GraphNorm: A Principled Approach to Accelerating Graph Neural Network Training

Tianle Cai [* 1] Shengjie Luo [* 2] Keyulu Xu [3] Di He [4] Tie-Yan Liu [4] Liwei Wang [2]

Abstract

Normalization is known to help the optimization of deep neural networks. Curiously, different architectures require specialized normalization methods. In this paper, we study what normalization is effective for Graph Neural Networks (GNNs). First, we adapt and evaluate the existing methods from other domains to GNNs. Faster convergence is achieved with InstanceNorm compared to BatchNorm and LayerNorm. We provide an explanation by showing that InstanceNorm serves as a preconditioner for GNNs, but such preconditioning effect is weaker with BatchNorm due to the heavy batch noise in graph datasets. Second, we show that the shift operation in InstanceNorm results in an expressiveness degradation of GNNs for highly regular graphs. We address this issue by proposing GraphNorm with a learnable shift. Empirically, GNNs with GraphNorm converge faster compared to GNNs using other normalization. GraphNorm also improves the 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 graphs (Gori et al., 2005; Scarselli et al., 2008; Hamilton et al., 2017; Kipf & Welling, 2017; Velickovic et al., 2018; Xu et al., 2018). GNNs learn node and graph representations by recursively aggregating and updating the node representations from neighbor representations (Gilmer et al., 2017). Empirically, GNNs have succeeded in a variety of tasks such as computational chemistry (Stokes et al., 2020), recommendation systems (Ying et al., 2018), and visual question answering (Santoro et al., 2017). Theoretically, existing works have studied GNNs through the lens of expressive power (Keriven & Peyré, 2019; Xu et al., 2019; Sato et al., 2019; Loukas, 2020), generalization (Scarselli et al., 2018; Du et al., 2019b; Xu et al., 2020), and extrapolation (Xu et al., 2021). However, the optimization of GNNs is less well understood, and in practice, the training of GNNs is often unstable and the convergence is slow (Xu et al., 2019).

In this paper, we study how to improve the training of GNNs via normalization. Normalization methods shift and scale the hidden representations and are shown to help the optimization for deep neural networks (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; Santurkar et al., 2018). Curiously, no single normalization helps in every domain, and different architectures require specialized methods. For example, Batch normalization (BatchNorm) is a standard component in computer vision (Ioffe & Szegedy, 2015); Layer normalization (LayerNorm) is popular in natural language processing (Ba et al., 2016; Xiong et al., 2020); Instance normalization (InstanceNorm) has been found effective for style transfer tasks (Ulyanov et al., 2016) . This motivates the question: *What normalization methods are effective for GNNs?*

We take an initial step towards answering the question above. First, we adapt the existing methods from other domains, including BatchNorm, LayerNorm, and InstanceNorm, to GNNs and evaluate their performance with extensive experiments on graph classification tasks. We observe that our adaptation of InstanceNorm to GNNs, which for each *individual graph* normalizes its node hidden representations, obtains much faster convergence compared to BatchNorm and LayerNorm. We provide an explanation for the success of InstanceNorm by showing that the shift operation in InstanceNorm serves as a preconditioner of the graph aggregation operation. Empirically, such preconditioning makes the optimization curvature smoother and makes the training more efficient. We also explain why the widely used BatchNorm does not bring the same level of acceleration. The variance of the batch-level statistics on graph datasets is much larger if we apply the normalization across graphs in a batch instead of across individual graphs. The noisy statistics during training may lead to unstable optimization.

Second, we show that the adaptation of InstanceNorm to

[*]Equal contribution [1]Princeton University [2]Peking University [3]MIT [4]Microsoft Research. Correspondence to: Liwei Wang <wanglw@pku.edu.cn>.

Preliminary work.

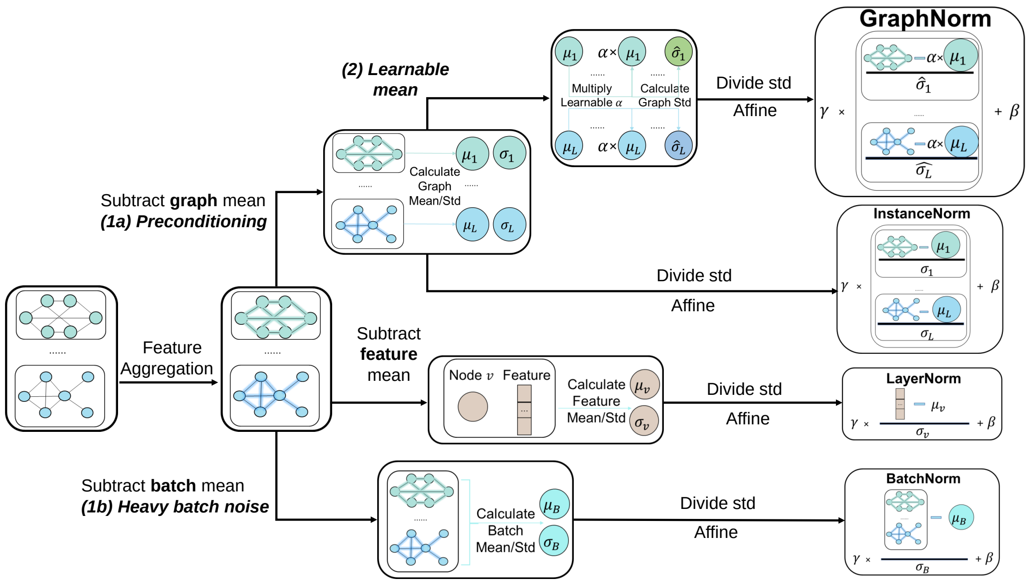

Figure 1: **Overview.** We evaluate and understand BatchNorm, LayerNorm, and InstanceNorm, when adapted to GNNs. InstanceNorm trains faster than LayerNorm and BatchNorm on most datasets (Section 3.1), as it serves as a preconditioner of the aggregation of GNNs (1a, Section 3.2). The preconditioning effect is weaker for BatchNorm due to heavy batch noise in graphs (1b, Section 3.3). We propose GraphNorm with a learnable shift to address the limitation of InstanceNorm. GraphNorm outperforms other normalization methods for both training speed (Figure 2) and generalization (Table 1, 2).

GNNs, while being helpful in general, has limitations. The shift operation in InstanceNorm, which subtracts the mean statistics from node hidden representations, may lead to an expressiveness degradation for GNNs. Specifically, for highly regular graphs, the mean statistics contain graph structural information, and thus removing them could hurt the performance. Based on our analysis, we propose *GraphNorm* to address the issue of InstanceNorm with a learnable shift (Step 2 in Figure 1). The learnable shift could learn to control the ideal amount of information to preserve for mean statistics. Together, GraphNorm normalizes the hidden representations across nodes in each individual graph with a learnable shift to avoid the expressiveness degradation while inheriting the acceleration effect of the shift operation.

We validate the effectiveness of GraphNorm on eight popular graph classification benchmarks. Empirical results confirm that GraphNorm consistently improves the speed of converge and stability of training for GNNs compared to those with BatchNorm, InstanceNorm, LayerNorm, and those without normalization. Furthermore, GraphNorm helps GNNs achieve better generalization performance on most benchmarks.

1.1. Related Work

Closely related to our work, InstanceNorm (Ulyanov et al., 2016) is originally proposed for real-time image generation. Variants of InstanceNorm are also studied in permutation equivalent data processing (Yi et al., 2018; Sun et al., 2020). We instead adapt InstanceNorm to GNNs and find it helpful for the training of GNNs. Our proposed GraphNorm builds on and improves InstanceNorm by addressing its expressiveness degradation with a learnable shift.

Few works have studied normalization in the GNN literature. Xu et al. (2019) adapts BatchNorm to GIN as a plug-in component. A preliminary version of Dwivedi et al. (2020) normalizes the node features with respect to the graph size. Our GraphNorm is size-agnostic and significantly differs from the graph size normalization. More discussions on other normalization methods are in Appendix E.

The reason behind the effectiveness of normalization has been intensively studied. While scale and shift are the main components of normalization, most existing works focus on the scale operation and the “scale-invariant” property: With a normalization layer after a linear (or convolutional) layer, the output values remain the same as the weights are scaled. Hence, normalization decouples the optimization of direction and length of the parameters (Kohler et al., 2019), implicitly tunes the learning rate (Ioffe & Szegedy, 2015; Hoffer et al., 2018; Arora et al., 2018b; Li & Arora, 2019), and smooths the optimization landscape (Santurkar et al., 2018). Our work offers a different view by instead showing specific *shift* operation has the preconditioning effect and can accelerate the training of GNNs.

