \tag{7}$$

$$\text{LayerNorm-MS}(\mathbf{h}_i) = \mathbf{h}_i - \mu_i. \tag{8}$$

The LayerNorm* variant does not include linear transformations, while the LayerNorm-MS variant performs only Mean-Subtraction (MS).

We conduct experiments with the two variants of 64 layers, with the same settings as in Sec. 4.1.1, and show results in Fig. 4. We can see that GCNs with LayerNorm*, LayerNorm or $\mathrm{NodeNorm}_1$ perform comparatively, while those with LayerNorm-MS perform significantly worse (on 4 datasets even worse than baseline GCNs). Note that $\mathrm{NodeNorm}_1$ is equivalent to the variance-scaling step in LayerNorm. Moreover, we also find that the linear transformation is trained to approximate identity mappings, which will be elaborated in Supplementary Materials. The above observations show that linear transformation and the mean-subtraction step are not critical for improving deep GCNs performance; instead, variance-scaling is the step that really works. This further demonstrates that reducing *variance inflammation* is the key to addressing performance degradation.

4.2 Evaluating effects of proposed $\mathrm{NodeNorm}$ in cases requiring deep models

Above we show variance-controlling techniques can successfully address performance degradation of GCNs on benchmark datasets under commonly adopted settings. However, in these experiments deep models do not significantly outperform shallow ones. This is because shallow models are sufficient to learn good node representations in these settings, as discussed in [18]. To further demonstrate the benefit of mitigating *variance inflammation*, we evaluate variance-controlling techniques in three exemplar cases where often deep models are desired.

4.2.1 Experiment settings

We first briefly introduce the exemplar cases:

1) **Citation networks with missing features** [18]. In [18], when some input node features are missing in citation graphs, deep models achieve better performance than shallow ones. Here shallow models are not sufficient to learn good node representations because nodes would benefit from a larger neighbourhood to recover effective feature representation.
2) **Citation graphs with low label rate** [25]. Deeper models achieve better performance than shallow ones on Cora, Citeseer and Pubmed at low training label rate. As explained in [25], when training label rate is low, more layers would be needed to reach the supervision information far away.
3) **US election datasets** [26]. US-elect12 and US-elect16 datasets are geographical graphs induced from statistics of United States (US) election of year 2012 and year 2016. Nodes represent US counties, and edges connect nodes whose corresponding counties are geographically bordering. Node features are demographic statistics such as income, education, population. The graph structures of the two datasets are exactly the same. Deep model significantly outperform shallow ones on them, possibly because their graph has a diameter of 69, which is notably larger than that of the commonly used datasets (*e.g.* about 20). In addition, the average shortest path length between node pairs in the two datasets is around 26, over 4 times larger than that of citation networks. Given such a large graph diameter and a long average distance among nodes, deep models would be desired to fully propagate information among nodes and learn good representations.

For the missing-feature case, we follow [18] to run experiments on Cora, Citeseer and Pubmed with the 100%-missing and 80%-missing setting. Dataset splits are the same as the 20-label-per-class setting. For the case of low label rate citation graphs, we run experiments on Cora, Citeseer and Pubmed with the 5-label-per-class setting (label rates for three datasets are 1.29%, 0.90%, and 0.08%) and 2-label-per-class setting (label rates for three datasets are 0.52%, 0.36%, and 0.03%). The sizes of validation and test sets are 500 and 1,000, as in the commonly adopted 20-label-per-class setting. For the US election datasets, we randomly split the nodes into train/val/test by 60%/20%/20% [26], and follow [13] to conduct a binary node classification task.

We run experiments with GCNs of {2,4,8,16,32,64} layers. We take $\mathrm{NodeNorm}_1$ as an example of the variance-controlling techniques, whose effectiveness is validated in Sec. 4.1. We run each experiment with 10 random dataset splits and report best average accuracy and corresponding model depth. Please refer to Supplementary Materials for standard deviation of results.

TABLE 2: Classification accuracy on Citation networks with missing features. In parentheses is the number of layers of the model. Please refer to Supplementary Materials for standard deviation.

Dataset	Missing rate	Method	
		GCN	+NodeNorm$_1$
Cora	100	0.7034 (8)	**0.7207 (64)**
	80	0.7270 (8)	**0.7739 (64)**
Citeseer	100	0.4429 (8)	**0.4861 (32)**
	80	0.4838 (4)	**0.5508 (32)**
Pubmed	100	0.4652 (16)	**0.5751 (16)**
	80	0.7125 (4)	**0.7509 (8)**

TABLE 3: Classification accuracy on Citation networks with low label rate.

Dataset	#Labels per class	Method	
		GCN	+NodeNorm$_1$
Cora	5	0.7209 (2)	**0.7395 (8)**
	2	0.6319 (4)	**0.6420 (16)**
Citeseer	5	0.6193 (4)	**0.6294 (32)**
	2	0.5277 (2)	**0.5516 (16)**
Pubmed	5	0.7272 (4)	**0.7591 (64)**
	2	0.6491 (4)	**0.6813 (64)**

TABLE 4: Classification accuracy on US election datasets.

	GCN	+NodeNorm$_1$
USelect-12	0.8296 (2)	**0.8677 (32)**
USelect-16	0.8840 (2)	**0.9017 (32)**

Note that though the first two cases, *i.e*. citation graphs with missing features and low label rate, are proposed in [18] and [25] respectively, we do not compare with their reported results here, because our goal is to check whether variance-controlling techniques make deeper GCNs outperform shallow ones. Moreover, [25] does not focus on deepening GCNs, and their method cannot solve performance degradation when the model gets deep (*e.g*. more than 10 layers), as stated in their paper. We compare with [18] using a commonly used setting in Sec. 4.3, and leave the comparison in the missing-feature case in Supplementary Materials.

4.2.2 Results

Tab. 2, Tab. 3 and Tab. 4 summarize the results of the experiments above. As can be seen, best performance of NodeNorm$_1$-augmented GCNs is achieved by deep models (*e.g*. 64-layer and 32-layer), and is higher than best performance of vanilla GCNs, which is generally obtained by shallow models (*e.g*. 2-layer and 4-layer).

Though both vanilla GCNs and NodeNorm$_1$-augmented GCNs are theoretically capable of aggregating information from distant nodes when their model depth increases, vanilla GCNs do not perform better as they become deeper. This is because the performance degradation issue offsets the benefit of aggregating distant node information. By contrast, deep NodeNorm$_1$-augmented GCNs successfully outperform their shallow counterparts because NodeNorm$_1$ resolves the performance degradation, which further demonstrates the benefits of alleviating *variance inflammation*.

TABLE 5: Classification performance of NodeNorm$_1$, PairNorm and Dropedge on a widely used split of Cora, Citeseer and Pubmed. Results for DropEdge are from their GitHub repository. For PairNorm, 2-layer results are reported in their paper, and we reproduce other results using their reported settings.

Dataset	Method	Model depth		
		2	32	64
Cora	NodeNorm$_1$	**0.830**	**0.829**	**0.837**
	PairNorm	0.783	0.759	0.778
	DropEdge	0.828	0.811	0.789
Citeseer	NodeNorm$_1$	**0.729**	**0.724**	**0.731**
	PairNorm	0.648	0.615	0.614
	DropEdge	0.723	0.700	0.651
Pubmed	NodeNorm$_1$	**0.807**	**0.808**	**0.804**
	PairNorm	0.756	0.768	0.737
	DropEdge	0.796	0.782	0.769

4.3 Comparing NodeNorm with best methods for training deep GCNs

In this subsection, we compare NodeNorm$_1$ with previous generic methods that can address performance degradation of GCNs and other GNN architectures. More specifically, we compare with PairNorm [18] and Dropedge [17] which are proven most effective. For fair comparison, we follow [17], [18] to run experiments on Cora, Citeseer and Pubmed in the 20-label-per-class semi-supervised classification setting, with a widely used standard split [9], [10], [15].

Tab. 5 summarizes the results. We can see though Dropedge and PairNorm alleviate performance degradation to some extent, deep models with these methods still underperform their shallow counterparts. In comparison, our NodeNorm$_1$ successfully addresses the performance degradation, enabling GCNs of 32 or 64 layers to achieve comparable results with 2-layer GCNs.

