GRAPHNORM: A PRINCIPLED APPROACH TO ACCELERATING GRAPH NEURAL NETWORK TRAINING

Anonymous authors
Paper under double-blind review

ABSTRACT

Normalization plays an important role in the optimization of deep neural networks. While there are standard normalization methods in computer vision and natural language processing, there is limited understanding of how to effectively normalize neural networks for graph representation learning. In this paper, we propose a principled normalization method, Graph Normalization (GraphNorm), where the key idea is to normalize the feature values across all nodes for each individual graph with a learnable shift. Theoretically, we show that GraphNorm serves as a preconditioner that smooths the distribution of the graph aggregation's spectrum, leading to faster optimization. Such an improvement cannot be well obtained if we use currently popular normalization methods, such as BatchNorm, which normalizes the nodes in a batch rather than in individual graphs, due to heavy batch noises. Moreover, we show that for some highly regular graphs, the mean of the feature values contains graph structural information, and directly subtracting the mean may lead to an expressiveness degradation. The learnable shift in GraphNorm enables the model to learn to avoid such degradation for those cases. Empirically, Graph neural networks (GNNs) with GraphNorm converge much faster compared to GNNs with other normalization methods, e.g., BatchNorm. GraphNorm also improves generalization of GNNs, achieving better performance on graph classification benchmarks.

1 INTRODUCTION

Recently, there has been a surge of interest in Graph Neural Networks (GNNs) for learning with graph-structured data (Hamilton et al., 2017; Kipf & Welling, 2017; Velickovic et al., 2018; Xu et al., 2018). GNNs learn node and graph features by following a neighbor aggregation (or message passing) scheme (Gilmer et al., 2017), where node features are recursively aggregated from their neighbours. One major theme of existing works is the design of GNN *architecture* variants, e.g., neighbor aggregation modules, that learn good graph representations (Xu et al., 2019). To that end, many theoretical aspects of GNNs have been studied, including their representation power (Xu et al., 2019), generalization ability (Xu et al., 2020), and infinite-width asymptotic behavior (Du et al., 2019). These theoretical understandings lead to GNN architectures that enjoy good representation power and generalization. However, an important problem remains: the optimization of GNNs is often unstable, and the convergence is slow (Xu et al., 2019). This raises the question:

Can we provably improve the optimization for GNNs?

We give an affirmative answer. Specifically, we study the optimization of GNNs through the lens of *normalization*. Feature normalization is an orthogonal direction to feature aggregation or architecture design, and it has been shown crucial when a neural network gets deeper, wider, and more sophisticated (He et al., 2016). Normalization methods that shift and scale feature values have proven to help the optimization of deep neural networks. Curiously, different domains require specialized normalization methods. In computer vision, batch normalization (Ioffe & Szegedy, 2015) is a standard component. While in natural language processing (NLP), layer normalization (Ba et al., 2016; Xiong et al., 2020) is more popularly used. Empirically, common normalization methods from other domains, e.g., BatchNorm and LayerNorm, often lead to unsatisfactory performance when applied to GNNs. Theoretically, there is limited understanding of what kind of normalization provably helps optimization of GNNs.

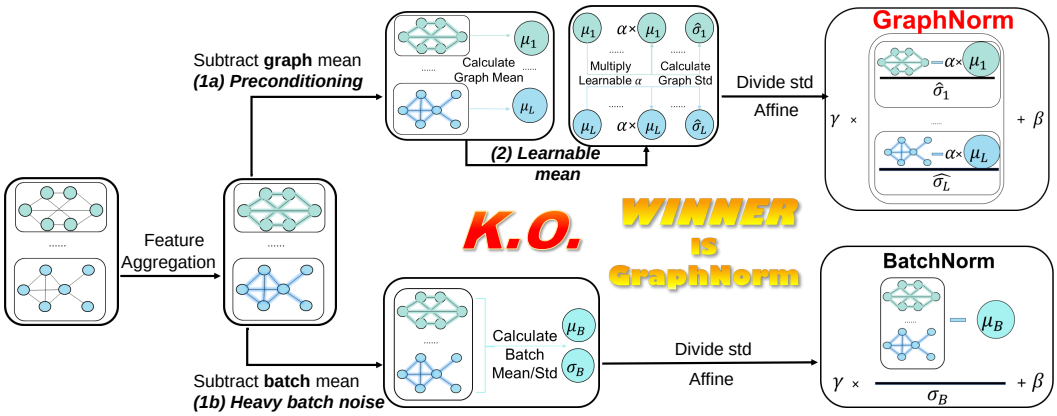

Figure 1: **Overview.** Our proposed GraphNorm is shown along the upper branch. Each step in this branch can boost the performance of GNNs: subtracting graph mean has preconditioning effect; introducing a learnable shift avoids the expressiveness degradation; further scaling to unit norm enjoys "scale-invariant" property (Ioffe & Szegedy, 2015; Hoffer et al., 2018; Arora et al., 2018). In comparison, BatchNorm in the lower branch suffers from heavy batch noise. Overall, GraphNorm significantly surpasses BatchNorm in training speed (Figure 4) and enjoys good generalization performance (Table 1).

In this paper, we propose a theoretically motivated normalization method for GNNs, *Graph Normalization (GraphNorm)*. GraphNorm normalizes the feature values across all nodes in each *individual* graph with a *learnable* shift. We derive GraphNorm from understanding how different components or steps of a normalization method influence the optimization (Figure 1). In particular, we identify the importance of appropriate shift steps for GNNs, an under-explored topic in normalization methods for other domains.

First, we show that applying normalization to each individual graph instead of to the whole mini-batch, is beneficial according to a theoretical understanding of the shift operation. We prove that when applying the normalization to each individual graph, the shift operation (Step 1a in Figure 1) serves as a preconditioner of the graph aggregation operation (Theorem 3.1). Empirically, the preconditioning makes the optimization curvature smoother and makes the training more efficient (Figure 2). Such an improvement cannot be well achieved if we apply the normalization across graphs in a batch, i.e., using BatchNorm. This is because the variation of the batch-level statistics on graph data is much larger (Figure 3). Therefore using noisy statistics during training may make the optimization even more unstable.

Second, we show that the standard shift that simply subtracts the mean of feature values may lead to an expressiveness degradation. Specifically, we prove that for some highly regular graphs, the mean statistics of feature values contains graph structural information which may be crucial for classification (Proposition 4.1 and 4.2). Therefore, directly removing them from the features will consequently hurt the performance (Figure 5). Based on this analysis, we propose the learnable shift (Step 2 in Figure 1) to control how much information in mean statistics to preserve. Together, our proposed GraphNorm normalizes the feature values across nodes in each graph using a learnable shift to avoid expressiveness degradation and enjoy effective optimization.

To validate the effectiveness of GraphNorm, we conduct extensive experiments on eight popular graph classification benchmarks. Empirical results confirm that GraphNorm consistently improves the optimization for GNNs, e.g., convergence speed and stability of training, by a large margin compared to BatchNorm (Figure 4). Furthermore, GraphNorm helps GNNs achieve better generalization performance on most benchmarks (Table 1).

1.1 Related Work

Normalization is important in optimizing deep neural networks, and different normalization techniques have been proposed to improve the training process in different applications (Ioffe & Szegedy,

2015; Ulyanov et al., 2016; Ba et al., 2016; Salimans & Kingma, 2016; Xiong et al., 2020; Salimans et al., 2016; Miyato et al., 2018; Wu & He, 2018). The reason behind the effectiveness of normalization has been intensively studied. Most of the works focus on the "scale-invariant" property: by using a normalization layer right after a linear (or convolutional) layer, the output values will not change when the weights of the parameters in the layer are scaled. Using this property, Kohler et al. (2019) suggests that normalization decouples the optimization of direction and length of the parameters; Ioffe & Szegedy (2015); Hoffer et al. (2018); Arora et al. (2018); Li & Arora (2019) show that the normalization implicitly tunes the learning rate. Santurkar et al. (2018) reveals that normalization smooths the optimization landscape. The "scale-invariant" property is a consequence of the scaling operation in normalization. However, the effect of the shift operation remains highly unexplored.

2 PRELIMINARIES

In this section, we introduce the notations and the basics of GNNs. Let $G = (V, E)$ denote a graph where $V = \{v_1, v_2, \cdots, v_n\}$, n is the number of nodes. Let the feature vector of node v_i be X_i. We denote the adjacency matrix of a graph as $A \in \mathbb{R}^{n\times n}$ with $A_{ij} = 1$ if $(v_i, v_j) \in E$ and 0 otherwise. The degree matrix associated with A is defined as $D = \operatorname{diag}(d_1, d_2, \ldots, d_n)$ where $d_i = \sum_{j=1}^{n} A_{ij}$.

Graph Neural Networks. GNNs use the graph structure and node features to learn the representations of nodes and graphs. Modern GNNs follow a neighborhood aggregation strategy (Sukhbaatar et al., 2016; Kipf & Welling, 2017; Hamilton et al., 2017; Velickovic et al., 2018; Monti et al., 2017), where the representation of a node is iteratively updated by aggregating the representation of its neighbors. To be concrete, we denote $h_i^{(k)}$ as the representation of v_i at the k-th layer and define $h_i^{(0)} = X_i$. We use AGGREGATE to denote the aggregation function in the k-th layer. Formally,

$$h_i^{(k)} = \text{AGGREGATE}^{(k)}\left(h_i^{(k-1)}, \left\{h_j^{(k-1)} : v_j \in \mathcal{N}(v_i)\right\}\right), i = 1, 2, \cdots, n, \tag{1}$$

where $\mathcal{N}(v_i)$ is the set of nodes adjacent to v_i. Different graph neural networks can be obtained by choosing different AGGREGATE functions. We introduce two popularly used networks in detail, Graph Convolutional Networks (GCN) (Kipf & Welling, 2017) and Graph Isomorphism Network (GIN) (Xu et al., 2019). In GCN, the AGGREGATE function is defined as:

$$h_i^{(k)} = \text{ReLU}\left(W^{(k)} \cdot \text{MEAN}\left\{h_j^{(k-1)},\ \forall v_j \in \mathcal{N}(v_i) \cup \{v_i\}\right\}\right), \tag{2}$$

where MEAN denotes the average pooling operation over each feature dimension and $W^{(k)}$ is the parameter matrix in layer k. Taking the matrix form, Eq. 2 can be rewritten as

$$H^{(k)} = \text{ReLU}\left(W^{(k)} H^{(k-1)} Q_{\text{GCN}}\right), \tag{3}$$

where $H^{(k)} = \left[h_1^{(k)}, h_2^{(k)}, \cdots, h_n^{(k)}\right] \in \mathbb{R}^{d^{(k)}\times n}$, $d^{(k)}$ denotes the feature dimension at the k-th layer. $Q_{\text{GCN}} = \hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}$, where $\hat{A} = A + I_n$ and $\hat{D}$ is the degree matrix of $\hat{A}$. I_n is the identity matrix.