2. Preliminaries

We begin by introducing our 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:

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

where $\mathcal{N}(v_i)$ is the set of nodes adjacent to v_i. Different GNNs can be obtained by choosing different AGGREGATE functions. Graph Convolutional Networks (GCN) (Kipf & Welling, 2017) can be defined in matrix form as:

$$H^{(k)} = \operatorname{ReLU}\left(W^{(k)} H^{(k-1)} Q_{\text{GCN}}\right), \tag{2}$$

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

Graph Isomorphism Network (GIN) (Xu et al., 2019) is defined in matrix form as

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

where MLP stands for multilayer perceptron, $\xi^{(k)}$ 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.

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 normalizes the feature values in the same channel across different samples in a batch. In NLP, LayerNorm normalizes the feature values at each position in a sequence separately.

3. Evaluating and Understanding Normalization for GNNs

In this section, we first adapt and evaluate existing normalization methods to GNNs. Then we give an explanation of the effectiveness of the variant of InstanceNorm, and show why the widely used BatchNorm fails to have such effectiveness. The understanding inspires us to develop better normalization methods, e.g., GraphNorm.

3.1. Adapting and Evaluating Normalization for GNNs

To investigate what normalization methods are effective for GNNs, we first adapt three typical normalization methods, i.e., BatchNorm, LayerNorm, and InstanceNorm, developed in other domain to GNNs. We apply the normalization after the linear transformation as in previous works (Ioffe & Szegedy, 2015; Xiong et al., 2020; Xu et al., 2019). The general GNN structure equipped with a normalization layer can be represented as:

$$H^{(k)} = F^{(k)}\left(\operatorname{Norm}\left(W^{(k)} H^{(k-1)} Q\right)\right), \tag{4}$$

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 can instantiate Eq. (4) 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. (2)), then Eq. (4) becomes GCN with normalization; Similarly, by setting $F^{(k)}$ to be $\text{MLP}^{(k)}$ and Q to be Q_{GIN} (Eq. (3)), we recover GIN with normalization.

We then describe the concrete operations of the adaptations of the normalization methods. Consider a batch of graphs $\{G_1, \cdots, G_b\}$ where b is the batch size. Let n_g be the number of nodes in graph G_g. We generally denote $\hat{h}_{i,j,g}$ as the inputs to the normalization module, e.g., the j-th feature value of node v_i of graph G_g, $i = 1, \cdots, n_g$, $j = 1, \cdots, d$, $g = 1, \cdots, b$. The adaptations take the general form:

$$\operatorname{Norm}\left(\hat{h}_{i,j,g}\right) = \gamma \cdot \frac{\hat{h}_{i,j,g} - \mu}{\sigma} + \beta, \tag{5}$$

where the scopes of mean μ, standard deviation σ, and affine

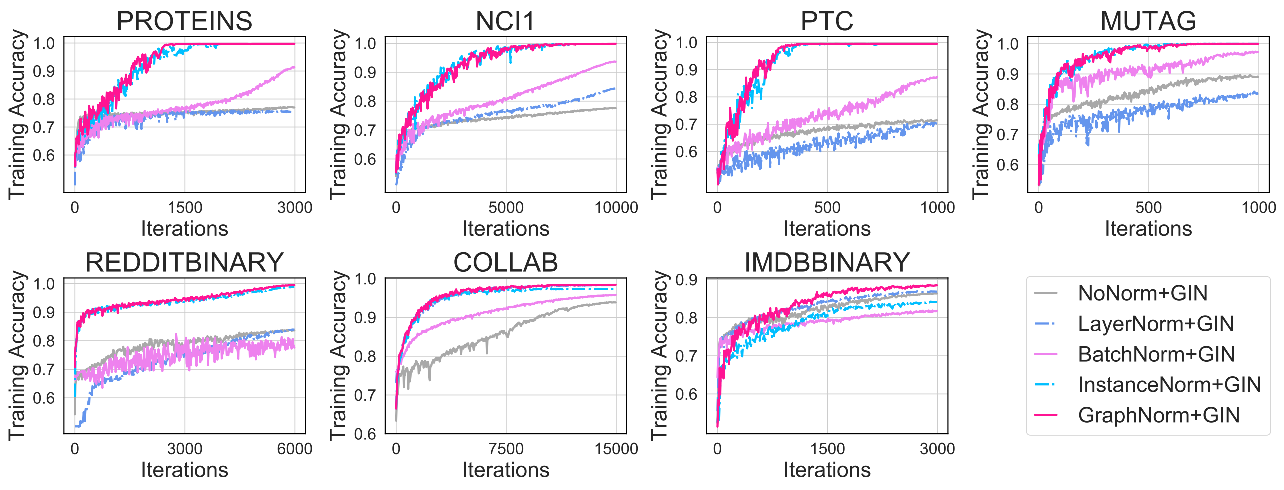

Figure 2: **Training performance** of GIN with different normalization methods and GIN without normalization in graph classification tasks. The convergence speed of our adaptation of InstanceNorm dominates BatchNorm and LayerNorm in most tasks. GraphNorm further improves the training over InstanceNorm especially on tasks with highly regular graphs, e.g., IMDB-BINARY (See Figure 5 for detailed illustration). Overall, GraphNorm converges faster than all other methods.

parameters γ, β differ for different normalization methods. For BatchNorm, normalization and the computation of μ and σ are applied to all values in the same feature dimension across the nodes of *all graphs in the batch* as in Xu et al. (2019), i.e., over dimensions g, i of $\hat{h}_{i,j,g}$. To adapt LayerNorm to GNNs, we view each node as a basic component, resembling words in a sentence, and apply normalization to all feature values across different dimensions of each node, i.e., over dimension j of $\hat{h}_{i,j,g}$. For InstanceNorm, we regard each graph as an instance. The normalization is then applied to the feature values across all nodes for each *individual graph*, i.e., over dimension i of $\hat{h}_{i,j,g}$.

In Figure 2 we show training curves of different normalization methods in graph classification tasks. We find that LayerNorm hardly improves the training process in most tasks, while our adaptation of InstanceNorm can largely boost the training speed compared to other normalization methods. The test performances have similar trends. We summarize the final test accuracies in Table 1. In the following subsections, we provide an explanation for the success of InstanceNorm and its benefits compared to BatchNorm, which is currently adapted in many GNNs.

3.2. Shift in InstanceNorm as a Preconditioner

As mentioned in Section 1.1, the scale-invariant property of the normalization has been investigated and considered as one of the ingredients that make the optimization efficient. In our analysis of normalizations for GNNs, we instead take a closer look at the *shift* operation in the normalization. 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 neighbors, see Eq. (1). Such uniqueness of GNNs makes it possible to study how the shift operation interplays with the graph data in detail.

We show that the shift operation in our adaptation of InstanceNorm serves as a preconditioner of the aggregation in GNNs and hypothesize this preconditioning effect can boost the training of GNNs. Though the current theory of deep learning has not been able to prove and compare the convergence rate in the real settings, we calculate the convergence rate of GNNs on a simple but fully characterizable setting to give insights on the benefit of the shift operation.

fWe first formulate our adaptation of InstanceNorm in the matrix form. 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]

$$\text{Norm}\left(W^{(k)}H^{(k-1)}Q\right) = S\left(W^{(k)}H^{(k-1)}Q\right)N, \quad (6)$$

where $S = \text{diag}\left(\frac{1}{\sigma_1}, \frac{1}{\sigma_2}, \cdots, \frac{1}{\sigma_{d^{(k)}}}\right)$ is the scaling, and Q is the GNN aggregation matrix. 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 the matrix form, shifting feature values on a single graph is equivalent to multiplying N as in Eq. (6). Therefore, we further check

[1]Standard normalization has an additional affine operation after shifting and scaling. Here we omit it in Eq. 6 for better demonstration. Adding this operation will not affect the theoretical analysis.

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. (6), $0 \leq \lambda_1 \leq \cdots \leq \lambda_n$ 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*

$$\lambda_1 \leq \mu_1 \leq \lambda_2 \leq \cdots \leq \lambda_{n-1} \leq \mu_{n-1} \leq \lambda_n, \quad (7)$$

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 $\mathbf{1}^\top \alpha_i = 0$.

The proof can be found in Appendix A.1.

We hypothesize that precoditioning Q can help the optimization. In the case of optimizing the weight matrix $W^{(k)}$, we can see from Eq. (6) 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.5 for more discussions. Similar preconditioning effects are believed to improve the training of deep learning models (Duchi et al., 2011; Kingma & Ba, 2015), and classic wisdom in optimization has also shown that preconditioning can accelerate the convergence of iterative methods (Axelsson, 1985; Demmel, 1997). Unfortunately, current theoretical toolbox only has a limited power on the optimization of deep learning models. Global convergence rates have only been proved for either simple models, e.g., linear models (Arora et al., 2018a), or extremely overparameterized models (Du et al., 2018; Allen-Zhu et al., 2019; Du et al., 2019a; Cai et al., 2019; Du et al., 2019b; Zou et al., 2020). To support our hypothesis that preconditioning may suggest better training, we investigate a simple but characterizable setting of training a linear GNN using gradient descent in Appendix A.2. In this setting, we prove that:

Proposition 3.1 (Concrete Example Showing Shift can Accelerate Training (Informal))**.** *With high probability over randomness of data generation, the parameter $\mathbf{w}_t^{\text{Shift}}$ of the model with shift at step t converges to the optimal parameter $\mathbf{w}_*^{\text{Shift}}$ linearly:*

$$\left\|\mathbf{w}_t^{\text{Shift}} - \mathbf{w}_*^{\text{Shift}}\right\|_2 = O\left(\rho_1^t\right),$$

where ρ_1 is the convergence rate.