It is worth noting that we do not compare with the methods [40], [41], [42] that achieve State-Of-The-Art (SOTA) performance on datasets we use, because their methods are orthogonal to ours. Their focus is not improving GNN when they get deeper and their major contribution is their proposed *new GNN architectures*. By contrast, in this work we aim to understand and alleviate *the performance degradation problem* of GCNs and extend GCNs and other existing popular GNN architectures to deep models. Based on such expectations we develop NodeNorm, which can be easily plugged in existing GNN architectures.

4.4 Applying variance-controlling to other GNN architectures

We apply NodeNorm and LayerNorm to other GNN architectures to study whether variance-controlling also works on improving their performance. At below we first experiment on GNNs that also suffer performance degradation when the model gets deep [6], [10], and then on those that can go deep without performance drop [27], [28].

We first apply the variance-controlling techniques (*i.e*. NodeNorm and LayerNorm) to other popular GNN architectures that also suffer from performance degradation, including GAT [10] and GraphSage [6], which achieve high

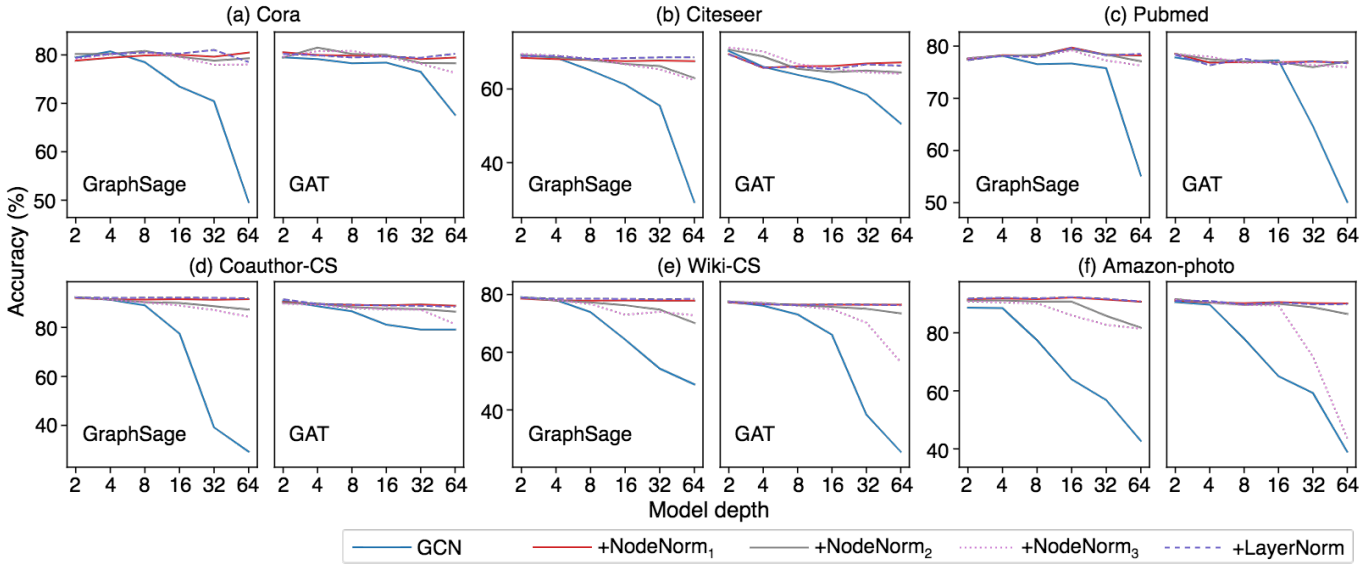


Fig. 5: Classification accuracy w.r.t. model depth of GAT and GraphSage.

performance in the node classification task. We conduct experiments using the same setting as in Sec. 4.1.1. As shown in Fig. 5, the variance-controlling techniques generalize well to these architectures, mitigating their performance degradation effectively. In particular, NodeNorm_1 enables deep GAT or GraphSage models to compete with their shallow counterparts. The results well demonstrate that controlling variance also addresses performance degradation for other GNN architectures in addition to GCNs.

There are also works obtaining deep GNNs by designing *new* GNN architectures that can naturally go deep without suffering severe performance degradation, which are orthogonal to our design in this work. We also include them in experiments to further show the effectiveness of our proposed variance controlling techniques. In particular, we compare our NodeNorm with GEN [28] and GCNII [27], which are among the best new GNN architectures that do not suffer declined performance as the model grows deep. We run experiments with the two models of $\{2, 4, 8, 16, 32, 64\}$ layers. For GEN, we follow [28] to run experiments on Ogbn-proteins [7], each with 5 random seeds, since the dataset split in [7] is practically meaningful and random split may not make sense anymore. For GCNII, we follow [27] to conduct experiments on web-page networks [43]: Cornell, Texas and Wisconsin, where each experiment is run with 10 different dataset splits. In the web-page networks, nodes and edges represent web pages and hyperlinks respectively. Please refer to Tab. 1 for their statistics. We follow experiment settings in GitHub repositories of the two works, and do not tune hyperparameters or use training tricks like one-hot-node-encoding in [28].

We take NodeNorm_1 as an example of variance-controlling techniques, as it is simple yet effective as shown in Sec. 4.1. In addition, we also run experiments of GEN with LayerNorm, as LayerNorm is adopted in [28]. Though GEN [28] applies LayerNorm to training deep GNNs, they do not investigate whether and how LayerNorm affects deep GNNs performance. In comparison, our experiments in Sec. 4.1 show that LayerNorm benefits deep GCNs by mitigating *variance inflammation*, thus addressing performance degradation.

TABLE 6: Best AUC-ROC and corresponding model depth on Obgn-proteins of GENs with or without applying variance-controlling. Results are obtained with the PyTorch Geometric library [44] implementation of GENs, provided by [28]. Experiments are run with settings in the GitHub repository of [28]. Note, results of GEN are those of vanilla GEN, while results reported in [28] are given by GEN with LayerNorm. See Supplementary Materials for standard deviation.

Method	AUC-ROC
GEN	0.7936 (64)
GEN+LayerNorm	0.8224 (64)
GEN+NodeNorm_1	**0.8226** (64)

TABLE 7: Best accuracy and corresponding model depth on web-pages datasets of GCNIIs with or without NodeNorm_1. Results are obtained with codes from their GitHub repository. See Supplementary Materials for standard deviation.

Method	Dataset		
	Cornell	Texas	Wisconsin
GCNII	0.7496 (16)	0.6946 (32)	0.7412 (16)
GCNII+NodeNorm_1	**0.8054** (16)	**0.7892** (32)	**0.8314** (16)

As shown in Tab. 6 and Tab. 7, the variance-controlling techniques improve the performance of GEN and GCNII. Notably, NodeNorm_1 improves the performance of GCNII by a margin of 10% on Texas and Wisconsin datasets. We emphasize again that GEN and GCNII are orthogonal to our work—we focus on understanding and addressing performance degradation of GCNs (and other popular GNNs), and claim that reducing *variance inflammation* helps mitigate this issue, while their contributions are the proposed architectures, *i.e.*, GEN or GCNII.

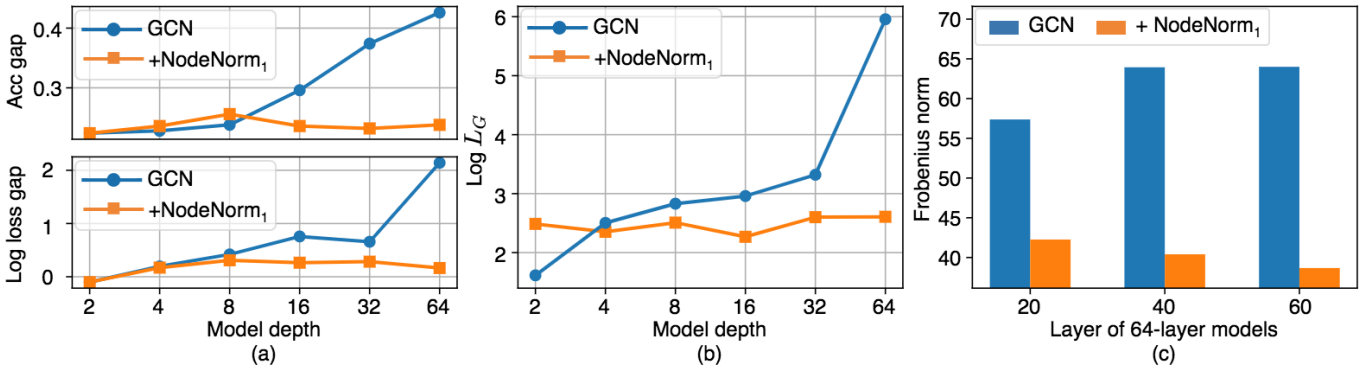


Fig. 6: (a) Accuracy gap (upper) and loss gap (lower) of GCNs between training and validation set. Note, loss gap is shown in a log scale with base of 10. (b) Graph Lipschitz constant (L_G) of models with different depths. Results are in log scale with base of 10. (c) Frobenius norm of feature correlation matrices of different hidden layers of 64-layer models. Results are on Cora. See Supplementary Materials for results on Citeseer and Pubmed.