In GIN, the AGGREGATE function is defined as

$$h_i^{(k)} = \text{MLP}^{(k)}\left(W^{(k)}\left(\left(1 + \xi^{(k)}\right) \cdot h_i^{(k-1)} + \sum_{v_j \in \mathcal{N}(v_i)} h_j^{(k-1)}\right)\right), \tag{4}$$

which in matrix form is

$$H^{(k)} = \text{MLP}^{(k)}\left(W^{(k)} H^{(k-1)} Q_{\text{GIN}}\right), \tag{5}$$

where $\xi^{(k)}$ is a learnable parameter and $Q_{\text{GIN}} = A + I_n + \xi^{(k)} I_n$.

For a K-layer GNN, the outputs of the final layer, i.e., $h_i^{(K)}$,$i = 1, \cdots, n$, will be used for prediction. For graph classification tasks, we can apply a READOUT function, e.g., summation, to aggregate node features $h_i^{(K)}$ to obtain the entire graph's representation $h_G = \text{READOUT}\left(\left\{h_i^{(K)} \mid v_i \in V\right\}\right)$. A classifier can be applied upon h_G to predict the labels.

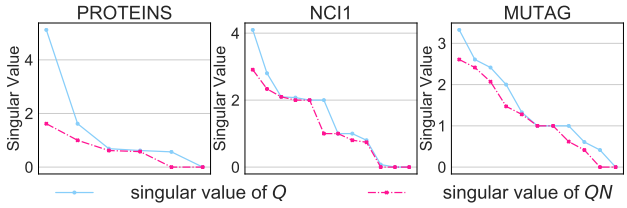

Figure 2: **Singular value distribution** of Q and QN for sampled graphs in different datasets using GIN. More visualizations for different types of graphs can be found in Appendix D.1

Normalization. Generally, given a set of values $\{x_1, x_2, \cdots, x_m\}$, a normalization operation first shifts each x_i by the mean μ, and then scales them down by standard deviation σ: $x_i \to \gamma \frac{x_i - \mu}{\sigma} + \beta$, where γ and β are learnable parameters, $\mu = \frac{1}{m}\sum_{i=1}^{m} x_i$ and $\sigma^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu)^2$. The major difference among different existing normalization methods is which set of feature values the normalization is applied to. For example, in computer vision, BatchNorm (Ioffe & Szegedy, 2015) is the de facto method that normalizes the feature values in the same channel across different samples in the batch. In NLP, LayerNorm (Ba et al., 2016) is more popularly used, which normalizes the feature values at each position in a sequence separately. In GNN literature, as the aggregation function is similar to the convolutional operation, BatchNorm is usually used. Xu et al. (2019) uses BatchNorm in the GIN model, where the BatchNorm is applied to all values in the same feature dimension across *the nodes of all graphs in the batch*.

3 Understanding Normalization for GNNs

In this section, we start from analyzing why and how normalization can help the optimization procedure of GNNs, and then use such a theoretical understanding to develop GraphNorm.

3.1 The Advantage of the Shift in Normalization

As mentioned previously, the scale-invariant property of the normalization has been investigated and considered as one of the ingredients that make the optimization efficient. However, as far as we know, the effectiveness of the shift is not well understood. Compared to the image and sequential data, the graph is explicitly structured, and the neural networks exploit the structural information directly in the aggregation of the neighbours, see Eq. (1). Such uniqueness of GNNs makes it possible to study how the shift operation interplays with the graph data in detail. We first consider the following general GNN structure equipped with a normalization layer:

$$H^{(k)} = F^{(k)}\left(\mathrm{Norm}\left(W^{(k)}H^{(k-1)}Q\right)\right), \tag{6}$$

where $F^{(k)}$ is a function that applies to each node separately, Q is an $n \times n$ matrix representing the neighbor aggregation, and $W^{(k)}$ is the weight/parameter matrix in layer k. We apply the normalization after the linear transformation as in previous works (Ioffe & Szegedy, 2015; Xiong et al., 2020; Xu et al., 2019). We can instantiate Eq. (6) as GCN and GIN, by setting proper $F^{(k)}$ and matrix Q. For example, if we set $F^{(k)}$ to be ReLU and set Q to be Q_{GCN} (Eq. (3)), then Eq. (6) becomes GCN with normalization; Similarly, by setting $F^{(k)}$ to be $\mathrm{MLP}^{(k)}$ and Q to be Q_{GIN} (Eq. (5)), we recover GIN with normalization.

We are interested in how this normalization layer affects the optimization of graph neural networks. Towards this goal, we first consider applying the normalization over *each individual graph* separately. Mathematically, for a graph of n nodes, denote $N = I_n - \frac{1}{n}\mathbf{1}\mathbf{1}^\top$. N is the matrix form of the shift operation, i.e., for any vector $\mathbf{z} = [z_1, z_2, \cdots, z_n]^\top \in \mathbb{R}^n$, $\mathbf{z}^\top N = \mathbf{z}^\top - \left(\frac{1}{n}\sum_{i=1}^{n} z_i\right)\mathbf{1}^\top$. Then the normalization together with the aggregation can be represented as[1]

$$\mathrm{Norm}\left(W^{(k)}H^{(k-1)}Q\right) = S\left(W^{(k)}H^{(k-1)}Q\right)N, \tag{7}$$

[1]Standard normalization has an additional affine operation after shifting and scaling. Here we omit it in Eq. 7 for easier understanding. Note that adding this operation will not affect the theoretical analysis.

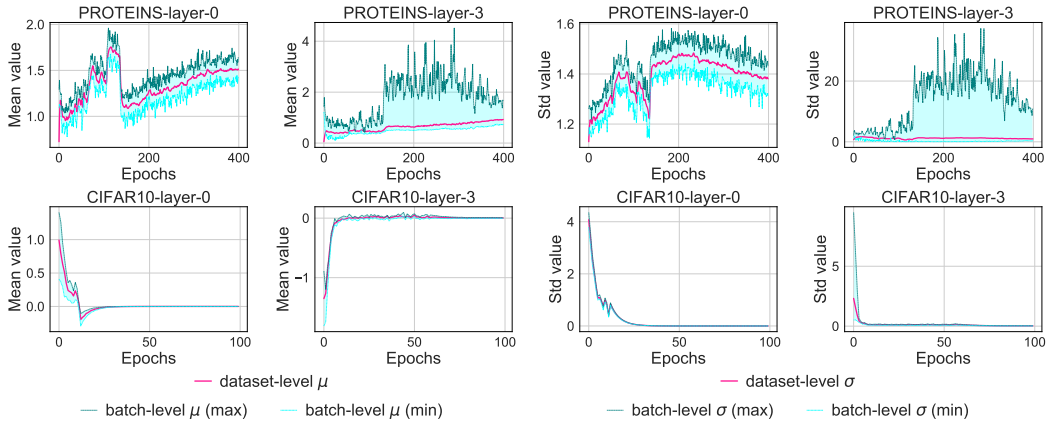

Figure 3: **Batch-level statistics are noisy for GNNs.** We plot the batch-level/dataset-level mean/standard deviation of the first (layer 0) and the last (layer 3) BatchNorm layers of different model checkpoints for a five-layer GIN on PROTEINS and a ResNet18 on CIFAR10. The batch size of all experiments are set to 128. More visualizations for different types of graphs can be found in Appendix D.2.

where $S = \text{diag}\left(\frac{1}{\sigma_1}, \frac{1}{\sigma_2}, \cdots, \frac{1}{\sigma_{d^{(k)}}}\right)$ is the scaling. Each σ_i is the standard deviation of the values of the i-th features among the nodes in the graph we consider. We can see that, in matrix form, shifting feature values on a single graph is equivalent to multiplying N as in Eq. (7). Therefore, we further check how this operation affects optimization. In particular, we examine the singular value distribution of QN. The following theorem shows that QN has a smoother singular value distribution than Q, i.e., N serves as a preconditioner of Q.

Theorem 3.1 (Shift Serves as a Preconditioner of Q)**.** *Let Q, N be defined as in Eq. (7), $0 \leq \lambda_1 \leq \cdots \leq \lambda_n$ be the singular values of Q. We have $\mu_n = 0$ is one of the singular values of QN, and let other singular values of QN be $0 \leq \mu_1 \leq \mu_2 \leq \cdots \leq \mu_{n-1}$. Then we have*

$$\lambda_1 \leq \mu_1 \leq \lambda_2 \leq \cdots \leq \lambda_{n-1} \leq \mu_{n-1} \leq \lambda_n, \tag{8}$$

where $\lambda_i = \mu_i$ or $\lambda_i = \mu_{i-1}$ only if there exists one of the right singular vectors α_i of Q associated with λ_i satisfying $\mathbf{1}^\top \alpha_i = 0$.