Similarly, the parameter $\mathbf{w}_t^{\text{Vanilla}}$ of the vanilla model converges linearly, but with a slower rate:

$$\left\|\mathbf{w}_t^{\text{Vanilla}} - \mathbf{w}_*^{\text{Vanilla}}\right\|_2 = O\left(\rho_2^t\right) \text{ and } \rho_1 < \rho_2,$$

which indicates that the model with shift converges faster than the vanilla model.

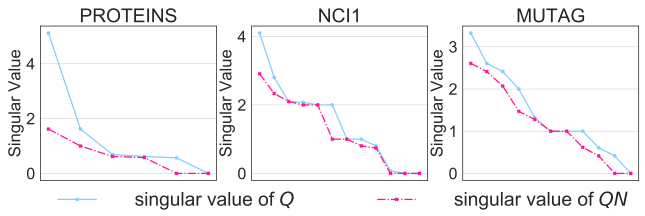

Figure 3: **Singular value distribution** of Q and QN for sampled graphs in different datasets using GIN. More visualizations can be found in Appendix D.1

The proof can be found in Appendix A.2. 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 3 (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.

3.3. Heavy Batch Noise in Graphs Makes BatchNorm Less Effective

The above analysis shows the adaptation of InstanceNorm has the effect of preconditioning the aggregation of GNNs. Then a natural question is whether a batch-level normalization for GNNs (Xu et al., 2019) has similar advantages. We show that BatchNorm is less effective in GNNs due to heavy batch noise on graph data.

In BatchNorm, the mean μ_B and standard deviation σ_B are calculated in a sampled batch during training, which can be viewed as random variables by the randomness of sampling. During testing, the estimated dataset-level statistics (running mean μ_D and standard deviation σ_D) are used instead of the batch-level statistics (Ioffe & Szegedy, 2015). To apply Theorem 3.1 to BatchNorm for the preconditioning effect, one could potentially view all graphs in a dataset as subgraphs in a *super graph*. Hence, Theorem 3.1 applies to BatchNorm if the batch-level statistics are well-concentrated around dataset-level statistics, i.e., $\mu_B \approx \mu_D$ and $\sigma_B \approx \sigma_D$. However, the concentration of batch-level statistics is heavily *domain-specific*. While Shen et al. (2020) find the variation of batch-level statistics in typical networks is small for computer vision, the concentration of batch-level statistics is still unknown for GNNs.

We study how the batch-level statistics μ_B, σ_B deviate from the dataset-level statistics μ_D, σ_D. For comparison, we train a 5-layer GIN with BatchNorm on the PROTEINS dataset and train a ResNet18 (He et al., 2016) on the CIFAR10

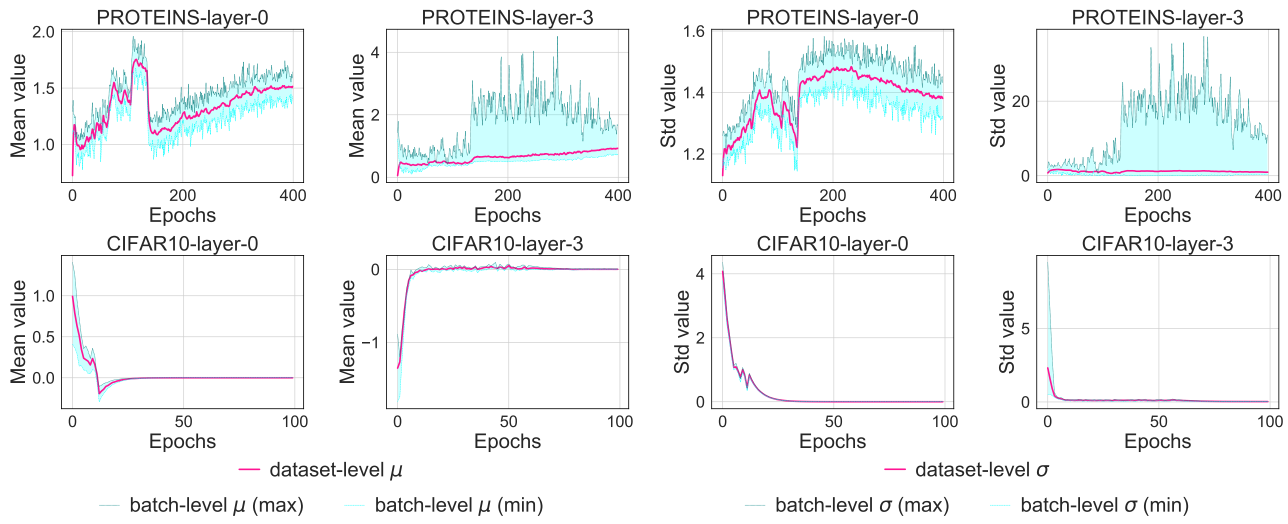

Figure 4: **Batch-level statistics are noisy for GNNs.** We plot the batch-level/dataset-level mean/standard deviation of models trained on PROTEINS (graph classification) and CIFAR10 (image classification). We observe that the deviation of batch-level statistics from dataset-level statistics is rather large for the graph task, while being negligible in image task.

dataset. We set batch size to 128. For each epoch, we record the batch-level max/min mean and standard deviation for the first and the last BatchNorm layer on a randomly selected dimension across batches. In Figure 4, pink line denotes the dataset-level statistics, and green/blue line denotes the max/min value of the batch-level statistics. We observe that for image tasks, the maximal deviation of the batch-level statistics from the dataset-level statistics is negligible (Figure 4) after a few epochs. In contrast, for the graph tasks, the variation of batch-level statistics stays large during training. Intuitively, the graph structure can be quite diverse and the a single batch cannot well represent the entire dataset. Hence, the preconditioning property also may not hold for BatchNorm. In fact, the heavy batch noise may bring instabilities to the training. More results may be found in Appendix D.2.

4. Graph Normalization

Although we provide evidence on the indispensability and advantages of our adaptation of InstanceNorm, 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 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. (6). 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, $\text{Norm}\left(W^{(1)}H^{(0)}Q_{\text{GIN}}\right) = S\left(W^{(1)}H^{(0)}Q_{\text{GIN}}\right)N = 0$, 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, $Q_{\text{GIN}}N = \xi^{(k)}N$, 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 by substituting Q_{GIN} with Q_{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 normaliza-

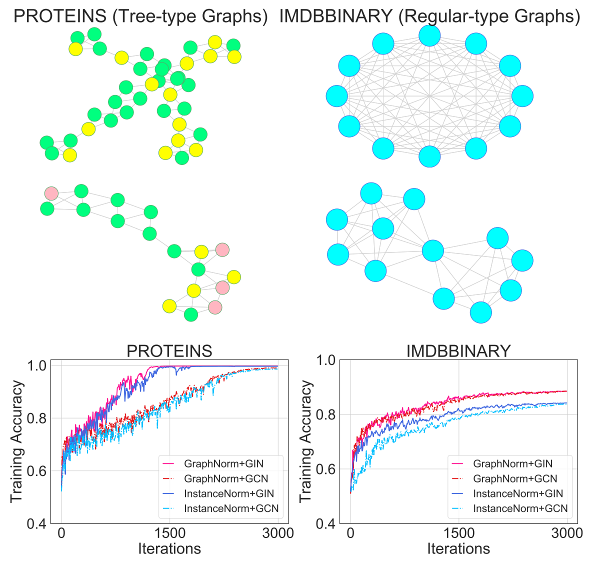

Figure 5: **Comparison of GraphNorm and InstanceNorm on different types of graphs.** Top: Sampled graphs with different topological structures. Bottom: Training curves of GIN/GCN using GraphNorm and InstanceNorm.

tion method with a *learnable parameter* to automatically control how much the mean to preserve in the shift operation. Combined with the graph-wise normalization, we name our new method Graph Normalization, i.e., 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:

$$\text{GraphNorm}\left(\hat{h}_{i,j}\right) = \gamma_j \cdot \frac{\hat{h}_{i,j} - \alpha_j \cdot \mu_j}{\hat{\sigma}_j} + \beta_j, \qquad (8)$$

where $\mu_j = \frac{\sum_{i=1}^n \hat{h}_{i,j}}{n}$, $\hat{\sigma}_j^2 = \frac{\sum_{i=1}^n \left(\hat{h}_{i,j} - \alpha_j \cdot \mu_j\right)^2}{n}$, 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.