TABLE 8: Classification accuracy of 64-layer GCNs with NodeNorm_1-In, NodeNorm_1 LayerNorm-In, or LayerNorm on Cora, Citeseer and Pubmed. We also include results of NodeNorm_1 and LayerNorm for clearer comparison, which are from Tab. 3.

	Cora	Citeseer	Pubmed
NodeNorm_1-In	0.8104	0.6901	0.7913
NodeNorm_1	0.8108	0.6790	0.7920
LayerNorm-In	0.8036	0.6901	0.7945
LayerNorm	0.8122	0.6875	0.7918

5 More Insights on Mitigating Variance Inflammation

5.1 Where to put variance-controlling techniques

Normalization operations [21], [22] are generally put after convolution layers in the literature of GNNs. Following this convention, in experiments in Sec. 4, we put variance-controlling techniques, *i.e.*, NodeNorm and LayerNorm, after each Graph Convolutional (GC) layer (after the TRAN of each layer), and show that this effectively addresses *variance inflammation* and performance degradation.

We note that placing variance-controlling techniques before the TRAN of each GC layer can also prevent node-wise variance from being amplified layer by layer, which also mitigates *variance inflammation*. This yields a GC layer with a variance-controlling technique placed *inside* it:

$$H^{(l)} = \text{ReLU}(f(\hat{A}H^{(l-1)})W^{(l)}), \tag{9}$$

where $f(\cdot)$ can be a NodeNorm or LayerNorm operation. We refer to this implementation of NodeNorm_p and LayerNorm as NodeNorm_p-In and LayerNorm-In respectively ("In" for "Inside GC").

To justify that this implementation also helps alleviate performance degradation, we conduct experiments of GCNs of layers {2, 4, 8, 16, 32, 64} equipped with NodeNorm_p-In and LayerNorm-In on Cora, Citeseer and Pubmed with the 20-label-per-class setting introduced in Sec. 4.1.1. We take NodeNorm_1 as an example of NodeNorm_p because it outperforms NodeNorm_p with other values of p (see Sec.4.1).

As shown in Tab. 8, NodeNorm_1-In and LayerNorm-In achieve comparable results with NodeNorm_1 and LayerNorm respectively. This shows this inside convolution implementation of variance-controlling techniques also effectively resolves performance degradation.

We believe the above results provide valuable insights to the community of GNNs: normalization operations do not necessarily need to be placed after convolution layers as in CNNs.

5.2 Other benefits of mitigating variance inflammation

As pointed out in [18], [34], deep GCNs tend to suffer more overfitting than shallow ones, which is also a factor contributing to performance degradation. Existing works generally tackle overfitting by regularization techniques such as Dropout [45]. We find that mitigating *variance inflammation* also helps reduce overfitting. We take NodeNorm_1 as an example to illustrate this.

We first show overfitting in GCNs via loss gap and accuracy gap between train split and validation split. We run experiments on Cora, Citeseer and Pubmed, and show the results in Fig. 6. Comparing the curves in Fig. 6 (a), we can see NodeNorm_1 effectively mitigates overfitting in deep GCNs.

We then experimentally investigate how NodeNorm_1 achieves this from two perspectives: characteristics of models, and characteristics of hidden features. From the perspective of model characteristics, we define the Graph Lipschitz Constant, denoted as L_G, for GNNs to measure their smoothness w.r.t. node features, which is inspired by observations in [46], [47], [48] that the networks enforced to have lower Lipschitz constant[1] tend to have better generalization ability and suffer less overfitting. We investigate how NodeNorm_1 affects model smoothness.

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 L_G is

$$L_G = \max_{i,j \in V} \frac{\|f(\mathbf{x}_i; G, \mathbf{w}) - f(\mathbf{x}_j; G, \mathbf{w})\|}{\|\mathbf{x}_j - \mathbf{x}_j\|}. \tag{10}$$

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

From Eqn. (10), the models with smaller L_G are less sensitive to disturbances in node features. We compare the L_G values of GCNs with or without $\mathrm{NodeNorm}_1$, as shown in Fig. 6 (b). We can see $\mathrm{NodeNorm}_1$ effectively reduces L_G and acts as an implicit regularizer in the training process of deep models.

From the perspective of characteristics of hidden features, inspired by observing that overfitting can be alleviated by decorrelating hidden features [49], we investigate whether $\mathrm{NodeNorm}_1$ can reduce feature correlation. We compare the correlation among hidden features of 64-layer GCNs with or without $\mathrm{NodeNorm}_1$. Specifically, we compute the Frobenius norm of feature correlation matrices of different layers of the two models, as shown in Fig. 6 (c). It can be seen the model trained with $\mathrm{NodeNorm}_1$ has less correlated features than others, showing $\mathrm{NodeNorm}_1$ can effectively reduce overfitting.

6 Conclusion

In this paper, we investigate the performance degradation problem of GCNs by focusing on effects of TRANsformation operations (`TRANs`). We find `TRANs` contribute significantly to the declined performance, providing a new understanding for the community. Furthermore, we find `TRANs` tend to amplify the node-wise feature variance of node representations, and then as GCNs get deeper, its node-wise feature variance becomes increasingly larger. We also find deep GCNs perform significantly worse than shallow ones on nodes with relatively large feature variance. We thus hypothesize and experimentally justify that mitigating *variance inflammation* effectively addresses performance degradation of GCNs. In particular, a simple variance-controlling technique termed $\mathrm{NodeNorm}_1$, which normalizes each node's hidden features with its own standard deviation, is developed. We experimentally prove it can enable deep GCNs (*e.g.* 64-layer) to compete with and even outperform shallow ones (*e.g.* 2-layer). $\mathrm{NodeNorm}_1$ outperforms existing best methods on addressing performance degradation of GCNs, and can generalize to other GNN architectures.

Acknowledgement

Jiashi Feng was partially supported by AI.SG R-263-000-D97-490, NUS ECRA R-263-000-C87-133 and MOE Tier-II R-263-000-D17-112. Yanfei Dong and Huan Xu were supported by Alibaba Group through Alibaba Innovative Research (AIR) Program. Wee Sun Lee was supported by the National Research Foundation Singapore under its AI Singapore Program (Award Number: AISGRP- 2018-006)

References

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[2] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[3] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner *et al.*, "Relational inductive biases, deep learning, and graph networks," *arXiv preprint arXiv:1806.01261*, 2018.

[4] J. Zhou, G. Cui, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, "Graph neural networks: A review of methods and applications," *arXiv preprint arXiv:1812.08434*, 2018.

[5] Z. Zhang, P. Cui, and W. Zhu, "Deep learning on graphs: A survey," *IEEE Transactions on Knowledge and Data Engineering*, 2020.

[6] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in neural information processing systems*, 2017, pp. 1024–1034.

[7] W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, and J. Leskovec, "Open graph benchmark: Datasets for machine learning on graphs," *arXiv preprint arXiv:2005.00687*, 2020.

[8] W. L. Hamilton, R. Ying, and J. Leskovec, "Representation learning on graphs: Methods and applications," *arXiv preprint arXiv:1709.05584*, 2017.

[9] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[10] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.

[11] K. Xu, C. Li, Y. Tian, T. Sonobe, K.-i. Kawarabayashi, and S. Jegelka, "Representation learning on graphs with jumping knowledge networks," *arXiv preprint arXiv:1806.03536*, 2018.

[12] O. Shchur, M. Mumme, A. Bojchevski, and S. Günnemann, "Pitfalls of graph neural network evaluation," *arXiv preprint arXiv:1811.05868*, 2018.

[13] Q. Huang, H. He, A. Singh, S.-N. Lim, and A. R. Benson, "Combining label propagation and simple models out-performs graph neural networks," *arXiv preprint arXiv:2010.13993*, 2020.