Classic wisdom in optimization shows that preconditioning can accelerate the convergence of iterative methods (Axelsson, 1985; Demmel, 1997), and similar ideas are also used to accelerate the optimization of deep neural networks (Duchi et al., 2011; Kingma & Ba, 2015). In the case of optimizing the weight matrix $W^{(k)}$, we can see from Eq. (7) that after applying normalization, the term Q in the gradient of $W^{(k)}$ will become QN which makes the optimization curvature of $W^{(k)}$ smoother, see Appendix A.4 for more discussions.

To check how much the matrix N improves the distribution of the spectrum of matrix Q in real practice, we sample graphs from different datasets for illustration, as showed in Figure 2 (more visualizations for different types of graph can be found in Appendix D.1). We can see that the singular value distribution of QN is much smoother, and the condition number is improved. Note that for a multi-layer GNN, the normalization will be applied in each layer. Therefore, the overall improvement of such preconditioning can be more significant.

3.2 The Disadvantages of Batch Normalization for Graph

The above analysis shows the benefits of using normalization on the nodes in a single graph. Then a natural question is whether using a batch-level normalization for GNNs (Xu et al., 2019) can lead to similar advantages. In batch normalization (BatchNorm), the mean and standard deviation in a sampled batch are random variables which try to provide accurate estimations for *the mean and standard deviation* over the whole dataset (Ioffe & Szegedy, 2015; Teye et al., 2018; Luo et al., 2019). During testing, the estimated dataset-level statistics are used instead of the batch-level statistics (Ioffe & Szegedy, 2015).

In GNNs, for each feature dimension, the BatchNorm normalizes the feature values of the dimension over all nodes across different graphs in the batch. Note that one can view all graphs in the dataset as isolated subgraphs in a *super graph*. If the batch-level statistics are well-concentrated around dataset-level statistics, we can use Eq. (7) to this super graph, and thus Theorem 3.1 can be applied. Then BatchNorm can be considered as normalizing isolated parts in the super graph, which will enjoy the preconditioning in the theorem. However, the concentration of batch-level statistics is heavily domain-specific. Shen et al. (2020) find that in computer vision, the variation of batch-level statistics in typical networks is quite small while in natural language processing, this variation is large. In GNNs, how the batch-level statistics are concentrated is still unknown. If those values poorly concentrate around the dataset-level statistics, we cannot expect the preconditioning property of the shift operation holds for batch normalization.

To study this, we train a 5-layer GIN with BatchNorm as in Xu et al. (2019) on the PROTEINS dataset and train a ResNet18 (He et al., 2016) on the CIFAR10 dataset for comparison. The batch size of all experiments are set to 128. For each model checkpoint, we record the maximum/minimum batch-level statistics (mean and standard deviation) for the first (layer 0) and the last (layer 3) BatchNorm layer on a randomly picked dimension across different batches. We also calculate the dataset-level statistics. In Figure 3, pink line denotes the dataset-level statistics, and green/blue line denotes the maximum/minimum value of the batch-level statistics respectively. We observe that for image tasks, the batch-level statistics well concentrate around the dataset-level statistics during training. On the contrary, on the graph tasks, the variation of batch-level statistics is rather large. We hypothesize this is due to that the graph structure is quite different between each other and the statistics of a batch is hard to reflect the statistics of the whole dataset. Such heavy noise brings instabilities to the optimization when using BatchNorm, and the preconditioning property also may not hold.

4 GRAPH NORMALIZATION

Although we provide evidence on the indispensability and advantages to apply the normalization in a graph-wise manner, simply normalizing the values in each feature dimension within a graph does not consistently lead to improvement. We show that in some situations, e.g., for regular graphs, the standard shift (e.g., shifting by subtracting the mean) may cause information loss on graph structures. We also show in the experimental section that some graphs in real-world datasets are highly regular.

We consider r-regular graphs, i.e., each node has a degree r. We first look into the case that there are no available node features, then X_i is set to be the one-hot encoding of the node degree (Xu et al., 2019). In a r-regular graph, all nodes have the same encoding, and thus the columns of $H^{(0)}$ are the same. We study the output of the standard shift operation in the first layer, i.e., $k = 1$ in Eq. (7). From the following proposition, we can see that when the standard shift operation is applied to GIN for a r-regular graph described above, the information of degree is lost:

Proposition 4.1. *For a r-regular graph with features described above, we have for GIN, $\mathrm{Norm}\left(W^{(1)}H^{(0)}Q_{\mathrm{GIN}}\right) = S\left(W^{(1)}H^{(0)}Q_{\mathrm{GIN}}\right)N = 0$, i.e., the output of normalization layer is a zero matrix without any information of the graph structure.*

Such information loss not only happens when there are no node features. For complete graphs, we can further show that even each node has different features, the graph structural information, i.e., adjacency matrix A, will always be ignored after the standard shift operation in GIN:

Proposition 4.2. *For a complete graph ($r = n - 1$), we have for GIN, $Q_{\mathrm{GIN}}N = \xi^{(k)}N$, i.e., graph structural information in Q will be removed after multiplying N.*

The proof of these two propositions can be found in Appendix A. Similar results can be easily derived for other architectures like GCN. As we can see from the above analysis, in graph data, the mean statistics after the aggregation sometimes contain structural information. Discarding the mean will degrade the expressiveness of the neural networks. Note that the problem may not happen in image domain. The mean statistics of image data contains global information such as brightness. Removing such information in images will not change the semantics of the objects and thus will not hurt the classification performance.

This analysis inspires us to modify the current normalization method with a *learnable parameter* to automatically control how much the mean to preserve in the shift operation. Combined with the

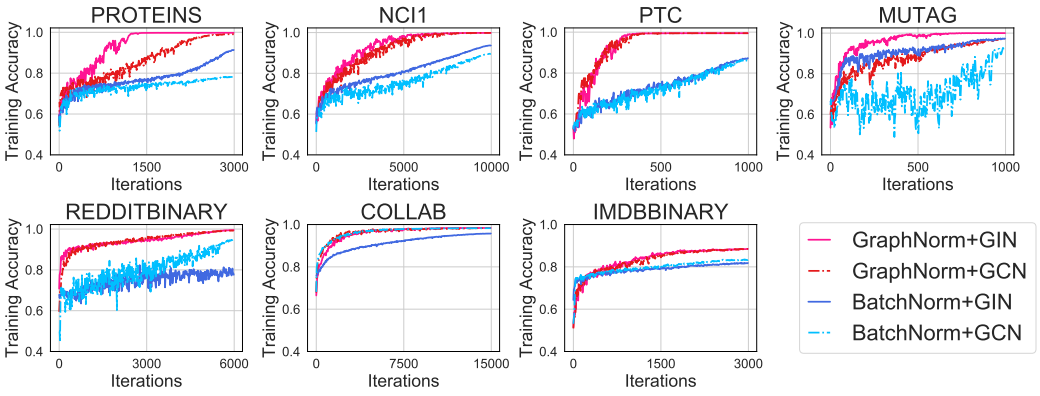

Figure 4: **Training performance** of GIN/GCN with GraphNorm and BatchNorm on different tasks.

Table 1: **Test performance** of GIN/GCN with GraphNorm and BatchNorm on different tasks.

Datasets	MUTAG	PTC	PROTEINS	NCI1	IMDB-B	RDT-B	COLLAB
# graphs	188	344	1113	4110	1000	2000	5000
# classes	2	2	2	2	2	2	2
Avg # nodes	17.9	25.5	39.1	29.8	19.8	429.6	74.5
WL SUBTREE (SHERVASHIDZE ET AL., 2011)	90.4 ± 5.7	59.9 ± 4.3	75.0 ± 3.1	$\mathbf{86.0 \pm 1.8}$	73.8 ± 3.9	81.0 ± 3.1	78.9 ± 1.9
DCNN (ATWOOD & TOWSLEY, 2016)	67.0	56.6	61.3	62.6	49.1	-	52.1
DGCNN (ZHANG ET AL., 2018)	85.8	58.6	75.5	74.4	70.0	-	73.7
AWL (IVANOV & BURNAEV, 2018)	87.9 ± 9.8	-	-	-	74.5 ± 5.9	87.9 ± 2.5	73.9 ± 1.9
GIN+BATCHNORM ((XU ET AL., 2019))	89.4 ± 5.6	64.6 ± 7.0	76.2 ± 2.8	82.7 ± 1.7	75.1 ± 5.1	92.4 ± 2.5	$\mathbf{80.2 \pm 1.9}$
GIN+GraphNorm	$\mathbf{91.6 \pm 6.5}$	$\mathbf{64.9 \pm 7.5}$	$\mathbf{77.4 \pm 4.9}$	81.4 ± 2.4	$\mathbf{76.0 \pm 3.7}$	$\mathbf{93.5 \pm 2.1}$	$\mathbf{80.2 \pm 1.0}$

Datasets	OGBG-MOLHIV
# graphs	41,127
# classes	2
Avg # nodes	25.5
Graph-agnostic MLP (Hu et al., 2020)	68.19 ± 0.71
GCN (Hu et al., 2020)	76.06 ± 0.97
GIN (Hu et al., 2020)	75.58 ± 1.40
GCN+BatchNorm	76.22 ± 0.95
GCN+GraphNorm	$\mathbf{78.30 \pm 0.69}$
GIN+BatchNorm	76.61 ± 0.97
GIN+GraphNorm	$\mathbf{77.73 \pm 1.29}$

graph-wise normalization, we call our new method GraphNorm. For each graph G, we generally denote value $\hat{h}_{i,j}$ as the inputs to GraphNorm, e.g., the j-th feature value of node v_i, $i = 1, \cdots, n$, $j = 1, \cdots, d$. GraphNorm takes the following form:

$$\text{GraphNorm}\left(\hat{h}_{i,j}\right) = \gamma_j \frac{\hat{h}_{i,j} - \alpha_j \cdot \mu_j}{\hat{\sigma}_j} + \beta_j, \tag{9}$$

where $\mu_j = \frac{1}{n}\sum_{i=1}^{n} \hat{h}_{i,j}$, $\hat{\sigma}_j^2 = \frac{1}{n}\sum_{i=1}^{n}\left(\hat{h}_{i,j} - \alpha_j \cdot \mu_j\right)^2$, and γ_j, β_j are the affine parameters as in other normalization methods. By introducing the learnable parameter α_j for each feature dimension j, we are able to learn how much the information we need to keep in the mean. In Section 5.3, we show that using this parameter consistently boosts the convergence speed, and makes a significant improvement on the datasets consisting of "regular" graphs.