To validate our theory and the proposed GraphNorm in real-world data, we conduct an ablation study on two typical datasets, PROTEINS and IMDB-BINARY. As shown in Figure 5, the graphs from PROTEINS and IMDB-BINARY exhibit irregular-type and regular-type graphs, respectively. We train GIN/GCN using our adaptation of InstanceNorm and GraphNorm under the same setting in Section 5. The training curves are presented in Figure 5. The curves show that using a learnable α slightly improves the convergence on PROTEINS, while significantly boost the training on IMDB-BINARY. This observation verify that shifting the feature values by subtracting the mean may lose information, especially for regular graphs. And the introduction of learnable shift in GraphNorm can effectively mitigate the expressive degradation.

5. Experiments

In this section, we evaluate and compare both the training and test performance of GraphNorm with other normalization methods on graph classification benchmarks.

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) (Hu et al., 2020). Dataset statistics are summarized in Table 1. We use two typical graph neural networks GIN (Xu et al., 2019) and GCN (Kipf & Welling, 2017) for our evaluations. 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.

Results. We plot the training curves of GIN with GraphNorm and other normalization methods[2] on different tasks in Figure 2. The results on GCN show similar trends, and are provided in Appendix D.3. As shown in Figure 2, GraphNorm enjoys the fastest convergence on all tasks. Compared to BatchNorm used in Xu et al. (2019), GraphNorm converges in roughly 5000/500 iterations on NCI1 and PTC datasets, while the model using BatchNorm does not even converge in 10000/1000 iterations. Remarkably, though InstanceNorm does *not* outperform other normalization methods on IMDB-BINARY, GraphNorm with learnable shift significantly boosts the training upon InstanceNorm and achieves the fastest convergence. We also validate the test performance and report the test accuracy in Table 1,2. The results show that GraphNorm also improves the generalization on most benchmarks.

5.1. Ablation Study

In this subsection, we summarize the results of some ablation studies. Due to the space limitation, the detailed results can be found in Appendix D.

[2]The graph size normalization in the preliminary version of Dwivedi et al. (2020) does not show significant improvement on the training and test performance, so we do not report it.

Table 1: **Test performance** of GIN/GCN with various normalization methods on graph classification 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	**86.0 ± 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+LAYERNORM	82.4 ± 6.4	62.8 ± 9.3	76.2 ± 3.0	$78.3 \pm 1{,}7$	$74.5 \pm 4{,}4$	82.8 ± 7.7	80.1 ± 0.8
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	**80.2 ± 1.9**
GIN+INSTANCENORM	90.5 ± 7.8	64.7 ± 5.9	76.5 ± 3.9	81.2 ± 1.8	74.8 ± 5.0	93.2 ± 1.7	80.0 ± 2.1
GIN+GraphNorm	**91.6 ± 6.5**	**64.9 ± 7.5**	**77.4 ± 4.9**	81.4 ± 2.4	**76.0 ± 3.7**	**93.5 ± 2.1**	**80.2 ± 1.0**

Table 2: **Test performance** on OGB.

Datasets	OGBG-MOLHIV
# graphs	41,127
# classes	2
Avg # nodes	25.5
GCN (Hu et al., 2020)	76.06 ± 0.97
GIN (Hu et al., 2020)	75.58 ± 1.40
GCN+LayerNorm	75.04 ± 0.48
GCN+BatchNorm	76.22 ± 0.95
GCN+InstanceNorm	78.18 ± 0.42
GCN+GraphNorm	**78.30 ± 0.69**
GIN+LayerNorm	74.79 ± 0.92
GIN+BatchNorm	76.61 ± 0.97
GIN+InstanceNorm	77.54 ± 1.27
GIN+GraphNorm	**77.73 ± 1.29**

BatchNorm with learnable shift. We conduct experiments on BatchNorm to investigate whether simply introducing a learnable shift can already improve the existing normalization methods without concrete motivation of overcoming expressiveness degradation. Specifically, we equip BatchNorm with a similar learnable shift as GraphNorm and evaluate its performance. We find that the learnable shift cannot further improve upon BatchNorm (See Appendix D), which suggests the introduction of learnable shift in GraphNorm is critical.

BatchNorm with running statistics. We study the variant of BatchNorm which uses running statistics to replace the batch-level mean and standard deviation (Similar idea is also proposed in Yan et al. (2019)). At first glance, this method may seem to be able to mitigate the problem of large batch noise. However, the running statistics change a lot during training, and using running statistics disables the model to back-propagate the gradients through mean and standard deviation. Results in Appendix D show this variant has even worse performance than BatchNorm.

The effect of batch size. We further compare the GraphNorm with BatchNorm with different batch sizes (8, 16, 32, 64). As shown in Appendix D, our GraphNorm consistently outperforms the BatchNorm on all the settings.

6. Conclusion and Future Work

In this paper, we adapt and evaluate three well-used normalization methods, i.e., BatchNorm, LayerNorm, and InstanceNorm to GNNs. We give explanations for the successes and failures of these adaptations. Based on our understanding of the strengths and limitations of existing adaptations, we propose Graph Normalization, that builds upon the adaptation of InstanceNorm with a learnable shift to overcome the expressive degradation of the original InstanceNorm. Experimental results show GNNs with GraphNorm not only converge faster, but also achieve better generalization performance on several benchmark datasets.

Though seeking theoretical understanding of normalization methods in deep learning is challenging (Arora et al., 2018b) due to limited understanding on the optimization of deep learning models and characterization of real world data, we take an initial step towards finding effective normalization methods for GNNs with theoretical guidance in this paper. The proposed theories and hypotheses are motivated by several simple models. And we are not able to give concrete theoretical results to problems such as: the convergence rate of general GNNs with normalization, the spectrum of Q normalized by learnable shift, etc. We believe the analyses of more realistic but complicated settings, e.g., the dynamics of GraphNorm on deep GNNs, are good future directions.

References

Allen-Zhu, Z., Li, Y., and Song, Z. A convergence theory for deep learning via over-parameterization. In *International Conference on Machine Learning*, pp. 242–252. PMLR, 2019.

Arora, S., Cohen, N., Golowich, N., and Hu, W. A convergence analysis of gradient descent for deep linear neural networks. In *International Conference on Learning Representations*, 2018a.

Arora, S., Li, Z., and Lyu, K. Theoretical analysis of auto rate-tuning by batch normalization. *arXiv preprint arXiv:1812.03981*, 2018b.

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

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

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

Bruna, J., Zaremba, W., Szlam, A., and LeCun, Y. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*, 2013.

Cai, T., Gao, R., Hou, J., Chen, S., Wang, D., He, D., Zhang, Z., and Wang, L. A gram-gauss-newton method learning overparameterized deep neural networks for regression problems. *arXiv preprint arXiv:1905.11675*, 2019.

Chen, Y., Tang, X., Qi, X., Li, C.-G., and Xiao, R. Learning graph normalization for graph neural networks, 2020.

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

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

Du, S., Lee, J., Li, H., Wang, L., and Zhai, X. Gradient descent finds global minima of deep neural networks. In *International Conference on Machine Learning*, pp. 1675–1685. PMLR, 2019a.

Du, S. S., Zhai, X., Poczos, B., and Singh, A. Gradient descent provably optimizes over-parameterized neural networks. In *International Conference on Learning Representations*, 2018.

Du, S. S., Hou, K., Salakhutdinov, R. R., Poczos, B., Wang, R., and Xu, K. Graph neural tangent kernel: Fusing graph neural networks with graph kernels. In *Advances in Neural Information Processing Systems*, pp. 5724–5734, 2019b.

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

Dwivedi, V. P., Joshi, C. K., Laurent, T., Bengio, Y., and Bresson, X. Benchmarking graph neural networks. *arXiv preprint arXiv:2003.00982*, 2020.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. Neural message passing for quantum chemistry. In *International Conference on Machine Learning*, pp. 1273–1272, 2017.

Gori, M., Monfardini, G., and Scarselli, F. A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 2, pp. 729–734. IEEE, 2005.

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

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

Hoffer, E., Banner, R., Golan, I., and Soudry, D. Norm matters: efficient and accurate normalization schemes in deep networks. In *Advances in Neural Information Processing Systems*, pp. 2160–2170, 2018.

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

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

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

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

Keriven, N. and Peyré, G. Universal invariant and equivariant graph neural networks. In *Advances in Neural Information Processing Systems*, pp. 7092–7101, 2019.

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

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

Kohler, J., Daneshmand, H., Lucchi, A., Hofmann, T., Zhou, M., and Neymeyr, K. 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.

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

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

Loukas, A. How hard is to distinguish graphs with graph neural networks? In *Advances in neural information processing systems*, 2020.

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

Monti, F., Boscaini, D., Masci, J., Rodola, E., Svoboda, J., and Bronstein, M. M. 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.

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

Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., and Chen, X. Improved techniques for training gans. In *Advances in neural information processing systems*, pp. 2234–2242, 2016.