[14] Q. Li, Z. Han, and X.-M. Wu, "Deeper insights into graph convolutional networks for semi-supervised learning," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[15] D. Chen, Y. Lin, W. Li, P. Li, J. Zhou, and X. Sun, "Measuring and relieving the over-smoothing problem for graph neural networks from the topological view," *arXiv preprint arXiv:1909.03211*, 2019.

[16] G. Li, M. Muller, A. Thabet, and B. Ghanem, "Deepgcns: Can gcns go as deep as cnns?" in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 9267–9276.

[17] Y. Rong, W. Huang, T. Xu, and J. Huang, "Dropedge: Towards deep graph convolutional networks on node classification," in *International Conference on Learning Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=Hkx1qkrKPr

[18] L. Zhao and L. Akoglu, "Pairnorm: Tackling oversmoothing in gnns," *arXiv preprint arXiv:1909.12223*, 2019.

[19] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Advances in neural information processing systems*, 2016, pp. 3844–3852.

[20] G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter, "Self-normalizing neural networks," in *Advances in neural information processing systems*, 2017, pp. 971–980.

[21] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[22] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *arXiv preprint arXiv:1607.06450*, 2016.

[23] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad, "Collective classification in network data," *AI magazine*, vol. 29, no. 3, pp. 93–93, 2008.

[24] P. Mernyei and C. Cangea, "Wiki-cs: A wikipedia-based benchmark for graph neural networks," *arXiv preprint arXiv:2007.02901*, 2020.

[25] K. Sun, Z. Zhu, and Z. Lin, "Multi-stage self-supervised learning for graph convolutional networks," *arXiv preprint arXiv:1902.11038*, 2019.

[26] J. Jia and A. R. Benson, "Residual correlation in graph neural network regression," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 588–598.

[27] M. Chen, Z. Wei, Z. Huang, B. Ding, and Y. Li, "Simple and deep graph convolutional networks," *arXiv preprint arXiv:2007.02133*, 2020.

[28] G. Li, C. Xiong, A. Thabet, and B. Ghanem, "Deepergcn: All you need to train deeper gcns," *arXiv preprint arXiv:2006.07739*, 2020.

[29] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *Proceedings*

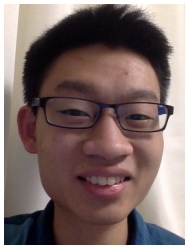
of the 34th International Conference on Machine Learning-Volume 70. JMLR. org, 2017, pp. 1263–1272.
[30] T. N. Kipf and M. Welling, "Variational graph auto-encoders," *arXiv preprint arXiv:1611.07308*, 2016.
[31] J. You, B. Liu, Z. Ying, V. Pande, and J. Leskovec, "Graph convolutional policy network for goal-directed molecular graph generation," in *Advances in neural information processing systems*, 2018, pp. 6410–6421.
[32] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" *arXiv preprint arXiv:1810.00826*, 2018.
[33] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, "Graph convolutional neural networks for web-scale recommender systems," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018, pp. 974–983.
[34] J. Klicpera, A. Bojchevski, and S. Günnemann, "Predict then propagate: Graph neural networks meet personalized pagerank," *arXiv preprint arXiv:1810.05997*, 2018.
[35] K. Oono and T. Suzuki, "On asymptotic behaviors of graph cnns from dynamical systems perspective," *arXiv preprint arXiv:1905.10947*, 2019.
[36] M. Telgarsky, "Benefits of depth in neural networks," *arXiv preprint arXiv:1602.04485*, 2016.
[37] F. Wu, T. Zhang, A. H. d. Souza Jr, C. Fifty, T. Yu, and K. Q. Weinberger, "Simplifying graph convolutional networks," *arXiv preprint arXiv:1902.07153*, 2019.
[38] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *ICML*, 2010.
[39] V. Verma, M. Qu, A. Lamb, Y. Bengio, J. Kannala, and J. Tang, "Graphmix: Regularized training of graph neural networks for semi-supervised learning," *arXiv preprint arXiv:1909.11715*, 2019.
[40] ——, "Graphmix: Improved training of graph neural networks for semi-supervised learning."
[41] W. Feng, J. Zhang, Y. Dong, Y. Han, H. Luan, Q. Xu, Q. Yang, E. Kharlamov, and J. Tang, "Graph random neural networks for semi-supervised learning on graphs," *Advances in Neural Information Processing Systems*, vol. 33, 2020.
[42] J. Ma, W. Tang, J. Zhu, and Q. Mei, "A flexible generative framework for graph-based semi-supervised learning," in *Advances in Neural Information Processing Systems*, 2019, pp. 3281–3290.
[43] H. Pei, B. Wei, K. C.-C. Chang, Y. Lei, and B. Yang, "Geom-gcn: Geometric graph convolutional networks," in *International Conference on Learning Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=S1e2agrFvS
[44] M. Fey and J. E. Lenssen, "Fast graph representation learning with PyTorch Geometric," in *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.
[45] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.
[46] P. L. Bartlett, D. J. Foster, and M. J. Telgarsky, "Spectrally-normalized margin bounds for neural networks," in *Advances in Neural Information Processing Systems*, 2017, pp. 6240–6249.
[47] H. Gouk, E. Frank, B. Pfahringer, and M. Cree, "Regularisation of neural networks by enforcing lipschitz continuity," *arXiv preprint arXiv:1804.04368*, 2018.
[48] D. Zou, R. Balan, and M. Singh, "On lipschitz bounds of general convolutional neural networks," *IEEE Transactions on Information Theory*, 2019.
[49] M. Cogswell, F. Ahmed, R. Girshick, L. Zitnick, and D. Batra, "Reducing overfitting in deep networks by decorrelating representations," *arXiv preprint arXiv:1511.06068*, 2015.

Kuangqi Zhou is a Ph.D. student in the Department of Electrical and Computer Engineering in National University of Singapore. He received the B.E. degree in School of the Gifted Young from University of Science and Technology of China. His research interests include deep learning and computer vision. Recently he works on understanding and improving Graph Neural Networks on different tasks. Moreover, he has served as external reviewer for IEEE Transactions on Neural Networks and Learning Systems (TNNLS) and IEEE Transactions on Circuits and Systems for Video Technology (TCSVT).

Yanfei Dong is a Ph.D. student under the joint talent program with Alibaba DAMO Academy and Department of Computer Science, National University of Singapore. She obtained her B.Comp and M.Comp degree in computer science and B.Sc degree in applied mathematics in National University of Singapore. Her current research interests include machine learning and deep learning, with a focus on graph-structured data.

Kaixin Wang is a Ph.D. student in NUS Graduate School for Integrative Sciences and Engineering. He received bachelor degree in School of Information Management from Nanjing University in 2018. His research interests include deep learning and reinforcement learning. Specifically, he works on improving generalization in reinforcement learning and learning adaptable representation. Moreover, he has served as external reviewer for IEEE Transactions on Image Processing (TIP).

Wee Sun Lee is a professor in the Department of Computer Science, National University of Singapore. He obtained his Ph.D. from the Australian National University in 1996. He has been a research fellow at the Australian Defense Force Academy, a fellow of the Singapore-MIT Alliance, and a visiting scientist at MIT. His research interests include machine learning, planning under uncertainty, and approximate inference. He has been an area chair for machine learning and AI conferences such as the Neural Information Processing Systems (NeurIPS), the international Conference on Machine Learning (ICML), the AAAI Conference on Artificial Intelligence (AAAI), and the International Joint Conference on Artificial Intelligence (IJCAI). He was a program, conference and journal track co-chair for the Asian Conference on Machine Learning (ACML), and he is currently the co-chair of the steering committee of ACML.



Bryan Hooi is an assistant professor in the Computer Science Department, School of Computing, and the Institute of Data Science in National University of Singapore. He received his PhD degree in Machine Learning from Carnegie Mellon University, USA in 2019. His research interests include machine learning on graph-structured data, robustness and novelty detection, and spatiotemporal data mining. His work aims to develop efficient and practical data mining algorithms, with applications including fraud detection, online commerce, and automatic monitoring of medical, industrial, weather and environmental sensor data.

Huan Xu has been with Alibaba Group since 2018. He obtained his Ph.D. degree in ECE from McGill University in 2009. He has been an associate professor in the Department of Industrial and Systems Engineering of National University of Singapore since 2016. His current research interests focus on learning and decision-making in large-scale complex systems. Specifically, he is interested in machine learning, high-dimensional statistics, robust and adaptable optimization, robust sequential decision making, and applications to large-scale systems. He is currently an associate editor of IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI) and Computational Management Science.