5 EXPERIMENTS

5.1 SETTINGS

We use eight popularly used benchmark datasets of different scales in the experiments (Yanardag & Vishwanathan, 2015; Xu et al., 2019), including four medium-scale bioinformatics datasets (MUTAG, PTC, PROTEINS, NCI1), three medium-scale social network datasets (IMDB-BINARY, COLLAB, REDDIT-BINARY), and one large-scale bioinformatics dataset ogbg-molhiv, which is recently released on Open Graph Benchmark (OGB). Dataset statistics are summarized in Table 1. We evaluate our proposed GraphNorm on two typical graph neural networks GIN (Xu et al., 2019) and GCN (Kipf & Welling, 2017) and compare it with BatchNorm[2].Specifically, we use a five-layer GCN/GIN. For GIN, the number of sub-layers in MLP is set to 2. Normalization is applied to each layer. To aggregate global features on top of the network, we use SUM readout for MUTAG, PTC, PROTEINS and NCI1 datasets, and use MEAN readout for other datasets, as in Xu et al. (2019). Details of the experimental settings are presented in Appendix C.

[2]We did not include LayerNorm as a baseline in the main body due to that we observe it usually leads to unsatisfactory performance, see Figure 10 in Appendix. Dwivedi et al. (2020) observed similar phenomena.

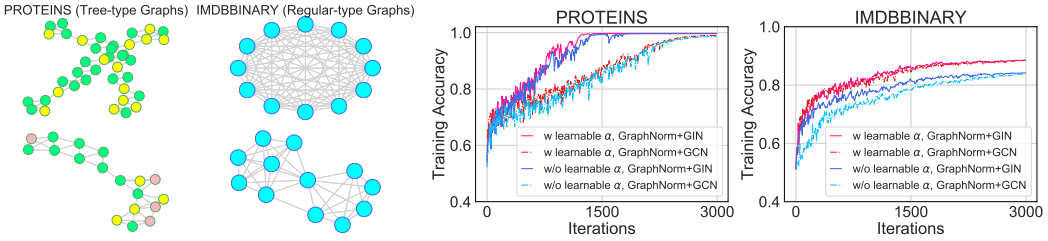

Figure 5: **Ablation study of the parameter α.** Left panel: Sampled graphs with different topological structures. Right panel: training curves of GIN/GCN using GraphNorm with or without α ($\alpha = 1$).

5.2 Results

We plot the training curve of GIN/GCN with GraphNorm and BatchNorm on different tasks in Figure 4. First, from the curve, we can see that GraphNorm is significantly better than BatchNorm in terms of the convergence speed. For example, GIN/GCN with GraphNorm converges in roughly 5000/500 iterations on NCI1 and PTC datasets, while the two models using BatchNorm does not even converge in 10000/1000 iterations. Second, the majority of modern deep learning models are shown to be able to interpolate the data (Zhang et al., 2017; Belkin et al., 2018; Liang & Rakhlin, 2018). But we found that GIN and GCN with BatchNorm are slow to fit the training set well, and the training performance is not very stable, which may due to the large noise induced by the batch-level statistics. However, when using GraphNorm, in most datasets, the model can fit the training data easily.

Besides the training performance, we report the test (validation) accuracy on the datasets in Table 1. From the table, we can see that by using GraphNorm, we can achieve better performance on five tasks, which shows that better optimization leads to better test performance. On the large-scale ogbg-molhiv dataset, the improvements are more impressive. We achieve state-of-the-art performance, and GraphNorm is 2.1/1.1 points better than BatchNorm with GCN/GIN, respectively. As a summary, the experimental results show that using GraphNorm is a better choice for GNNs in terms of both optimization and generalization performance.

5.3 Ablation Study

As mentioned in Section 4, the mean statistics of the feature values in a graph contains structural information. In GraphNorm, we use a learnable shift with parameter α (see Eq. (9)) to preserve such useful information automatically. We conduct experiments to show whether such a learnable α is essential. We use two typical datasets, PROTEINS and IMDB-BINARY, which exhibit irregular-type and regular-type graphs. Sampled cases are visualized in Figure 5.

We follow the same experimental setting as above and train GIN/GCN using GraphNorm. In the first setting, we train the model with a learnable α, and in the second setting, we train the model without α, i.e., by fixing $\alpha = 1$. The training curves are presented in Figure 5. The figure shows that using a learnable α slightly improves the convergence on PROTEINS while significantly boost the training on IMDB-BINARY. This observation shows that shifting the feature values by subtracting the mean losses information, especially for regular graphs. Such results are consistent with our theoretical analysis in Section 4 and verify the necessity of the learnable shift.

6 Conclusion

In this paper, we propose a principled normalization method, called Graph Normalization (GraphNorm), where the key idea is to normalize all nodes for each individual graph with a learnable shift. Theoretically, we show that GraphNorm serves as a preconditioner that smooths the distribution of the graph aggregation's spectrum, and the learnable shift is used to improve the expressiveness of the networks. Experimental results show GNNs with GraphNorm achieve better generalization performance on several benchmark datasets. In the future, we will apply our method to more scenarios and explore other aspects of the optimization for GNNs.

REFERENCES

Sanjeev Arora, Zhiyuan Li, and Kaifeng Lyu. Theoretical analysis of auto rate-tuning by batch normalization. *arXiv preprint arXiv:1812.03981*, 2018.

James Atwood and Don Towsley. Diffusion-convolutional neural networks. In *Advances in neural information processing systems*, pp. 1993–2001, 2016.

Owe Axelsson. A survey of preconditioned iterative methods for linear systems of algebraic equations. *BIT Numerical Mathematics*, 25(1):165–187, 1985.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Mikhail Belkin, Siyuan Ma, and Soumik Mandal. To understand deep learning we need to understand kernel learning. In *International Conference on Machine Learning*, pp. 541–549, 2018.

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

Yihao Chen, Xin Tang, Xianbiao Qi, Chun-Guang Li, and Rong Xiao. Learning graph normalization for graph neural networks, 2020.

Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in neural information processing systems*, pp. 3844–3852, 2016.

James W Demmel. *Applied numerical linear algebra*, volume 56. Siam, 1997.

Simon S Du, Kangcheng Hou, Russ R Salakhutdinov, Barnabas Poczos, Ruosong Wang, and Keyulu Xu. Graph neural tangent kernel: Fusing graph neural networks with graph kernels. In *Advances in Neural Information Processing Systems*, pp. 5724–5734, 2019.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(Jul):2121–2159, 2011.

Vijay Prakash Dwivedi, Chaitanya K Joshi, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking graph neural networks. *arXiv preprint arXiv:2003.00982*, 2020.

Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International Conference on Machine Learning*, pp. 1273–1272, 2017.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in neural information processing systems*, pp. 1024–1034, 2017.

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

Elad Hoffer, Ron Banner, Itay Golan, and Daniel Soudry. Norm matters: efficient and accurate normalization schemes in deep networks. In *Advances in Neural Information Processing Systems*, pp. 2160–2170, 2018.

Roger A Horn and Charles R Johnson. *Matrix analysis*. Cambridge university press, 2012.

Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *arXiv preprint arXiv:2005.00687*, 2020.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pp. 448–456, 2015.

Sergey Ivanov and Evgeny Burnaev. Anonymous walk embeddings. In *International Conference on Machine Learning*, pp. 2191–2200, 2018.

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

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

Jonas Kohler, Hadi Daneshmand, Aurelien Lucchi, Thomas Hofmann, Ming Zhou, and Klaus Neymeyr. Exponential convergence rates for batch normalization: The power of length-direction decoupling in non-convex optimization. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pp. 806–815, 2019.

Guohao Li, Chenxin Xiong, Ali Thabet, and Bernard Ghanem. Deepergcn: All you need to train deeper gcns. *arXiv preprint arXiv:2006.07739*, 2020.

Zhiyuan Li and Sanjeev Arora. An exponential learning rate schedule for deep learning. *arXiv preprint arXiv:1910.07454*, 2019.

Tengyuan Liang and Alexander Rakhlin. Just interpolate: Kernel" ridgeless" regression can generalize. *arXiv preprint arXiv:1808.00387*, 2018.

Ping Luo, Xinjiang Wang, Wenqi Shao, and Zhanglin Peng. Towards understanding regularization in batch normalization. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=HJlLKjR9FQ`.

Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=B1QRgziT-`.

Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodola, Jan Svoboda, and Michael M Bronstein. Geometric deep learning on graphs and manifolds using mixture model cnns. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5115–5124, 2017.

Tim Salimans and Durk P Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Advances in neural information processing systems*, pp. 901–909, 2016.

Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. In *Advances in neural information processing systems*, pp. 2234–2242, 2016.

Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How does batch normalization help optimization? In *Advances in Neural Information Processing Systems*, pp. 2483–2493, 2018.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2008.

Sheng Shen, Zhewei Yao, Amir Gholami, Michael W. Mahoney, and Kurt Keutzer. Powernorm: Rethinking batch normalization in transformers, 2020.

Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(Sep): 2539–2561, 2011.

Sainbayar Sukhbaatar, Rob Fergus, et al. Learning multiagent communication with backpropagation. In *Advances in neural information processing systems*, pp. 2244–2252, 2016.

Mattias Teye, Hossein Azizpour, and Kevin Smith. Bayesian uncertainty estimation for batch normalized deep networks. In *International Conference on Machine Learning*, pp. 4907–4916, 2018.

Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022*, 2016.

Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. 2018.

Yuxin Wu and Kaiming He. Group normalization. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 3–19, 2018.

Zhenqin Wu, Bharath Ramsundar, Evan N. Feinberg, Joseph Gomes, Caleb Geniesse, Aneesh S. Pappu, Karl Leswing, and Vijay S. Pande. Moleculenet: A benchmark for molecular machine learning. *CoRR*, abs/1703.00564, 2017. URL http://arxiv.org/abs/1703.00564.

Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tie-Yan Liu. On layer normalization in the transformer architecture. *arXiv preprint arXiv:2002.04745*, 2020.

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

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=ryGs6iA5Km.

Keyulu Xu, Jingling Li, Mozhi Zhang, Simon S. Du, Ken ichi Kawarabayashi, and Stefanie Jegelka. What can neural networks reason about? In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=rJxbJeHFPS.

Pinar Yanardag and SVN Vishwanathan. Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1365–1374, 2015.

Chaoqi Yang, Ruijie Wang, Shuochao Yao, Shengzhong Liu, and Tarek Abdelzaher. Revisiting" over-smoothing" in deep gcns. *arXiv preprint arXiv:2003.13663*, 2020.

Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. In *International Conference on Learning Representations*, 2017.

Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. An end-to-end deep learning architecture for graph classification. pp. 4438–4445, 2018.

Ziwei Zhang, Peng Cui, and Wenwu Zhu. Deep learning on graphs: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 2020.

Lingxiao Zhao and Leman Akoglu. Pairnorm: Tackling oversmoothing in gnns. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=rkecl1rtwB.

Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *arXiv preprint arXiv:1812.08434*, 2018.

Kaixiong Zhou, Xiao Huang, Yuening Li, Daochen Zha, Rui Chen, and Xia Hu. Towards deeper graph neural networks with differentiable group normalization. *arXiv preprint arXiv:2006.06972*, 2020a.

Kuangqi Zhou, Yanfei Dong, Wee Sun Lee, Bryan Hooi, Huan Xu, and Jiashi Feng. Effective training strategies for deep graph neural networks, 2020b.

Difan Zou, Ziniu Hu, Yewen Wang, Song Jiang, Yizhou Sun, and Quanquan Gu. Layer-dependent importance sampling for training deep and large graph convolutional networks. In *Advances in Neural Information Processing Systems*, pp. 11249–11259, 2019.

A PROOFS

A.1 PROOF OF THEOREM 3.1

We first introduce the Cauchy interlace theorem:

Lemma A.1 (Cauchy interlace theorem (Theorem 4.3.17 in Horn & Johnson (2012))). *Let $S \in \mathbb{R}^{(n-1)\times(n-1)}$ be symmetric, $y \in \mathbb{R}^n$ and $a \in \mathbb{R}$ be given, and let $R = \begin{pmatrix} S & y \\ y^\top & a \end{pmatrix} \in \mathbb{R}^{n\times n}$. Let $\lambda_1 \le \lambda_2 \le \cdots \le \lambda_n$ be the eigenvalues of R and $\mu_1 \le \mu_2 \le \cdots \le \mu_{n-1}$ be the eigenvalues of S. Then*

$$\lambda_1 \le \mu_1 \le \lambda_2 \le \cdots \le \lambda_{n-1} \le \mu_{n-1} \le \lambda_n, \tag{10}$$

where $\lambda_i = \mu_i$ only when there is a nonzero $z \in \mathbb{R}^{n-1}$ such that $Sz = \mu_i z$ and $y^\top z = 0$; if $\lambda_i = \mu_{i-1}$ then there is a nonzero $z \in \mathbb{R}^{n-1}$ such that $Sz = \mu_{i-1}z$, $y^\top z = 0$.

Using Lemma A.1, the theorem can be proved as below.

Proof. For any matrices $P, R \in \mathbb{R}^{n\times n}$, we use $P \sim R$ to denote that the matrix P is similar to the matrix R. Note that if $P \sim R$, the eigenvalues of P and R are the same. As the singular values of P are equal to the square root of the eigenvalues of $P^\top P$, we have the eigenvalues of $Q^\top Q$ and that of $NQ^\top QN$ are $\left\{\lambda_i^2\right\}_{i=1}^n$ and $\left\{\mu_i^2\right\}_{i=1}^n$, respectively.

Note that N is a projection operator onto the orthogonal complement space of the subspace spanned by $\mathbf{1}$, and N can be decomposed as $N = U \operatorname{diag}\left(\underbrace{1, \cdots, 1}_{\times n-1}, 0\right) U^\top$ where U is an orthogonal matrix.

Since $\mathbf{1}$ is the eigenvector of N associated with eigenvalue 0, we have

$$U = \begin{pmatrix} U_1 & \frac{1}{\sqrt{n}}\mathbf{1} \end{pmatrix}, \tag{11}$$

where $U_1 \in \mathbb{R}^{n\times(n-1)}$ satisfies $U_1\mathbf{1} = 0$ and $U_1^\top U_1 = I_{n-1}$.

Then we have $NQ^\top QN = U \operatorname{diag}(1, \cdots, 1, 0) U^\top Q^\top QU \operatorname{diag}(1, \cdots, 1, 0) U^\top \sim \operatorname{diag}(1, \cdots, 1, 0) U^\top Q^\top QU \operatorname{diag}(1, \cdots, 1, 0)$.

Let

$$D = \operatorname{diag}(1, \cdots, 1, 0) = \begin{pmatrix} I_{n-1} & \mathbf{0} \\ \mathbf{0}^\top & 0 \end{pmatrix}, \tag{12}$$

$$B = \begin{pmatrix} I_{n-1} \\ \mathbf{0}^\top \end{pmatrix}, \tag{13}$$

$$\bar{C} = Q^\top Q, \tag{14}$$

where $\mathbf{0} = \left[\underbrace{0, \cdots, 0}_{\times n-1}\right]^\top$.

We have

$$NQ^\top QN \sim DU^\top \bar{C} UD \tag{15}$$

$$= D \begin{pmatrix} U_1^\top \\ \frac{1}{\sqrt{n}}\mathbf{1}^\top \end{pmatrix} \bar{C} \begin{pmatrix} U_1 & \frac{1}{\sqrt{n}}\mathbf{1} \end{pmatrix} D \tag{16}$$

$$= D \begin{pmatrix} U_1^\top \bar{C} U_1 & \frac{1}{\sqrt{n}} U_1^\top \bar{C}\mathbf{1} \\ \frac{1}{\sqrt{n}}\mathbf{1}^\top \bar{C} U_1 & \frac{1}{n}\mathbf{1}^\top \bar{C}\mathbf{1} \end{pmatrix} D \tag{17}$$

$$= \begin{pmatrix} B^\top \\ \mathbf{0}^\top \quad 0 \end{pmatrix} \begin{pmatrix} U_1^\top \bar{C} U_1 & \frac{1}{\sqrt{n}} U_1^\top \bar{C}\mathbf{1} \\ \frac{1}{\sqrt{n}}\mathbf{1}^\top \bar{C} U_1 & \frac{1}{n}\mathbf{1}^\top \bar{C}\mathbf{1} \end{pmatrix} \begin{pmatrix} B & \mathbf{0} \\ & 0 \end{pmatrix} \tag{18}$$

$$= \begin{pmatrix} U_1^\top \bar{C} U_1 & \mathbf{0} \\ \mathbf{0}^\top & 0 \end{pmatrix}. \tag{19}$$

Using Lemma A.1 and taking $R = U^\top \bar{C} U$ and $S = U_1^\top \bar{C} U_1$, we have the eigenvalues of $U_1^\top \bar{C} U_1$ are interlacing between the eigenvalues of $U^\top \bar{C} U$. Note that the eigenvalues of $DU^\top \bar{C} U D$ are $\mu_1^2 \leq \mu_2^2 \leq \cdots \leq \mu_{n-1}^2$ and $\mu_n^2 = 0$, and by Eq. (19), the eigenvalues of $DU^\top \bar{C} U D$ contain the eigenvalues of $U_1^\top \bar{C} U_1$ and 0. Since the eigenvalues of $U^\top \bar{C} U$ are $\lambda_1^2 \leq \lambda_2^2 \leq \cdots \leq \lambda_n^2$ (By similarity of $U^\top \bar{C} U$ and $\bar{C}$), we then have

$$\lambda_1^2 \leq \mu_1^2 \leq \lambda_2^2 \leq \cdots \leq \lambda_{n-1}^2 \leq \mu_{n-1}^2 \leq \lambda_n^2. \tag{20}$$

Moreover, the equality holds only when there is a nonzero $z \in \mathbb{R}^{n-1}$ that satisfies

$$U_1^\top \bar{C} U_1 z = \mu z, \tag{21}$$

$$\mathbf{1}^\top \bar{C} U_1 z = 0, \tag{22}$$

where μ is one of μ_i^2s.