Santoro, A., Raposo, D., Barrett, D. G., Malinowski, M., Pascanu, R., Battaglia, P., and Lillicrap, T. A simple neural network module for relational reasoning. In *Advances in neural information processing systems*, pp. 4967–4976, 2017.

Santurkar, S., Tsipras, D., Ilyas, A., and Madry, A. How does batch normalization help optimization? In *Advances in Neural Information Processing Systems*, pp. 2483–2493, 2018.

Sato, R., Yamada, M., and Kashima, H. Approximation ratios of graph neural networks for combinatorial problems. In *Advances in Neural Information Processing Systems*, pp. 4083–4092, 2019.

Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., and Monfardini, G. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2008.

Scarselli, F., Tsoi, A. C., and Hagenbuchner, M. The vapnik–chervonenkis dimension of graph and recursive neural networks. *Neural Networks*, 108:248–259, 2018.

Shen, S., Yao, Z., Gholami, A., Mahoney, M. W., and Keutzer, K. Powernorm: Rethinking batch normalization in transformers, 2020.

Shervashidze, N., Schweitzer, P., Leeuwen, E. J. v., Mehlhorn, K., and Borgwardt, K. M. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(Sep):2539–2561, 2011.

Stokes, J. M., Yang, K., Swanson, K., Jin, W., Cubillos-Ruiz, A., Donghia, N. M., MacNair, C. R., French, S., Carfrae, L. A., Bloom-Ackerman, Z., et al. A deep learning approach to antibiotic discovery. *Cell*, 180(4):688–702, 2020.

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

Sun, W., Jiang, W., Trulls, E., Tagliasacchi, A., and Yi, K. M. Acne: Attentive context normalization for robust permutation-equivariant learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11286–11295, 2020.

Ulyanov, D., Vedaldi, A., and Lempitsky, V. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022*, 2016.

Velickovic, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., and Bengio, Y. Graph attention networks. 2018.

Wainwright, M. J. *High-dimensional statistics: A non-asymptotic viewpoint*, volume 48. Cambridge University Press, 2019.

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

Wu, Z., Ramsundar, B., Feinberg, E. N., Gomes, J., Geniesse, C., Pappu, A. S., Leswing, K., and Pande, V. S. Moleculenet: A benchmark for molecular machine learning. *CoRR*, abs/1703.00564, 2017. URL http://arxiv.org/abs/1703.00564.

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., and Philip, S. Y. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

Xiong, R., Yang, Y., He, D., Zheng, K., Zheng, S., Xing, C., Zhang, H., Lan, Y., Wang, L., and Liu, T.-Y. On layer normalization in the transformer architecture. *arXiv preprint arXiv:2002.04745*, 2020.

Xu, K., Li, C., Tian, Y., Sonobe, T., Kawarabayashi, K.-i., and Jegelka, S. Representation learning on graphs with jumping knowledge networks. In *International Conference on Machine Learning*, pp. 5453–5462, 2018.

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

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

Xu, K., Zhang, M., Li, J., Du, S. S., Kawarabayashi, K.-I., and Jegelka, S. How neural networks extrapolate: From feedforward to graph neural networks. In *International Conference on Learning Representations*, 2021.

Yan, J., Wan, R., Zhang, X., Zhang, W., Wei, Y., and Sun, J. Towards stabilizing batch statistics in backward propagation of batch normalization. In *International Conference on Learning Representations*, 2019.

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

Yang, C., Wang, R., Yao, S., Liu, S., and Abdelzaher, T. Revisiting" over-smoothing" in deep gcns. *arXiv preprint arXiv:2003.13663*, 2020.

Yi, K. M., Trulls, E., Ono, Y., Lepetit, V., Salzmann, M., and Fua, P. Learning to find good correspondences. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2666–2674, 2018.

Ying, R., He, R., Chen, K., Eksombatchai, P., Hamilton, W. L., and Leskovec, J. Graph convolutional neural networks for web-scale recommender systems. In *kdd*, pp. 974–983, 2018.

Zhang, M., Cui, Z., Neumann, M., and Chen, Y. An end-to-end deep learning architecture for graph classification. pp. 4438–4445, 2018.

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

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

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

Zhou, K., Dong, Y., Lee, W. S., Hooi, B., Xu, H., and Feng, J. Effective training strategies for deep graph neural networks, 2020a.

Zhou, K., Huang, X., Li, Y., Zha, D., Chen, R., and Hu, X. Towards deeper graph neural networks with differentiable group normalization. *arXiv preprint arXiv:2006.06972*, 2020b.

Zou, D., Hu, Z., Wang, Y., Jiang, S., Sun, Y., and Gu, Q. Layer-dependent importance sampling for training deep and large graph convolutional networks. In *Advances in Neural Information Processing Systems*, pp. 11249–11259, 2019.

Zou, D., Cao, Y., Zhou, D., and Gu, Q. Gradient descent optimizes over-parameterized deep relu networks. *Machine Learning*, 109(3):467–492, 2020.

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*

$$\lambda_1 \le \mu_1 \le \lambda_2 \le \cdots \le \lambda_{n-1} \le \mu_{n-1} \le \lambda_n, \tag{9}$$

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

$$U = \begin{pmatrix} U_1 & \frac{1}{\sqrt{n}}\mathbf{1} \end{pmatrix}, \tag{10}$$

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

Let

$$D = \operatorname{diag}(1, \cdots, 1, 0) = \begin{pmatrix} I_{n-1} & \mathbf{0} \\ \mathbf{0}^\top & 0 \end{pmatrix}, \tag{11}$$

$$B = \begin{pmatrix} I_{n-1} \\ \mathbf{0}^\top \end{pmatrix}, \tag{12}$$

$$\bar{C} = Q^\top Q, \tag{13}$$

where $\mathbf{0} = \begin{bmatrix} \underbrace{0, \cdots, 0}_{\times n-1} \end{bmatrix}^\top$.

We have

$$NQ^\top QN \sim DU^\top \bar{C} U D \tag{14}$$

$$= D \begin{pmatrix} U_1^\top \\ \frac{1}{\sqrt{n}}\mathbf{1}^\top \end{pmatrix} \bar{C} \begin{pmatrix} U_1 & \frac{1}{\sqrt{n}}\mathbf{1} \end{pmatrix} D \tag{15}$$

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

$$= \begin{pmatrix} B^\top \\ \mathbf{0}^\top & 0 \end{pmatrix} \begin{pmatrix} U_1^\top \bar{C} U_1 & \frac{1}{\sqrt{n}} U_1^\top \bar{C} \mathbf{1} \\ \frac{1}{\sqrt{n}}\mathbf{1}^\top \bar{C} U_1 & \frac{1}{n}\mathbf{1}^\top \bar{C}\mathbf{1} \end{pmatrix} \begin{pmatrix} B & \mathbf{0} \\ & 0 \end{pmatrix} \tag{17}$$

$$= \begin{pmatrix} U_1^\top \bar{C} U_1 & \mathbf{0} \\ \mathbf{0}^\top & 0 \end{pmatrix}. \tag{18}$$

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 $\mu_1^2 \le \mu_2^2 \le \cdots \le \mu_{n-1}^2$ and $\mu_n^2 = 0$, and by Eq. (18), 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 $\lambda_1^2 \le \lambda_2^2 \le \cdots \le \lambda_n^2$ (By similarity of $U^\top \bar{C} U$ and $\bar{C}$), we then have

$$\lambda_1^2 \le \mu_1^2 \le \lambda_2^2 \le \cdots \le \lambda_{n-1}^2 \le \mu_{n-1}^2 \le \lambda_n^2. \tag{19}$$

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

$$U_1^\top \bar{C} U_1 z = \mu z, \tag{20}$$

$$\mathbf{1}^\top \bar{C} U_1 z = 0, \tag{21}$$

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. (21) 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

$$\bar{C} U_1 z = U_1 y. \tag{22}$$

Substituting this into Eq. (20), we have

$$U_1^\top U_1 y = \mu z. \tag{23}$$

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

$$y = \mu z, \tag{24}$$

which means

$$\bar{C} U_1 z = U_1 y = \mu U_1 z, \tag{25}$$

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. $\square$

A.2. Concrete example of the acceleration

To get more intuition on how the preconditioning effect of the shift can accelerate the training of GNNs, we provide a concrete example showing that shift indeed improves the convergence rate. Note that the global convergence rate of widely-used deep GNNs on general data remains highly unexplored, and the existing works mainly focus on some simplified case, e.g., GNTK (Du et al., 2019b). To make things clear without loss of intuition, we focus on a *simple linear GNN* applied to a *well-specified task* where we are able to explicitly compare the convergence rates.