Jiashi Feng (Member, IEEE) is currently an Assistant Professor with the Department of Electrical and Computer Engineering with the National University of Singapore. He received the B.Eng. degree from the University of Science and Technology, China, in 2007, and the Ph.D. degree from the National University of Singapore in 2014. He was a Postdoctoral Researcher with the University of California from 2014 to 2015. He His current research interests include machine learning and computer vision techniques for large-scale data analysis. Specifically, he has done works in object recognition, deep learning, machine learning, high-dimensional statistics, and big data analysis.

SUPPLEMENTARY MATERIALS

In Supplementary Materials A (SM. A), we show results of analytical experiments in Sec. 3 on other datasets. Then in SM. B we give numerical results of figures in Sec.4, and some supplementary experimental results. Next, we implementation details of our experiments in Sec. 4 in SM. C.

A Analytical experiments on other datasets

In Sec.3, we use Cora dataset to illustrate our observations. Here we show corresponding results on Citeseer, Pubmed, Coauthor-CS, Wiki-CS and Amazon-photo.

A.1 TRANs contribute significantly to performance degradation

Fig. 7 shows performance degradation of GCNs, P-GCNs and T-GCNs on the aforementioned datasets.

A.2 TRANs tend to amplify node-wise variance

Fig. 8 shows that TRANs tend to amplify node-wise variance var_i.

A.3 Variance inflaming

Fig.9 shows *variance inflammation* of GCNs on these datasets.

B Supplementary results for Sec. 4

B.1 Numerical results

Tab. 9 shows numerical results (including standard deviation) for Fig 3. Tab. 10 to Tab. 14 show standard deviation for results in Tab. 2, Tab. 3, Tab. 4, Tab. 6 and Tab. 7 respectively.

B.2 Further Comparisons with best competing methods

We conduct further comparisons with current best competing methods (i.e. PairNorm and Dropedge) in cases where deeper models are desired. To be more specific, we compare our method with them in two scenarios. The first one is a low training label rate setting when only 2 labels per class are available, and the second one is when all non-training features are missing. As Tab. 10 shows, models augmented with $\mathrm{NodeNorm}_1$ are superior to all other methods in most of the scenarios.

B.3 Visualizing learned parameters in LayerNorm

As mentioned in Sec. 4.1.2, the feature-wise linear transformation in LayerNorm are trained to approximate identity mappings. Here we elaborate this observation by visualizing the learned parameters in LayerNorm. Specifically, we visualize entries of α and β of the LayerNorm in the different hidden layers layer of a 64-layer trained GCN model in Fig. 10. We can see that entries of α are close 1, while thoes of β are close to. Consequently, the feature-wise linear transformation approximates an identical mapping.

B.4 Scenarios where larger p in $\mathrm{NodeNorm}_p$ is desired

From experimental results in Sec. 4, we can see that $\mathrm{NodeNorm}_1$ brings more performance improvement of deep GCNs (*e.g.* 64-layer) than $\mathrm{NodeNorm}_2$ and $\mathrm{NodeNorm}_3$. This is because deep GCNs suffer suffer severe *variance inflammation*, and thus techniques that controlling variance more strictly brings more performance gain. However, in scenarios where *variance inflammation* is less severe, $\mathrm{NodeNorm}_p$ with larger p which is less strict than $\mathrm{NodeNorm}_1$, would be more desired. This is evidenced by Tab. 9: for shallow models (*e.g.* 2-layer, 4-layer, 8-layer), GCNs with $\mathrm{NodeNorm}_3$ and $\mathrm{NodeNorm}_2$ generally achieves better than those with $\mathrm{NodeNorm}_1$. This demonstrates that $\mathrm{NodeNorm}$ is flexible. Indeed, we can control the value of p so that $\mathrm{NodeNorm}_p$ collaborate well with GCNs in different scenarios.

B.5 Supplementary results for Sec. 5.2

In Fig. 11 and Fig. 12, we show results of analytical experiments in Sec. 5.2 on Citeseer and Pubmed respectively.

Implementation details

In Tab. 15 to Tab. 22, we list hyperparameters used in Sec. 4. The hyperparameters are obtained by grid search.

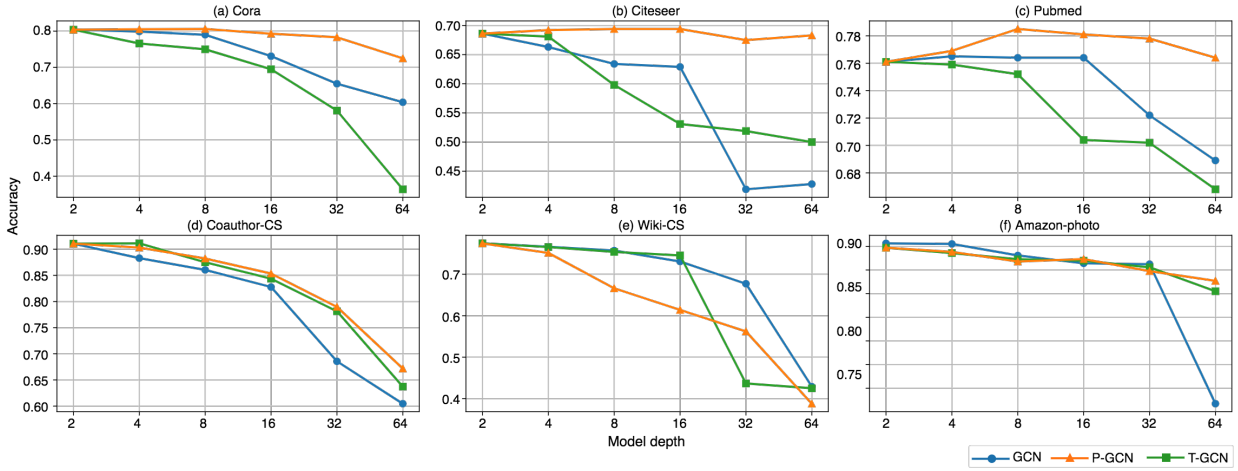


Fig. 7: Classification accuracy w.r.t. model depth.

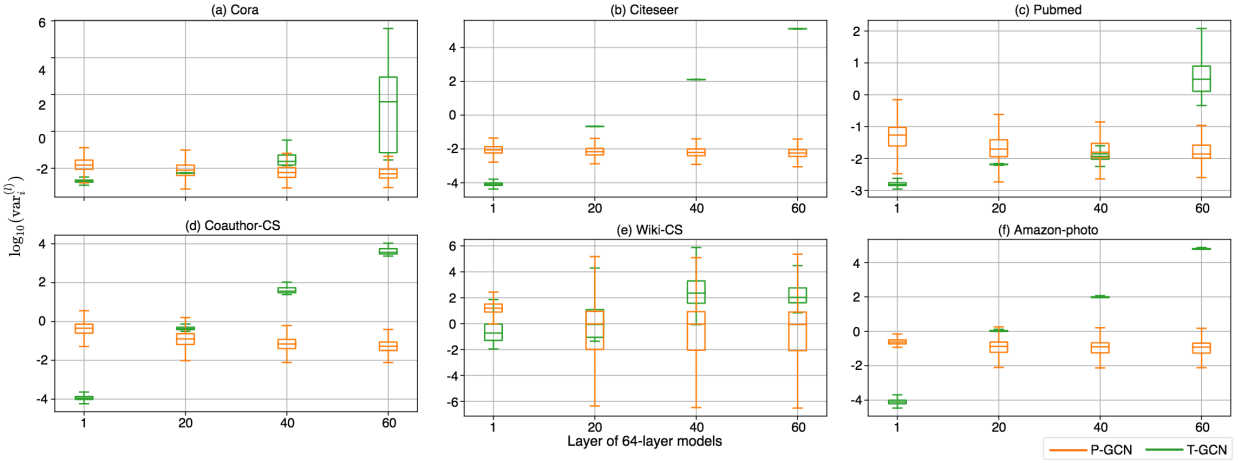


Fig. 8: Node-wise variance, *i.e.*, $\mathrm{var}_i^{(l)}$ of all nodes in different layers ($l = 1, 20, 40, 60$) of 64-layer T-GCN and P-GCN models. Results are shown in log scale with a base of 10.