Since U_1 forms an orthogonal basis of the orthogonal complement space of $\mathbf{1}$ and Eq. (22) is equivalent to "$\bar{C} U_1 z$ lies in the orthogonal complement space", we have that there is a vector $y \in \mathbb{R}^{n-1}$ such that

$$\bar{C} U_1 z = U_1 y. \tag{23}$$

Substituting this into Eq. (21), we have

$$U_1^\top U_1 y = \mu z. \tag{24}$$

Since $U_1^\top U_1 = I_{n-1}$, the equation above is equivalent to

$$y = \mu z, \tag{25}$$

which means

$$\bar{C} U_1 z = U_1 y = \mu U_1 z, \tag{26}$$

i.e., $U_1 z$ is the eigenvector of $\bar{C}$ associated with μ. By noticing $U_1 z$ lies in the orthogonal complement space of $\mathbf{1}$ and the eigenvector of $\bar{C}$ is right singular vector of Q, we complete the proof. □

A.2 Proof of Proposition 4.1

Proof. For r-regular graph, $A = r \cdot I_n$ and $Q_{\text{GIN}} = \left(r + 1 + \xi^{(1)}\right) I_n$. Since $H^{(0)}$ is given by one-hot encodings of node degrees, the row of $H^{(0)}$ can be represented as $c \cdot \mathbf{1}^\top$ where $c = 1$ for the r-th row and $c = 0$ for other rows. By the associative property of matrix multiplication, we only need to show $H^{(0)} Q_{\text{GIN}} N = 0$. This is because, for each row

$$c \cdot \mathbf{1}^\top Q_{\text{GIN}} N = c \cdot \mathbf{1}^\top (r + 1 + \xi^{(1)}) I_n \left(I_n - \frac{1}{n} \mathbf{1}\mathbf{1}^\top \right) \tag{27}$$

$$= c \left(r + 1 + \xi^{(1)}\right) \left(\mathbf{1}^\top - \mathbf{1}^\top \cdot \frac{1}{n} \mathbf{1}\mathbf{1}^\top \right) = 0. \tag{28}$$

□

A.3 Proof of Proposition 4.2

Proof.

$$Q_{\text{GIN}} N = (A + I_n + \xi^{(k)} I_n) N == (\mathbf{1}\mathbf{1}^\top + \xi^{(k) I_n}) N = \xi^{(k)} N, \tag{29}$$

□

A.4 Gradient of $W^{(k)}$

We first calculate the gradient of $W^{(k)}$ when using normalization. Denote $Z^{(k)} = \text{Norm}\left(W^{(k)} H^{(k-1)} Q\right)$ and $\mathcal{L}$ as the loss. Then the gradient of $\mathcal{L}$ w.r.t. the weight matrix $W^{(k)}$ is

$$\frac{\partial \mathcal{L}}{\partial W^{(k)}} = \left(\left(H^{(k-1)} Q N \right)^\top \otimes S \right) \frac{\partial \mathcal{L}}{\partial Z^{(k)}}, \tag{30}$$

Table 2: **Summary of statistics of benchmark datasets.**

Datasets	MUTAG	PTC	PROTEINS	NCI1	IMDB-B	RDT-B	COLLAB	OGBG-MOLHIV
# graphs	188	344	1113	4110	1000	2000	5000	41127
# classes	2	2	2	2	2	2	2	2
Avg # nodes	17.9	25.5	39.1	29.8	19.8	429.6	74.5	25.5
Avg # edges	57.5	72.5	184.7	94.5	212.8	1425.1	4989.5	27.5
Avg # degrees	3.2	3.0	4.7	3.1	10.7	3.3	66.9	2.1

where $\otimes$ represents the Kronecker product, and thus $\left(H^{(k-1)}QN\right)^\top \otimes S$ is an operator on matrices.

Analogously, the gradient of $W^{(k)}$ without normalization consists a $\left(H^{(k-1)}Q\right)^\top \otimes I_n$ term. As suggested by Theorem 3.1, QN has a smoother distribution of spectrum than Q, so that the gradient of $W^{(k)}$ with normalization enjoys better optimization curvature than that without normalizaiton.

B DATASETS

Detailed of the datasets used in our experiments are presented in this section. Brief statistics of the datasets are summarized in Table 2. Those information can be also found in Xu et al. (2019) and Hu et al. (2020).

Bioinformatics datasets. PROTEINS is a dataset where nodes are secondary structure elements (SSEs) and there is an edge between two nodes if they are neighbors in the amino-acid sequence or in 3D space. It has 3 discrete labels, representing helix, sheet or turn. NCI1 is a dataset made publicly available by the National Cancer Institute (NCI) and is a subset of balanced datasets of chemical compounds screened for ability to suppress or inhibit the growth of a panel of human tumor cell lines, having 37 discrete labels. MUTAG is a dataset of 188 mutagenic aromatic and heteroaromatic nitro compounds with 7 discrete labels. PTC is a dataset of 344 chemical compounds that reports the carcinogenicity for male and female rats and it has 19 discrete labels.

Social networks datasets. IMDB-BINARY is a movie collaboration dataset. Each graph corresponds to an ego-network for each actor/actress, where nodes correspond to actors/actresses and an edge is drawn betwen two actors/actresses if they appear in the same movie. Each graph is derived from a pre-specified genre of movies, and the task is to classify the genre graph it is derived from. REDDIT-BINARY is a balanced dataset where each graph corresponds to an online discussion thread and nodes correspond to users. An edge was drawn between two nodes if at least one of them responded to another's comment. The task is to classify each graph to a community or a subreddit it belongs to. COLLAB is a scientific collaboration dataset, derived from 3 public collaboration datasets, namely, High Energy Physics, Condensed Matter Physics and Astro Physics. Each graph corresponds to an ego-network of different researchers from each field. The task is to classify each graph to a field the corresponding researcher belongs to.

Large-scale Open Graph Benchmark: ogbg-molhiv. Ogbg-molhiv is a molecular property prediction dataset, which is adopted from the the MOLECULENET (Wu et al., 2017). Each graph represents a molecule, where nodes are atoms and edges are chemical bonds. Both nodes and edges have associated diverse features. Node features are 9-dimensional, containing atomic number and chirality, as well as other additional atom features. Edge features are 3-dimensional, containing bond type, stereochemistry as well as an additional bond feature indicating whether the bond is conjugated.

C THE EXPERIMENTAL SETUP

Network architecture. For the medium-scale bioinformatics and social network datasets, we use 5-layer GIN/GCN with a linear output head for prediction followed Xu et al. (2019) with residual connection. The hidden dimension of GIN/GCN is set to be 64. For the large-scale ogbg-

molhiv dataset, we also use 5-layer GIN/GCN(Xu et al., 2019) architecture with residual connection. Following Hu et al. (2020), we set the hidden dimension as 300.

Baselines. For the medium-scale bioinformatics and social network datasets, we compare several competitive baselines as in x̂u2018how, including the WL subtree kernel model (Shervashidze et al., 2011), diffusion-convolutional neural networks (DCNN) (Atwood & Towsley, 2016), Deep Graph CNN (DGCNN) (Zhang et al., 2018) and Anonymous Walk Embeddings (AWL) (Ivanov & Burnaev, 2018). We report the accuracies reported in the original paper (Xu et al., 2019). For the large-scale ogbg-molhiv dataset, we use the baselines in Hu et al. (2020), including the Graph-agnostic MLP model, GCN (Kipf & Welling, 2017) and GIN (Xu et al., 2019). We also report the roc-auc values reported in the original paper (Hu et al., 2020).

Hyper-parameter configurations. We use Adam (Kingma & Ba, 2015) optimizer with a linear learning rate decay schedule. We follow previous work Xu et al. (2019) and Hu et al. (2020) to use hyper-parameter search (grid search) to select the best hyper-parameter based on validation performance. In particular, we select the batch size $\in \{64, 128\}$, the dropout ratio $\in \{0, 0.5\}$, weight decay $\in \{5e-2, 5e-3, 5e-4, 5e-5\} \cup \{0.0\}$, the learning rate $\in \{1e-4, 1e-3, 1e-2\}$. For the drawing of the training curves in Figure 4, for simplicity, we set batch size to be 128, dropout ratio to be 0.5, weight decay to be 0.0, learning rate to be 1e-2, and train the models for 400 epochs for all settings.

Evaluation. Using the chosen hyper-parameter, we report the averaged test performance over different random seeds (or cross-validation). In detail, for the medium-scale datasets, following Xu et al. (2019), we perform a 10-fold cross-validation as these datasets do not have a clear train-validate-test splitting format. The mean and standard deviation of the validation accuracies across the 10 folds are reported. For the ogbg-molhiv dataset, we follow the official setting (Hu et al., 2020). We repeat the training process with 10 different random seeds.

For all experiments, we select the best model checkpoint with the best validation accuracy and record the corresponding test performance.

D Additional Experimental Results

D.1 Visualization of the singular value distributions

As stated in Theorem 3.1, the shift operation N serves as a preconditioner of Q which makes the singular value distribution of Q smoother. To check the improvements, we sample graphs from 6 median-scale datasets (PROTEINS, NCI1, MUTAG, PTC, IMDB-BINARY, COLLAB) for visualization, as in Figure 6.

D.2 Visualization of noise in the batch statistics

We show the noise of the batch statistics on the PROTEINS task in the main body. Here we provide more experiment details and results.

For graph tasks (PROTEINS, PTC, NCI1, MUTAG, IMDB-BINARY datasets), we train a 5-layer GIN with BatchNorm as in Xu et al. (2019) and the number of sub-layers in MLP is set to 2. For image task (CIFAR10 dataset), we train a ResNet18 (He et al., 2016). Note that for a 5-layer GIN model, it has four graph convolution layers (indexed from 0 to 3) and each graph convolution layer has two BatchNorm layers; for a ResNet18 model, except for the first 3×3 convolution layer and the final linear prediction layer, it has four basic layers (indexed from 0 to 3) and each layer consists of two basic blocks (each block has two BatchNorm layers). For image task, we set the batch size as 128, epoch as 100, learning rate as 0.1 with momentum 0.9 and weight decay as 5e-4. For graph tasks, we follow the setting of Figure 4 (described in Appendix C).

The visualization of the noise in the batch statistics is obtained as follows. We first train the models and dump the model checkpoints at the end of each epoch; Then we randomly sample one feature dimension and fix it. For each model checkpoint, we feed different batches to the model and record

the maximum/minimum batch-level statistics (mean and standard deviation) of the feature dimension across different batches. We also calculate dataset-level statistics.