A.2.1. SETTINGS

Data. We describe each sample, i.e., graph, with n nodes by a tuple $G = \{X, Q, \mathbf{p}, y\}$, where

- $X \in \mathbb{R}^{d\times n}$ is the feature matrix of the graph, where d is the dimension of the of each feature vector.
- $Q \in \mathbb{R}^{n\times n}$ representing the matrix representing the neighbor aggregation as Eq. (4). Note that this matrix depends on the aggregation scheme used by the chosen architecture, but for simplicity, we model this as a part of data structure.
- $\mathbf{p} \in \mathbb{R}^{n\times 1}$ is a weight vector representing the importance of each node. This will be used to calculate the READOUT step. Note that this vector is not provided in many real-world datasets, so the READOUT step usually takes operations such as summation.
- $y \in \mathbb{R}$ is the label.

The whole dataset $S = \{G_1, \cdots, G_m\}$ consists of m graphs where $G_i = \{X_i, Q_i, \mathbf{p}_i, y_i\}$. We make the following assumptions on the data generation process:

Assumption 1 (Independency). *We assume X_i, Q_i, $\mathbf{p}_i$ are drawn from three independent distributions in an i.i.d. manner, e.g., $X_1, \cdots, X_m$ are i.i.d..*

Assumption 2 (Structure of data distributions). *For clearness and simplicity of statement, we assume the number of nodes in each graph G_i are the same, we will use n to denote this number and we further assume $n = d$. We assume that the distribution of $\mathbf{p}_i$ satisfies $\mathbb{E}\left[\mathbf{p}\mathbf{p}^\top\right] = I_n, \mathbb{E}\mathbf{p} = 0$, which means the importance vector is non-degenerate. Let $\mathbb{E}XQ = Y$, we assume that Y is* full ranl. *We make the following assumptions on XQ: $\mathbf{1}^\top Y^{-1}XQ = 0$, which ensures that there is no information in the direction $\mathbf{1}^\top Y^{-1}$; there is a constant δ_1 such that $\mathbb{E}(XQ-Y)(XQ-Y)^\top \preceq \delta_1 I_d$ and $\mathbb{E}(XQ-Y)N(XQ-Y)^\top \preceq \delta_1 I_d$, where δ_1 characterizes the noise level; none of the eigenvectors of $YY^\top$ is orthogonal to $\mathbf{1}$.*

Remark 1. *A few remarks are in order, firstly, the assumption that each graph has the same number of nodes and the number n is equal to feature dimension d can be achieved by "padding", i.e., adding dummy points or features to the graph or the feature matrix. The assumption that $\mathbf{1}^\top Y^{-1}XQ = 0$ is used to guarantee that there is no information loss caused by shift ($\mathbf{1}^\top Y^{-1}YNY^\top = 0$). Though we make this strong assumption to ensure no information loss in theoretical part, we introduce "learnable shift" to mitigate this problem in the practical setting. The theory taking learnable shift into account is an interesting future direction.*

Assumption 3 (Boundness). *We make the technical assumption that there is a constant b such that the distributions of $X_i, Q_i, \mathbf{p}_i$ ensures*

$$\|X_i\|\,\|Q_i\|\,\|\mathbf{p}_i\| \leq \sqrt{b}. \tag{26}$$

Model. We consider a simple *linear graph neural network* with parameter $\mathbf{w} \in \mathbb{R}^{d\times 1}$:

$$f^{\text{Vanilla}}_{\mathbf{w}}(X, Q, \mathbf{p}) = \mathbf{w}^\top XQ\mathbf{p}. \tag{27}$$

Then, the model with shift can be represented as:

$$f^{\text{Shift}}_{\mathbf{w}}(X, Q, \mathbf{p}) = \mathbf{w}^\top XQN\mathbf{p}, \tag{28}$$

where $N = I_n - \frac{1}{n}\mathbf{1}\mathbf{1}^\top$.

Criterion. We consider using square loss as training objective, i.e.,

$$L(f) = \sum_{i=1}^{m} \frac{1}{2}\left(f(X_i, Q_i, \mathbf{p}_i) - y_i\right)^2. \tag{29}$$

Algorithm. We consider using gradient descent to optimize the objective function. Let the initial parameter $\mathbf{w}_0 = 0$. The update rule of w from step t to $t+1$ can be described as:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \nabla_{\mathbf{w}} L(f_{\mathbf{w}_t}), \tag{30}$$

where η is the learning rate.

Theorem A.1. *Under Assumption 1,2,3, for any $\epsilon > 0$ there exists constants C_1, C_2, such that for $\delta_1 < C_1, m > C_2$, with probability $1-\epsilon$, the parameter $\mathbf{w}^{\text{Vanilla}}_t$ of vanilla model converges to the optimal parameter $\mathbf{w}^{\text{Vanilla}}_*$ linearly:*

$$\left\|\mathbf{w}^{\text{Vanilla}}_t - \mathbf{w}^{\text{Vanilla}}_*\right\|_2 \leq O\left(\rho_1^t\right), \tag{31}$$

while the parameter $\mathbf{w}^{\text{Shfit}}_t$ of the shifted model converges to the optimal parameter $\mathbf{w}^{\text{Shfit}}_*$ linearly:

$$\left\|\mathbf{w}^{\text{Shift}}_t - \mathbf{w}^{\text{Shfit}}_*\right\|_2 \leq O\left(\rho_2^t\right), \tag{32}$$

where

$$1 > \rho_1 > \rho_2, \tag{33}$$

which indicates the shifted model has a faster convergence rate.

Proof. We firstly reformulate the optimization problem in matrix form.

Notice that in our linear model, the representation and structure of a graph $G_i = \{X_i, Q_i, \mathbf{p}_i, y_i\}$ can be encoded as a whole in a single vector, i.e., $\mathbf{z}_i^{\text{Vanilla}} = X_i Q_i \mathbf{p}_i \in \mathbb{R}^{d\times 1}$ for vanilla model in Eq. (27), and $\mathbf{z}_i^{\text{Shift}} = X_i Q_i N \mathbf{p}_i \in \mathbb{R}^{d\times 1}$ for shifted model in Eq. (28). We call $\mathbf{z}_i$ and $\mathbf{z}_i^{\text{Shift}}$ "combined features". Let $Z^{\text{Vanilla}} = \left[\mathbf{z}_1^{\text{Vanilla}}, \cdots, \mathbf{z}_m^{\text{Vanilla}}\right] \in \mathbb{R}^{d\times m}$ and $Z^{\text{Shift}} = \left[\mathbf{z}_1^{\text{Shift}}, \cdots, \mathbf{z}_m^{\text{Shift}}\right] \in \mathbb{R}^{d\times m}$ be the matrix of combined features of valinna linear model and shifted linear model respectively. For clearness of the proof, we may abuse the notations and use Z to represent Z^{Vanilla}. Then the objective in Eq. (29) for vanilla linear model can be reformulated as:

$$L(f_{\mathbf{w}}) = \frac{1}{2}\left\|Z^\top \mathbf{w} - \mathbf{y}\right\|_2^2, \tag{34}$$

where $\mathbf{y} = [y_1, \cdots, y_m]^\top \in \mathbb{R}^{m\times 1}$.

Then the gradient descent update can be explicitly writen as:

$$\begin{aligned}\mathbf{w}_{t+1} &= \mathbf{w}_t - \eta\left(ZZ^\top \mathbf{w}_t - Z\mathbf{y}\right) &(35)\\ &= (I_d - \eta ZZ^\top)\mathbf{w}_t + \eta Z\mathbf{y}, &(36)\end{aligned}$$

which converges to $\mathbf{w}_* = \left(ZZ^\top\right)^\dagger Z\mathbf{y}$ according to classic theory of least square problem (Horn & Johnson, 2012), where $\left(ZZ^\top\right)^\dagger$ is the Moore–Penrose inverse of $ZZ^\top$.

By simultaneously subtracting $\mathbf{w}_*$ in the update rule, we have

$$\mathbf{w}_{t+1} - \mathbf{w}_* = \left(I_d - \eta ZZ^\top\right)(\mathbf{w}_t - \mathbf{w}_*). \tag{37}$$

So the residual of $\mathbf{w}_t$ is

$$\begin{aligned}\|\mathbf{w}_t - \mathbf{w}_*\| &= \left\|\left(I_d - \eta ZZ^\top\right)^t \mathbf{w}_*\right\| &(38)\\ &\le \left\|I_d - \eta ZZ^\top\right\|^t \|\mathbf{w}_*\|. &(39)\end{aligned}$$

Let $\sigma_{\max}(A)$ and $\sigma_{\min}(A)$ be the maximal and mininal *positive* eigenvalues of A, respectively. Then the optimial learning rate (the largest learning rate that ensures $I_d - \eta ZZ^\top$ is positive semidefinite) is $\eta = \frac{1}{\sigma_{\max}(ZZ^\top)}$. Under this learning rate we have the convergence rate following Eq. (39):

$$\begin{aligned}\|\mathbf{w}_t - \mathbf{w}_*\| &\le \left\|I_d - \eta ZZ^\top\right\|^t \|\mathbf{w}_*\| &(40)\\ &\le \left(1 - \frac{\sigma_{\min}\left(ZZ^\top\right)}{\sigma_{\max}\left(ZZ^\top\right)}\right)^t \|\mathbf{w}_*\|. &(41)\end{aligned}$$

For now, we show that the convergence rate of the optimization problem with vanilla model depends on $\frac{\sigma_{\min}\left(ZZ^\top\right)}{\sigma_{\max}\left(ZZ^\top\right)}$. Follwing the same argument, we can show the convergence rate of the optimization problem with shifted model depends on $\frac{\sigma_{\min}\left(Z^{\text{Shift}}Z^{\text{Shfit}\top}\right)}{\sigma_{\max}\left(Z^{\text{Shift}}Z^{\text{Shfit}\top}\right)}$. We then aim to bound this term, which we call effective condition number.