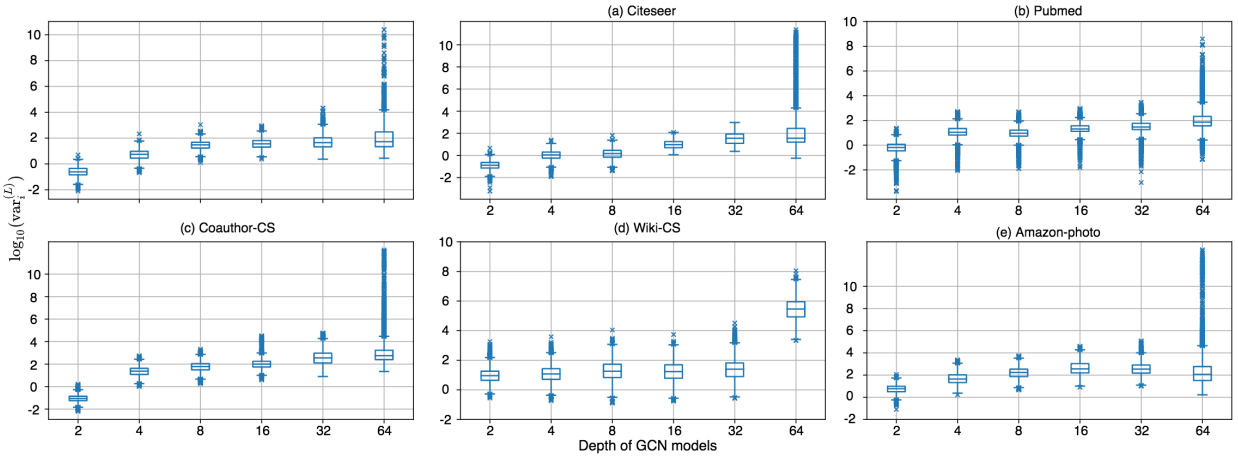


Fig. 9: Node-wise variance of the last layer, *i.e.*, $\mathrm{var}_i^{(L)}$ (in log scale with a base of 10) for $L = 2, 4, 8, 16, 32, 64$ of all nodes.

B Supplementary experimental results

B.1 Numerical results

TABLE 9: Numerical results of Fig. 3.

Method	Dataset	2-Layer	4-Layer	8-Layer	16-Layer	32-Layer	64-Layer
GCN	Cora	81.09±0.85	80.74±1.87	80.03±1.56	74.72±2.63	63.74±1.79	61.13±3.28
	Citeseer	70.97±1.16	67.70±1.49	66.45±1.50	61.75±2.71	41.83±4.53	39.92±2.06
	Pubmed	77.76±2.36	77.44±2.22	78.09±1.53	73.55±4.38	70.22±2.95	66.99±4.24
	Coauthor-CS	91.30±0.65	90.04±0.58	88.69±0.91	84.44±1.36	70.54±5.05	61.41±3.15
	Wiki-CS	76.98±0.65	76.20±0.54	75.99±0.63	76.09±0.72	76.13±0.69	76.25±0.64
	Amazon-photo	91.46±1.07	90.82±0.93	88.62±1.36	85.86±2.17	83.59±2.11	38.08±10.64
+$\mathrm{NodeNorm}_1$	Cora	80.48±1.23	80.58±1.71	80.96±1.62	80.16±1.04	80.13±0.84	81.08±1.87
	Citeseer	69.34±1.02	68.34±2.80	68.85±1.82	68.50±1.83	69.28±1.63	67.90±1.84
	Pubmed	79.97±1.44	78.61±2.19	79.98±0.94	78.35±1.68	78.55±2.14	79.20±2.37
	Coauthor-CS	92.12±0.35	91.06±0.92	91.36±0.37	91.22±0.46	91.17±0.52	90.59±0.81
	Wiki-CS	77.43±0.68	76.56±0.63	75.99±0.62	75.05±0.71	72.53±2.38	68.26±3.15
	Amazon-photo	92.05±1.14	92.07±0.62	91.10±0.84	91.68±0.71	91.76±0.91	91.88±1.31
+$\mathrm{NodeNorm}_2$	Cora	81.17±1.77	80.96±1.86	81.25±1.90	80.41±0.92	80.01±1.44	79.69±2.59
	Citeseer	69.91±2.09	68.66±1.99	68.34±1.97	68.19±2.04	68.48±1.96	65.92±0.66
	Pubmed	80.61±1.50	78.65±1.27	79.67±1.51	77.74±2.55	77.74±1.94	77.83±2.57
	Coauthor-CS	92.43±0.32	90.40±0.99	90.10±0.31	89.56±0.60	89.22±0.71	88.41±0.45
	Wiki-CS	77.40±0.71	76.78±0.52	75.74±0.47	71.41±2.82	70.66±2.38	67.76±1.91
	Amazon-photo	92.17±0.85	91.95±0.64	90.59±1.60	91.00±0.96	90.31±1.05	89.91±1.86
+$\mathrm{NodeNorm}_3$	Cora	81.13±1.12	81.47±1.37	81.43±1.59	79.45±1.80	73.84±3.48	73.55±4.09
	Citeseer	69.86±0.83	68.98±1.49	67.80±1.81	66.52±2.31	59.45±10.36	62.64±3.36
	Pubmed	80.33±1.76	78.46±1.86	79.59±1.48	77.63±2.23	76.34±2.31	75.54±3.71
	Coauthor-CS	92.36±0.40	90.51±0.63	89.06±0.74	88.82±0.56	87.64±0.61	85.59±1.27
	Wiki-CS	77.14±0.62	76.76±0.41	76.68±0.67	76.59±0.62	76.95±0.50	76.70±0.56
	Amazon-photo	92.15±0.94	91.81±0.95	90.21±1.19	90.02±1.65	87.45±2.22	83.91±2.69
+LayerNorm	Cora	80.46±1.10	80.80±1.47	80.63±0.95	80.36±1.54	79.91±1.38	81.22±1.54
	Citeseer	69.49±1.14	68.65±1.79	69.25±2.02	68.94±1.62	69.82±1.77	68.75±1.25
	Pubmed	80.07±1.49	79.10±2.03	79.73±1.19	78.69±1.72	78.86±1.05	79.18±2.02
	Coauthor-CS	92.19±0.65	91.12±0.84	91.40±0.43	91.14±0.45	91.29±0.47	90.80±0.66
	Wiki-CS	77.14±0.62	76.76±0.41	76.68±0.67	76.59±0.62	76.95±0.50	76.70±0.56
	Amazon-photo	92.14±1.04	92.25±0.62	91.30±0.97	91.71±1.28	91.93±0.70	92.05±1.05

TABLE 10: Further comparisons with best competing methods. We conduct experiments on two scenarios where deeper models are desired: 1) when only 2 training labels per class are available, 2) when feature missing rate is 100%.

Dataset	Scenario	Method			
		GCN	+PairNorm	+DropEdge	+$\mathrm{NodeNorm}_1$
Cora	Low label rate	0.6319 (4)	0.5777 (16)	0.6193 (4)	**0.6420 (16)**
	Missing features	0.7034 (8)	0.6847 (64)	**0.7335 (16)**	0.7207 (64)
Citeseer	Low label rate	0.5277 (2)	0.4805 (8)	0.4710 (8)	**0.5516 (16)**
	Missing features	0.4429 (8)	0.4475 (32)	0.4811 (16)	**0.4861 (32)**
Pubmed	Low label rate	0.6491 (4)	0.6525 (32)	0.6557 (4)	**0.6813 (64)**
	Missing features	0.4652 (16)	**0.6683 (32)**	0.4292 (32)	0.5751 (16)

TABLE 11: Standard deviation of results in Tab. 2.

Dataset	Missing rate	Method	
		GCN	+NodeNorm$_1$
Cora	100	0.0235	0.0122
	80	0.0398	0.0183
Citeseer	100	0.0226	0.0234
	80	0.0499	0.0204
Pubmed	100	0.0678	0.0904
	80	0.0308	0.0225

TABLE 12: Standard deviation of results in Tab. 3.

Dataset	#Labels per class	Method	
		GCN	+NodeNorm$_1$
Cora	5	0.0221	0.0178
	2	0.0982	0.0301
Citeseer	5	0.0354	0.0104
	2	0.0695	0.0702
Pubmed	5	0.0282	0.0191
	2	0.0675	0.0350

TABLE 13: Standard deviation of results in Tab. 4.