As Figure 3 in the main body, pink line denotes the dataset-level statistics, and green/blue line denotes the maximum/minimum value of the batch-level statistics respectively. First, we provide more results on PTC, NCI1, MUTAG, IMDB-BINARY tasks, as in Figure 7. We visualize the statistics from the first (layer-0) and the last (layer-3) BatchNorm layers in GIN for comparison. Second, we further visualize the statistics from different BatchNorm layers (layer 0 to layer 3) in GIN on PROTEINS and ResNet18 in CIFAR10, as in Figure 8. Third, we conduct experiments to investigate the influence of the batch size. We visualize the statistics from BatchNorm layers under different settings of batch sizes [8, 16, 32, 64], as in Figure 9. We can see that the observations are consistent and the batch statistics on graph data are noisy, as in Figure 3 in the main body.

E Other Related Works

Due to space limitations, we add some more related works on normalization and graph neural networks here. Zou et al. (2019) used normalization to stabilize the training process of GNNs. Zhao & Akoglu (2020) introduced PAIRNORM to prevent node embeddings from over-smoothing on the node classification task. Our GraphNorm focuses on accelerating the training and has faster convergence speed on graph classification tasks. Yang et al. (2020) interpreted the effect of mean subtraction on GCN as approximating the Fiedler vector. We analyze more general aggregation schemes, e.g., those in GIN, and understand the effect of the shift through the distribution of spectrum. Some concurrent and independent works (Li et al., 2020; Chen et al., 2020; Zhou et al., 2020a;b) also seek to incorporate normalization schemes in GNNs, which show the urgency of developing normalization schemes for GNNs. In this paper, we provide several insights on how to design a proper normalization for GNNs. Before the surge of deep learning, there are also many classic architectures of GNNs such as Scarselli et al. (2008); Bruna et al. (2013); Defferrard et al. (2016) that are not mentioned in the main body of the paper. We refer the readers to Zhou et al. (2018); Wu et al. (2020); Zhang et al. (2020) for surveys of graph representation learning.

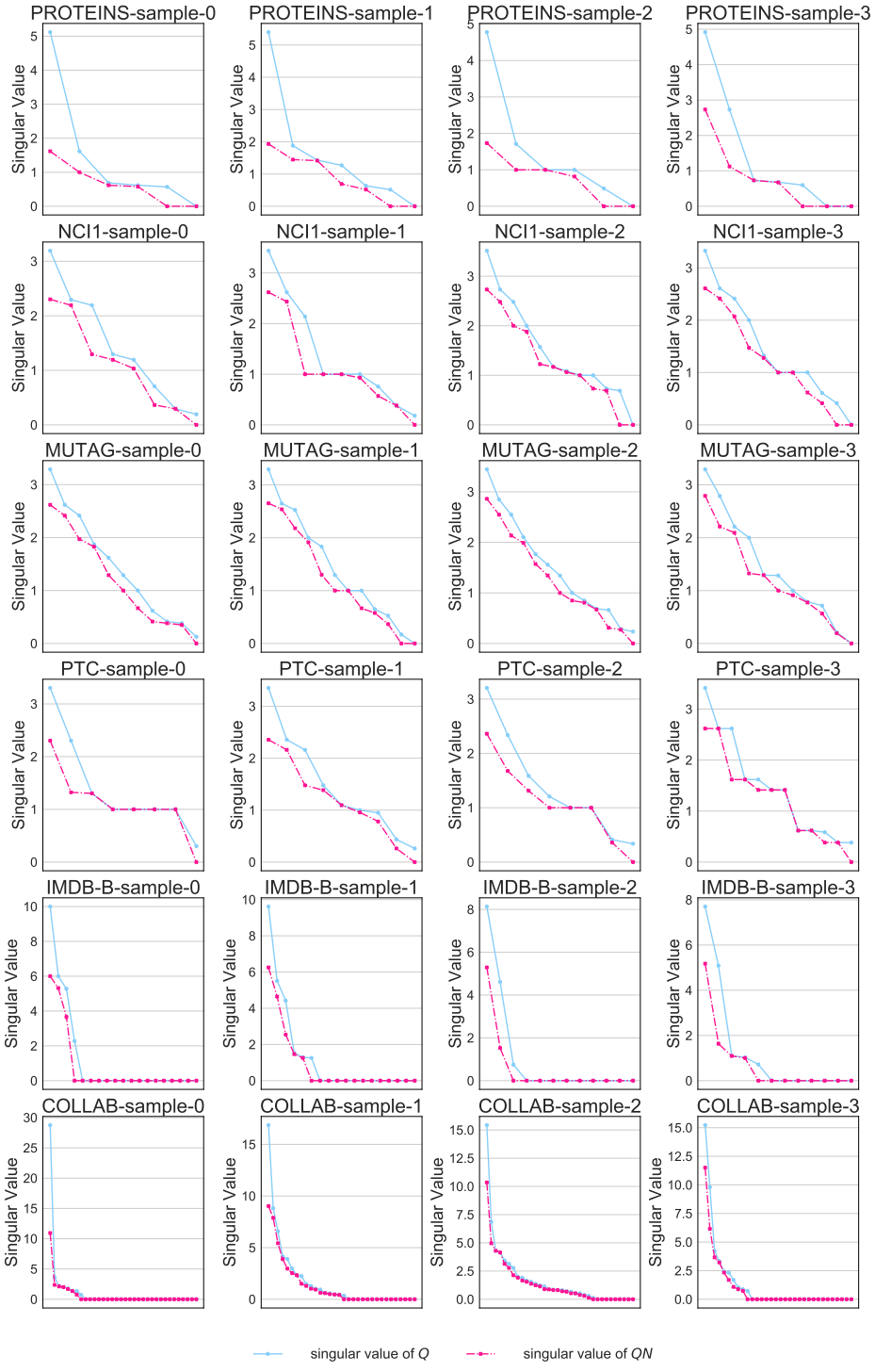

Figure 6: **Singular value distribution of Q and QN.** Graph samples from PROTEINS, NCI1, MUTAG, PTC, IMDB-BINARY, COLLAB are presented.

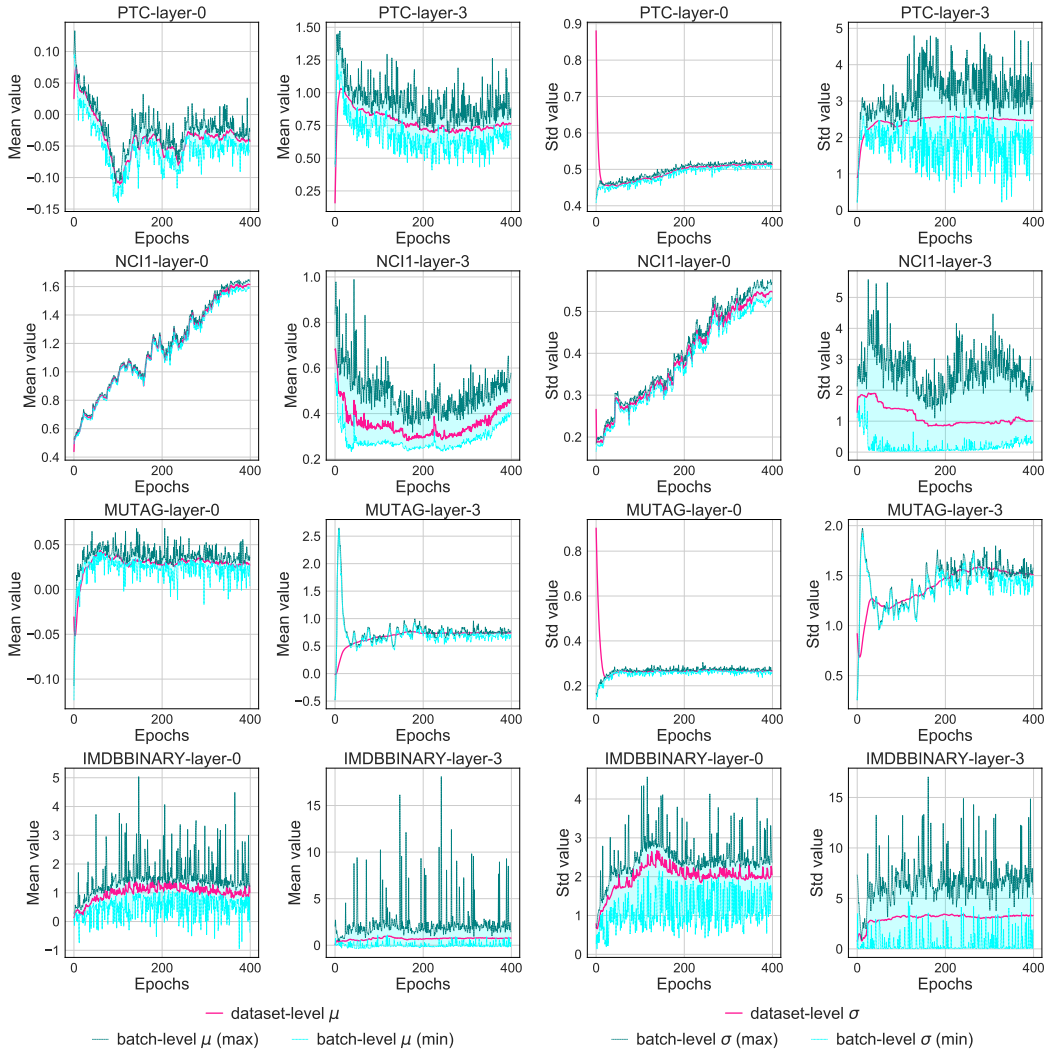

Figure 7: **Batch-level statistics are noisy for GNNs** (Examples from PTC, NCI1, MUTAG, IMDB-BINARY datasets). We plot the batch-level mean/standard deviation and dataset-level mean/standard deviation of the first (layer 0) and the last (layer 3) BatchNorm layers in different checkpoints. GIN with 5 layers is employed.