Similarly, we investigate the effective condition number for $ZZ^\top$ first, and the analysis of $Z^{\text{Shift}}Z^{\text{Shift}\top}$ follows the same manner. As multiplying a constant does not affect the effective condition number, we first scale $ZZ^\top$ by $\frac{1}{m}$ and expand it as:

$$\frac{1}{m}ZZ^\top = \frac{1}{m}\sum_{i=1}^m \mathbf{z}_i\mathbf{z}_i^\top, \tag{42}$$

which is the empirical estimation of the covariance matrix of the combined feature. By concentration inequality, we know this quantity is concentrated to the covariance matrix, i.e.,

$$\begin{aligned}\mathbb{E}_{\mathbf{z}}\mathbf{z}\mathbf{z}^\top &= \mathbb{E}_{X,Q,\mathbf{p}}XQ\mathbf{p}\left(XQ\mathbf{p}\right)^\top\\ &= \mathbb{E}_{X,Q}XQ\left(\mathbb{E}\left[\mathbf{p}\mathbf{p}^\top\right]\right)(XQ)^\top\\ &= \mathbb{E}_{X,Q}XQ(XQ)^\top \quad \text{(By Assumption 1)}\\ &= YY^\top + \mathbb{E}_{X,Q}(XQ-Y)(XQ-Y)^\top.\end{aligned}$$

Noticing that $\mathbf{0} \preceq \mathbb{E}_{X,Q}(XQ-Y)(XQ-Y)^\top \preceq \delta_1 I_d$ by Assumption 2, and Y is full rank, we can conclude that $\sigma_{\max}\left(YY^\top\right) \le \sigma_{\max}\left(\mathbb{E}_{\mathbf{z}}\mathbf{z}\mathbf{z}^\top\right) \le \sigma_{\max}\left(YY^\top\right) + \delta_1$, and $\sigma_{\min}\left(YY^\top\right) \le \sigma_{\min}\left(\mathbb{E}_{\mathbf{z}}\mathbf{z}\mathbf{z}^\top\right) \le \sigma_{\min}\left(YY^\top\right) + \delta_1$ by Weyl's inequality.

By similar argument, we have that $\frac{1}{m}Z^{\text{Shift}}Z^{\text{Shift}\top}$ concentrates to

$$\begin{aligned}&\mathbb{E}_{\mathbf{z}^{\text{Shift}}}\mathbf{z}^{\text{Shift}}\mathbf{z}^{\text{Shift}\top}\\ =&\mathbb{E}_{X,Q}(XQ)N^2(XQ)^\top\\ =&\mathbb{E}_{X,Q}(XQ)N(XQ)^\top \quad (N^2 = N)\\ =&YNY^\top + \mathbb{E}_{X,Q}(XQ-Y)N(XQ-Y)^\top.\end{aligned}$$

By Assumption 2, we have

$$\begin{aligned}0 =&\mathbf{1}^\top Y^{-1}\mathbb{E}_{\mathbf{z}^{\text{Shift}}}\mathbf{z}^{\text{Shift}}\mathbf{z}^{\text{Shift}\top}\\ =&\mathbf{1}^\top Y^{-1}\left(YNY^\top + \mathbb{E}_{X,Q}(XQ-Y)N(XQ-Y)^\top\right)\\ =&\mathbf{1}^\top Y^{-1}\mathbb{E}_{X,Q}(XQ-Y)N(XQ-Y)^\top,\end{aligned}$$

which means $\mathbb{E}_{X,Q}(XQ-Y)N(XQ-Y)^\top$ has the same eigenspace as $YNY^\top$ with respect to eigenvalue 0. Combining with $\mathbf{0} \preceq \mathbb{E}_{X,Q}(XQ-Y)N(XQ-Y)^\top \preceq \delta_1 I_d$, we have $\sigma_{\max}\left(YNY^\top\right) \le \sigma_{\max}\left(\mathbb{E}_{\mathbf{z}^{\text{Shift}}}\mathbf{z}^{\text{Shift}}\mathbf{z}^{\text{Shift}\top}\right) \le \sigma_{\max}\left(YNY^\top\right) + \delta_1$, and $\sigma_{\min}\left(YNY^\top\right) \le \sigma_{\min}\left(\mathbb{E}_{\mathbf{z}^{\text{Shift}}}\mathbf{z}^{\text{Shift}}\mathbf{z}^{\text{Shift}\top}\right) \le \sigma_{\min}\left(YNY^\top\right) + \delta_1$.

It remains to bound the finite sample error, i.e., $\left\|\frac{1}{m}ZZ^\top - \mathbb{E}_{\mathbf{z}}\mathbf{z}\mathbf{z}^\top\right\|_2$ and $\left\|\frac{1}{m}Z^{\text{Shift}}Z^{\text{Shfit}\top} - \mathbb{E}_{\mathbf{z}}\mathbf{z}\mathbf{z}^\top\right\|_2$. These bounds can be obtained by the following lemma:

Lemma A.2 (Corollary 6.1 in Wainwright (2019))**.** *Let $\mathbf{z}_1, \cdots, \mathbf{z}_m$ be i.i.d. zero-mean random vectors with covariance matrix Σ such that $\|\mathbf{z}\|_2 \leq \sqrt{b}$ almost surely. Then for all $\delta > 0$, the sample covariance matrix $\hat{\Sigma} = \frac{1}{m}\sum_{i=1}^m \mathbf{z}_i\mathbf{z}_i^\top$ satisfies*

$$\Pr\left[\left\|\hat{\Sigma} - \Sigma\right\|_2 \geq \delta\right] \leq 2d\exp\left(-\frac{\delta^2}{2b\left(\|\Sigma\|_2 + \delta\right)}\right). \tag{43}$$

By this lemma, we further have

Lemma A.3 (Bound on the sample covariance matrix)**.** *Let $\mathbf{z}_1, \cdots, \mathbf{z}_m$ be i.i.d. zero-mean random vectors with covariance matrix Σ such that $\|\mathbf{z}\|_2 \leq \sqrt{b}$ almost surely. Then with probability $1-\epsilon$, the sample covariance matrix $\hat{\Sigma} = \frac{1}{m}\sum_{i=1}^m \mathbf{z}_i\mathbf{z}_i^\top$ satisfies*

$$\left\|\hat{\Sigma} - \Sigma\right\|_2 \leq O\left(\sqrt{\frac{\log(1/\epsilon)}{m}}\right), \tag{44}$$

where we hide constants $b, \|\Sigma\|_2, d$ in the big-O notation and highlight the dependence on the number of samples m.

Combining with previous results, we conclude that:

$$\begin{aligned}
&\sigma_{\max}\left(YY^\top\right) - O\left(\sqrt{\frac{\log(1/\epsilon)}{m}}\right)\\
\leq&\sigma_{\max}\left(\frac{1}{m}ZZ^\top\right)\\
\leq&\sigma_{\max}\left(YY^\top\right) + \delta_1 + O\left(\sqrt{\frac{\log(1/\epsilon)}{m}}\right);
\end{aligned}$$

$$\begin{aligned}
&\sigma_{\min}\left(YY^\top\right) - O\left(\sqrt{\frac{\log(1/\epsilon)}{m}}\right)\\
\leq&\sigma_{\min}\left(\frac{1}{m}ZZ^\top\right)\\
\leq&\sigma_{\min}\left(YY^\top\right) + \delta_1 + O\left(\sqrt{\frac{\log(1/\epsilon)}{m}}\right);
\end{aligned}$$

$$\begin{aligned}
&\sigma_{\max}\left(YNY^\top\right) - O\left(\sqrt{\frac{\log(1/\epsilon)}{m}}\right)\\
\leq&\sigma_{\max}\left(\frac{1}{m}Z^{\text{Shift}}Z^{\text{Shift}\top}\right)\\
\leq&\sigma_{\max}\left(YNY^\top\right) + \delta_1 + O\left(\sqrt{\frac{\log(1/\epsilon)}{m}}\right)
\end{aligned}$$

$$\begin{aligned}
&\sigma_{\min}\left(YNY^\top\right) - O\left(\sqrt{\frac{\log(1/\epsilon)}{m}}\right)\\
\leq&\sigma_{\min}\left(\frac{1}{m}Z^{\text{Shift}}Z^{\text{Shift}\top}\right)\\
\leq&\sigma_{\min}\left(YNY^\top\right) + \delta_1 + O\left(\sqrt{\frac{\log(1/\epsilon)}{m}}\right).
\end{aligned}$$