	GCN	+NodeNorm$_1$
USelect-12	0.0147	0.0099
USelect-16	0.0094	0.0063

TABLE 14: Standard deviation of results in Tab. 6.

Method	AUC-ROC
GEN	0.0086
GEN+LayerNorm	0.0029
GEN+NodeNorm$_1$	0.0035

TABLE 15: Standard deviation of results in Tab. 7.

Method	Dataset		
	Cornell	Texas	Wisconsin
GCNII	0.0514	0.0793	0.0526
GCNII+NodeNorm$_1$	0.0692	0.0637	0.0542

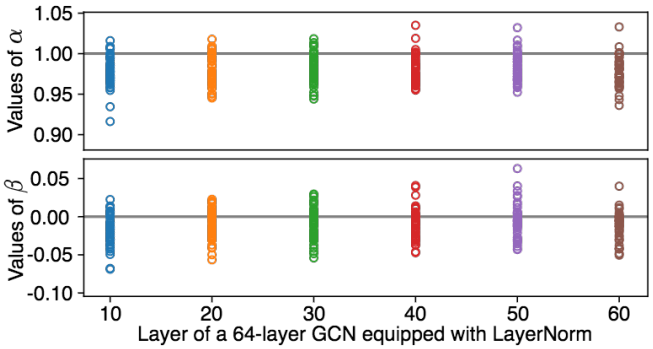


Fig. 10: Visualization of learned parameters in LayerNorm. Each single circlet represents the value of a feature dimension, *i.e.*, an entry of α or β.

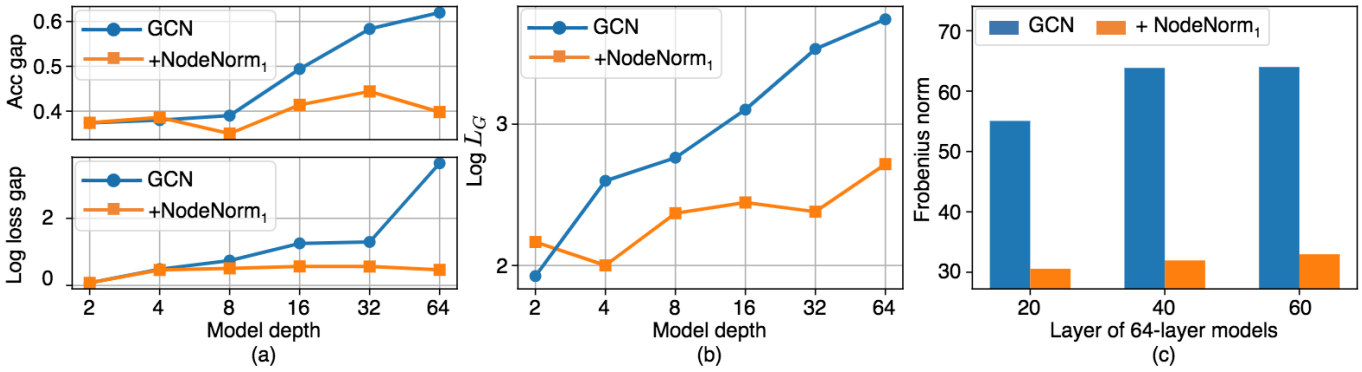


Fig. 11: $\mathrm{NodeNorm}_1$ reduces overfitting. Results are on Citeseer.

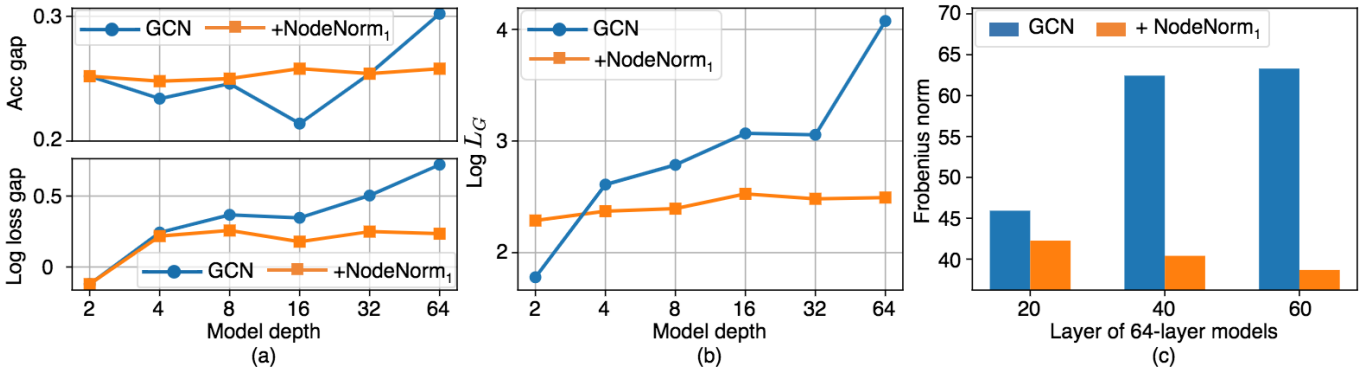


Fig. 12: $\mathrm{NodeNorm}_1$ reduces overfitting. Results are on Pubmed.

TABLE 16: Hyperparameters of Fig. 3 (GCNs with $\mathrm{NodeNorm}_1$)

Dataset	# layers	dropout rate	ℓ_1 weight	weight decay factor	learning rate	# epochs
Cora	2	0.8	0.0005	0.0005	0.005	400
	4	0.6	0.01	0.0005	0.005	400
	8	0.7	0.005	0.001	0.005	400
	16	0.8	0.001	0.001	0.005	400
	32	0.7	0.001	0.0005	0.005	400
	64	0.5	0.0005	0.001	0.005	400
Citeseer	2	0.6	0.01	0.0005	0.005	400
	4	0.6	0.01	0.001	0.005	400
	8	0.5	0.005	0.001	0.005	400
	16	0.6	0.001	0.001	0.005	400
	32	0.6	0.005	0.001	0.005	400
	64	0.6	0.01	0.001	0.005	400
Pubmed	2	0.7	0.005	0.0005	0.005	400
	4	0.6	0.005	0.0005	0.005	400
	8	0.8	0.005	0.001	0.005	400
	16	0.6	0.01	0.001	0.005	400
	32	0.5	0.005	0.0005	0.005	400
	64	0.7	0.005	0.0005	0.005	400
Coauthor-CS	2	0	0.0005	0.001	0.005	400
	4	0.6	0.0001	0.001	0.005	400
	8	0.5	0.0001	0.0005	0.005	400
	16	0.5	0	0.001	0.005	400
	32	0.6	0	0.0005	0.005	400
	64	0.5	0	0.001	0.005	400
Wiki-CS	2	0.3	0	0.0005	0.005	400
	4	0.3	0.0001	0.0005	0.005	400
	8	0.5	0	0.001	0.005	400
	16	0.3	0.0001	0.001	0.005	400
	32	0.3	0	0.001	0.005	400
	64	0.3	0	0.001	0.005	400
Amazon-photo	2	0.5	0.0005	0.0005	0.005	400
	4	0.8	0.001	0.0005	0.005	400
	8	0.8	0.0005	0.001	0.005	400
	16	0.7	0.001	0.001	0.005	400
	32	0.6	0.001	0.0005	0.005	400
	64	0.5	0.0001	0.001	0.005	400

TABLE 17: Hyperparameters of Fig. 3 (GCNs with $\mathrm{NodeNorm}_2$)