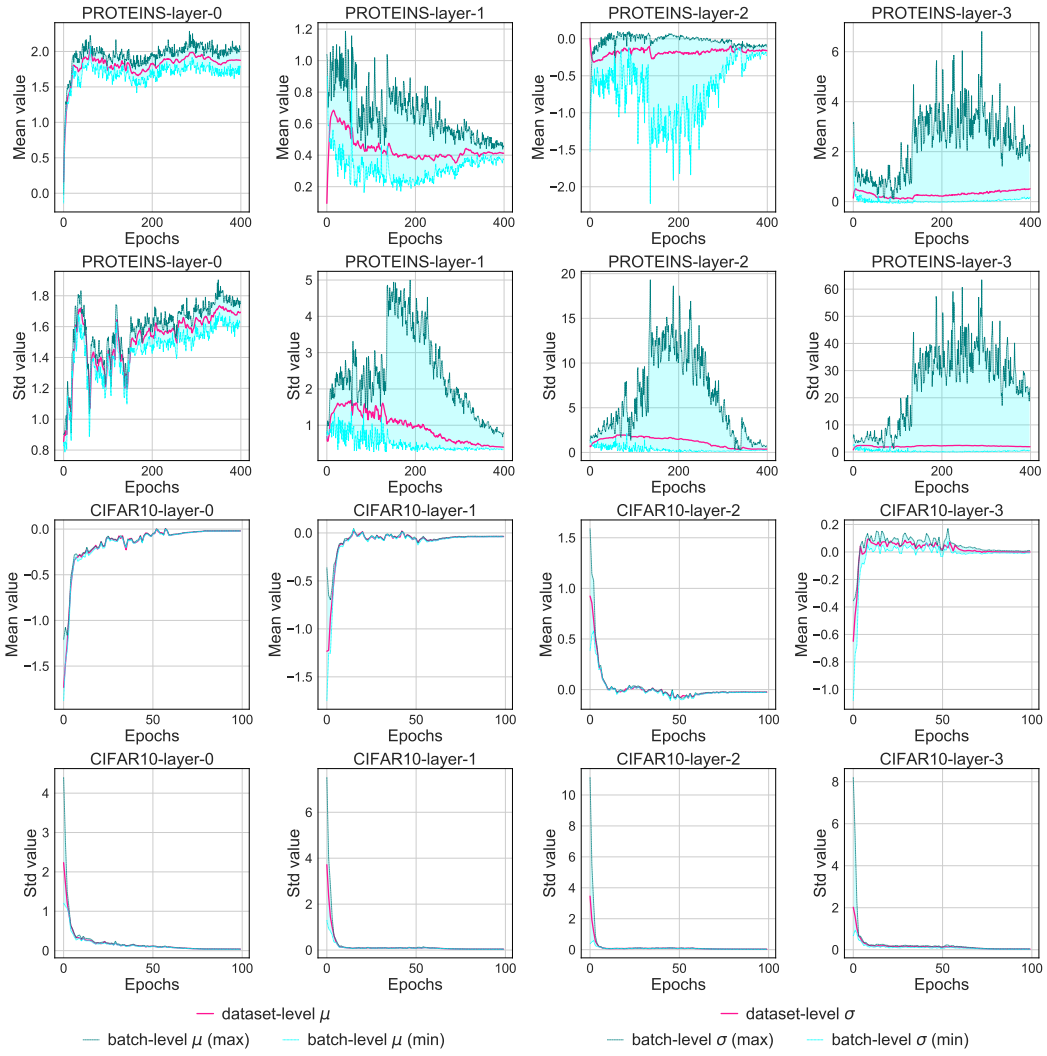

Figure 8: **Batch-level statistics are noisy for GNNs of different depth.** We plot the batch-level mean/standard deviation and dataset-level mean/standard deviation of different BatchNorm layers (from layer 0 to layer 3) in different checkpoints. We use a five-layer GIN on PROTEINS and ResNet18 on CIFAR10 for comparison.

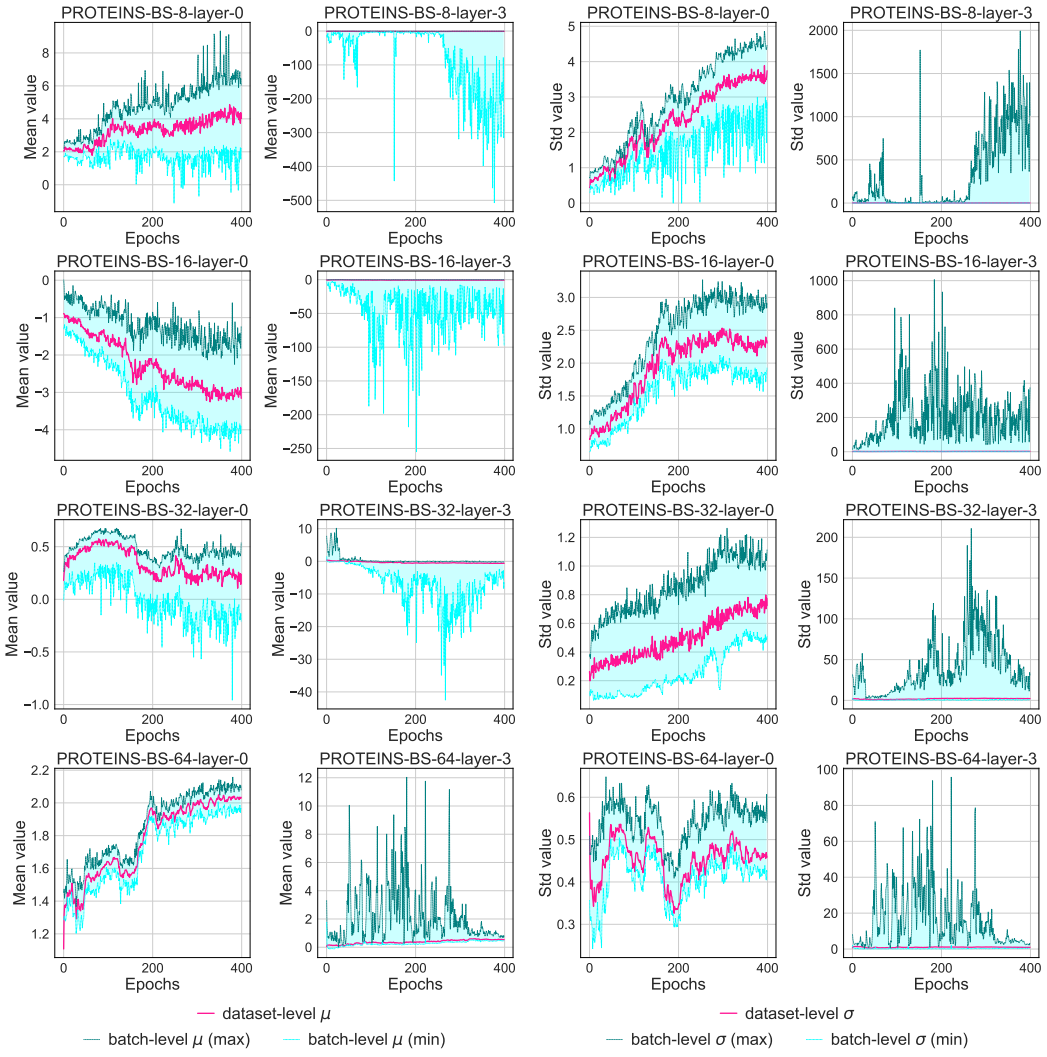

Figure 9: **Batch-level statistics are noisy for GNNs of different batch sizes.** We plot the batch-level mean/standard deviation and dataset-level mean/standard deviation of different BatchNorm layers (layer 0 and layer 3) in different checkpoints. Specifically, different batch sizes (8, 16, 32, 64) are chosed for comparison. GIN with 5 layers is employed.

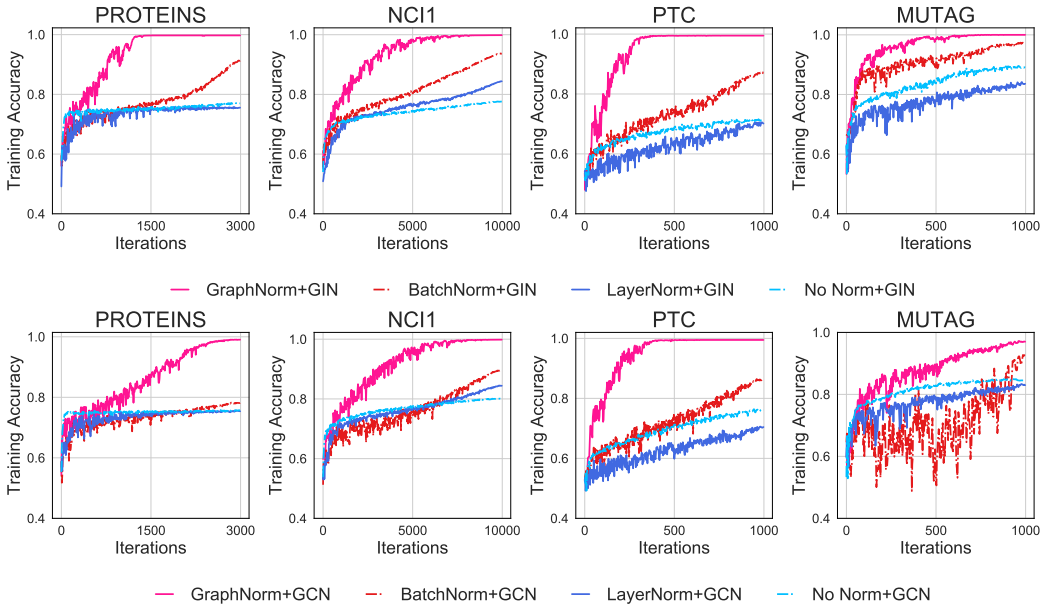

Figure 10: **Training performance** of GIN/GCN with GraphNorm, BatchNorm , LayerNorm and without normalization on PROTEINS, NCI1, PTC and MUTAG datasets.