Dataset	# layers	dropout rate	ℓ_1 weight	weight decay factor	learning rate	# epochs
Cora	2	0.8	0.0001	0.0005	0.005	400
	4	0.8	0.001	0.001	0.005	400
	8	0.8	0.0005	0.001	0.005	400
	16	0.8	0.001	0.001	0.005	400
	32	0.6	0.005	0.0005	0.005	400
	64	0.5	0.01	0.0005	0.005	400
Citeseer	2	0.8	0.0005	0.001	0.005	400
	4	0.6	0.005	0.0005	0.005	400
	8	0.8	0.001	0.0005	0.005	400
	16	0.8	0.001	0.0005	0.005	400
	32	0.8	0.001	0.001	0.005	400
	64	0.5	0.005	0.0005	0.005	400
Pubmed	2	0.6	0.001	0.0005	0.005	400
	4	0.8	0.0005	0.0005	0.005	400
	8	0.6	0.001	0.0005	0.005	400
	16	0.7	0.005	0.001	0.005	400
	32	0.5	0.01	0.001	0.005	400
	64	0.6	0.005	0.0005	0.005	400
Coauthor-CS	2	0.6	0	0.001	0.005	400
	4	0.5	0.0001	0.0005	0.005	400
	8	0.7	0.0001	0.0005	0.005	400
	16	0.7	0.0001	0.001	0.005	400
	32	0.6	0	0.0005	0.005	400
	64	0.7	0	0.001	0.005	400
Wiki-CS	2	0.6	0	0.0005	0.005	400
	4	0.6	0.0001	0.0005	0.005	400
	8	0.6	0	0.001	0.005	400
	16	0.5	0.0001	0.001	0.005	400
	32	0.3	0.0001	0.0005	0.005	400
	64	0.3	0.005	0.0005	0.005	400
Amazon-photo	2	0.7	0.0005	0.001	0.005	400
	4	0.6	0.001	0.0005	0.005	400
	8	0.5	0.0005	0.0005	0.005	400
	16	0.6	0.001	0.0005	0.005	400
	32	0.6	0.005	0.001	0.005	400
	64	0	0.01	0.001	0.005	400

TABLE 18: Hyperparameters of Fig. 3 (GCNs with $\mathrm{NodeNorm}_3$)

Dataset	# layers	dropout rate	ℓ_1 weight	weight decay factor	learning rate	# epochs
Cora	2	0.8	0.0001	0.0005	0.005	400
	4	0.8	0.001	0.0005	0.005	400
	8	0.8	0.001	0.0005	0.005	400
	16	0.7	0.001	0.001	0.005	400
	32	0	0.005	0.0005	0.005	400
	64	0	0.005	0.0005	0.005	400
Citeseer	2	0.8	0.0001	0.0005	0.005	400
	4	0.8	0.001	0.0005	0.005	400
	8	0.8	0.001	0.001	0.005	400
	16	0.8	0.001	0.0005	0.005	400
	32	0.5	0.001	0.0005	0.005	400
	64	0.5	0.005	0.001	0.005	400
Pubmed	2	0.6	0.0005	0.0005	0.005	400
	4	0.7	0.001	0.0005	0.005	400
	8	0.7	0.0005	0.0005	0.005	400
	16	0.6	0.005	0.0005	0.005	400
	32	0.5	0.001	0.0005	0.005	400
	64	0.6	0.005	0.001	0.005	400
Coauthor-CS	2	0.5	0	0.001	0.005	400
	4	0.5	0.001	0.0005	0.005	400
	8	0.5	0	0.0005	0.005	400
	16	0.8	0	0.001	0.005	400
	32	0.5	0.0005	0.0005	0.005	400
	64	0.5	0.0005	0.001	0.005	400
Wiki-CS	2	0.5	0	0.001	0.005	400
	4	0.6	0	0.001	0.005	400
	8	0.5	0	0.0005	0.005	400
	16	0.3	0.0005	0.001	0.005	400
	32	0.7	0.0001	0.001	0.001	1500
	64	0.6	0	0.0005	0.001	1500
Amazon-photo	2	0.5	0.0005	0.0005	0.005	400
	4	0.5	0.001	0.001	0.005	400
	8	0.6	0.0005	0.001	0.005	400
	16	0.5	0.005	0.001	0.005	400
	32	0	0.01	0.001	0.005	400
	64	0	0.0005	0.0005	0.005	400

TABLE 19: Hyperparameters of Fig. 3 (GCNs with LayerNorm)

Dataset	# layers	dropout rate	ℓ_1 weight	weight decay factor	learning rate	# epochs
Cora	2	0.7	0.001	0.0005	0.005	400
	4	0.8	0.001	0.0005	0.005	400
	8	0.8	0.001	0.001	0.005	400
	16	0.8	0.001	0.001	0.005	400
	32	0.8	0.005	0.001	0.005	400
	64	0.5	0.001	0.001	0.005	400
Citeseer	2	0.6	0.01	0.001	0.005	400
	4	0.6	0.005	0.001	0.005	400
	8	0.5	0.005	0.0005	0.005	400
	16	0.5	0.005	0.0005	0.005	400
	32	0.8	0.0005	0.001	0.005	400
	64	0.7	0.005	0.001	0.005	400
Pubmed	2	0.6	0.01	0.0005	0.005	400
	4	0.8	0.005	0.001	0.005	400
	8	0.7	0.01	0.001	0.005	400
	16	0.8	0.005	0.001	0.005	400
	32	0.7	0.005	0.001	0.005	400
	64	0.7	0.01	0.001	0.005	400
Coauthor-CS	2	0	0.0005	0.001	0.005	400
	4	0.5	0.0001	0.001	0.005	400
	8	0.6	0.0001	0.0005	0.005	400
	16	0.6	0.0001	0.0005	0.005	400
	32	0.6	0.0001	0.0005	0.005	400
	64	0.5	0	0.001	0.005	400
Wiki-CS	2	0.5	0	0.0005	0.005	400
	4	0.7	0	0.0005	0.005	400
	8	0.7	0	0.0005	0.005	400
	16	0.6	0	0.001	0.005	400
	32	0.7	0	0.0005	0.005	400
	64	0.6	0.0001	0.0005	0.005	400
Amazon-photo	2	0.7	0.001	0.001	0.005	400
	4	0.5	0.005	0.0005	0.005	400
	8	0.7	0.001	0.0005	0.005	400
	16	0.5	0.001	0.0005	0.005	400
	32	0.6	0.001	0.0005	0.005	400
	64	0.6	0.0005	0.001	0.005	400

TABLE 20: Hyperparameters of Tab. 2 (GCN with $\mathrm{NodeNorm}_1$ on citation graphs with missing features).

Dataset	Missing rate(%)	dropout rate	ℓ_1 weight	weight decay factor	learning rate	# epochs	Best layer
Cora	100	0.6	0.01	0.001	0.005	1500	64
	80	0.8	0.001	0.0005	0.005	1500	64
Citeseer	100	0.5	0.005	0.0005	0.005	1500	32
	80	0.6	0.0001	0.001	0.005	1500	32
Pubmed	100	0.5	0	0.0005	0.005	1500	16
	80	0.8	0.001	0.001	0.005	1500	8

TABLE 21: Hyperparameters of Tab. 3 (GCN with $\mathrm{NodeNorm}_1$ on citation graphs with low label rate).

Dataset	# labels per class	dropout rate	ℓ_1 weight	weight decay factor	learning rate	# epochs	Best layer
Cora	5	0.5	0.005	0.0005	0.005	400	8
	2	0.6	0.005	0.001	0.005	400	16
Citeseer	5	0.5	0.001	0.001	0.005	400	32
	2	0.7	0.01	0.001	0.005	400	16
Pubmed	5	0.5	0.005	0.001	0.005	400	64
	2	0.5	0.005	0.001	0.005	400	64

TABLE 22: Hyperparameters of Tab. 4 (GCN with $\mathrm{NodeNorm}_1$ on USelect-12 and USelect-16).

Dataset	dropout rate	ℓ_1 weight	weight decay factor	learning rate	# epochs	Best layer
USelect-12	0	0	0.0005	0.01	1500	32
USelect-16	0.6	0.0005	0.001	0.01	1500	32

TABLE 23: Hyperparameters of Tab. 5.

Dataset	# layers	dropout rate	ℓ_1 weight	weight decay factor	learning rate	# epochs
Cora	2	0.7	0.01	0.0005	0.005	400
	4	0.7	0.0005	0.0003	0.005	400
	8	0.7	0.0008	0.0008	0.005	400
	16	0	0.003	0.001	0.005	400
	32	0.4	0.0008	0.0003	0.005	400
	64	0.7	0.008	0.0005	0.005	400
Citeseer	2	0.8	0.001	0.001	0.005	400
	4	0.5	0.003	0.0005	0.005	400
	8	0.9	0.0005	0.0001	0.005	400
	16	0.8	0.001	0.001	0.005	400
	32	0.4	0.001	0.0005	0.005	400
	64	0.6	0.005	0.0003	0.005	400
Pubmed	2	0.4	0.005	0.0001	0.005	400
	4	0.8	0.01	0.001	0.005	400
	8	0.9	0.005	0.0005	0.005	400
	16	0.7	0.01	0.0001	0.005	400
	32	0	0.01	0.0003	0.005	400
	64	0.9	0.003	0.0003	0.005	400