By now, we have transfered the analysis of $ZZ^\top$ and $Z^{\text{Shift}}Z^{\text{Shfit}\top}$ to the analysis of $YY^\top$ and $YNY^\top$. And the positive eigenvalues of $YNY^\top$ is interlaced between the positive eigenvalues of $YY^\top$ by the same argument as Theorem 3.1. Concretely, we have $\sigma_{\min}\left(YY^\top\right) \leq \sigma_{\min}\left(YNY^\top\right) \leq \sigma_{\max}\left(YNY^\top\right) \leq \sigma_{\max}\left(YY^\top\right)$. Noticing that none of the eigenvectors of $YY^\top$ is orthogonal to $\mathbf{1}$, the first and last equalies can not be achieved, so $\sigma_{\min}\left(YY^\top\right) < \sigma_{\min}\left(YNY^\top\right) \leq \sigma_{\max}\left(YNY^\top\right) < \sigma_{\max}\left(YY^\top\right)$. Finally, we can conclude for small enough δ_1 and large enough m, with probability ϵ,

$$\begin{aligned}
&\sigma_{\min}\left(\frac{1}{m}ZZ^\top\right)\\
\leq&\sigma_{\min}\left(YY^\top\right) + \delta_1 + O\left(\sqrt{\frac{\log(1/\epsilon)}{m}}\right)\\
<&\sigma_{\min}\left(YNY^\top\right) - O\left(\sqrt{\frac{\log(1/\epsilon)}{m}}\right)\\
\leq&\sigma_{\min}\left(\frac{1}{m}Z^{\text{Shift}}Z^{\text{Shift}\top}\right)\\
\leq&\sigma_{\max}\left(\frac{1}{m}Z^{\text{Shift}}Z^{\text{Shift}\top}\right)\\
\leq&\sigma_{\max}\left(YNY^\top\right) + \delta_1 + O\left(\sqrt{\frac{\log(1/\epsilon)}{m}}\right)\\
<&\sigma_{\max}\left(YY^\top\right) - O\left(\sqrt{\frac{\log(1/\epsilon)}{m}}\right)\\
\leq&\sigma_{\max}\left(\frac{1}{m}ZZ^\top\right).
\end{aligned}$$

So

$$\begin{aligned}
&\rho_2 = 1 - \frac{\sigma_{\min}\left(Z^{\text{Shift}}Z^{\text{Shift}\top}\right)}{\sigma_{\max}\left(Z^{\text{Shift}}Z^{\text{Shift}\top}\right)}\\
<&\rho_1 = 1 - \frac{\sigma_{\min}\left(ZZ^\top\right)}{\sigma_{\max}\left(ZZ^\top\right)},
\end{aligned}$$

where ρ_1, ρ_2 are the constants in the statement of the theorem. This inequality means the shifted model has better convergence speed by Eq. (41). $\square$

A.3. 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

$$c \cdot \mathbf{1}^\top Q_{\text{GIN}} N = c \cdot \mathbf{1}^\top (r+1+\xi^{(1)}) I_n \left(I_n - \frac{1}{n}\mathbf{1}\mathbf{1}^\top\right) \tag{45}$$

$$= c\left(r+1+\xi^{(1)}\right)\left(\mathbf{1}^\top - \mathbf{1}^\top \cdot \frac{1}{n}\mathbf{1}\mathbf{1}^\top\right) = 0. \tag{46}$$

□

A.4. Proof of Proposition 4.2

Proof.

$$Q_{\text{GIN}} N = (A + I_n + \xi^{(k)} I_n) N == (\mathbf{1}\mathbf{1}^\top + \xi^{(k)I_n}) N = \xi^{(k)} N \tag{47}$$

□

A.5. 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

$$\frac{\partial \mathcal{L}}{\partial W^{(k)}} = \left(\left(H^{(k-1)} Q N\right)^\top \otimes S\right) \frac{\partial \mathcal{L}}{\partial Z^{(k)}}, \tag{48}$$

where $\otimes$ represents the Kronecker product, and thus $\left(H^{(k-1)} Q N\right)^\top \otimes S$ 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 3. 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.

Table 3: **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

Baselines. For the medium-scale bioinformatics and social network datasets, we compare several competitive baselines as in Xu et al. (2019), 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 2, 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 7.

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 2 (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 4 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 8. 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 9. 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 10. We can see that the observations are consistent and the batch statistics on graph data are noisy, as in Figure 4 in the main body.

D.3. Training Curves on GCN

As Figure 2 in the main body, we train GCNs with different normalization methods (GraphNorm, InstanceNorm, BatchNorm and LayerNorm) and GCN without normalization in graph classification tasks and plot the training curves in Figure 6. It is obvious that the GraphNorm also enjoys the fastest convergence on all tasks. Remarkably, GCN with InstanceNorm even underperforms GCNs with other normalizations, while our GraphNorm with learnable shift significantly boosts the training upon InstanceNorm and achieves the fastest convergence.

D.4. Further Results of Ablation Study

BatchNorm with learnable shift. We conduct experiments on BatchNorm to investigate whether simply introducing a learnable shift can already improve the existing normalization methods without concrete motivation of overcoming expressiveness degradation. Specifically, we equip BatchNorm with a similar learnable shift (α-BatchNorm for short) as GraphNorm and evaluate its performance. As shown in Figure 12, the α-BatchNorm cannot outperform the BatchNorm on the three datasets. Moreover, as shown in Figure 5 in the main body, the learnable shift significantly improve upon GraphNorm on IMDB-BINARY dataset, while it cannot further improve upon BatchNorm, which suggests the introduction of learnable shift in GraphNorm is critical.

BatchNorm with running statistics. We study the variant of BatchNorm which uses running statistics (MS-BatchNorm for short) to replace the batch-level mean and standard deviation (similar idea is also proposed in Yan et al. (2019)). At first glance, this method may seem to be able to mitigate the problem of large batch noise. However, the running statistics change a lot during training, and using running statistics disables the model to back-propagate the gradients through mean and standard deviation. Thus, we also train GIN with BatchNorm which stops the back-propagation of the graidients through mean and standard deviation (DT-BatchNorm for short). As shown in Figure 12, both the MS-BatchNorm and DT-BatchNorm underperform the BatchNorm by a large margin, which shows that the problem of the heavy batch noise cannot be mitigated by simply using the running statistics.

The effect of batch size. We further compare the GraphNorm and BatchNorm with different batch sizes (8, 16, 32, 64). As shown in Figure 11, our GraphNorm consistently outperforms the BatchNorm on all the settings.

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., 2020b;a) 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.

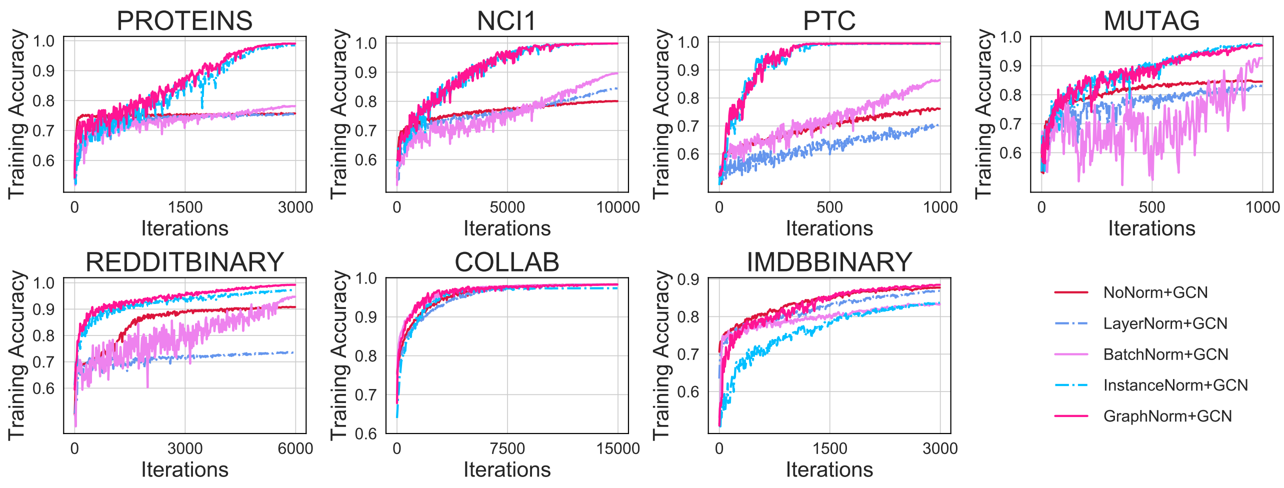

Figure 6: **Training performance** of GCN with different normalization methods and GCN without normalization in graph classification tasks.

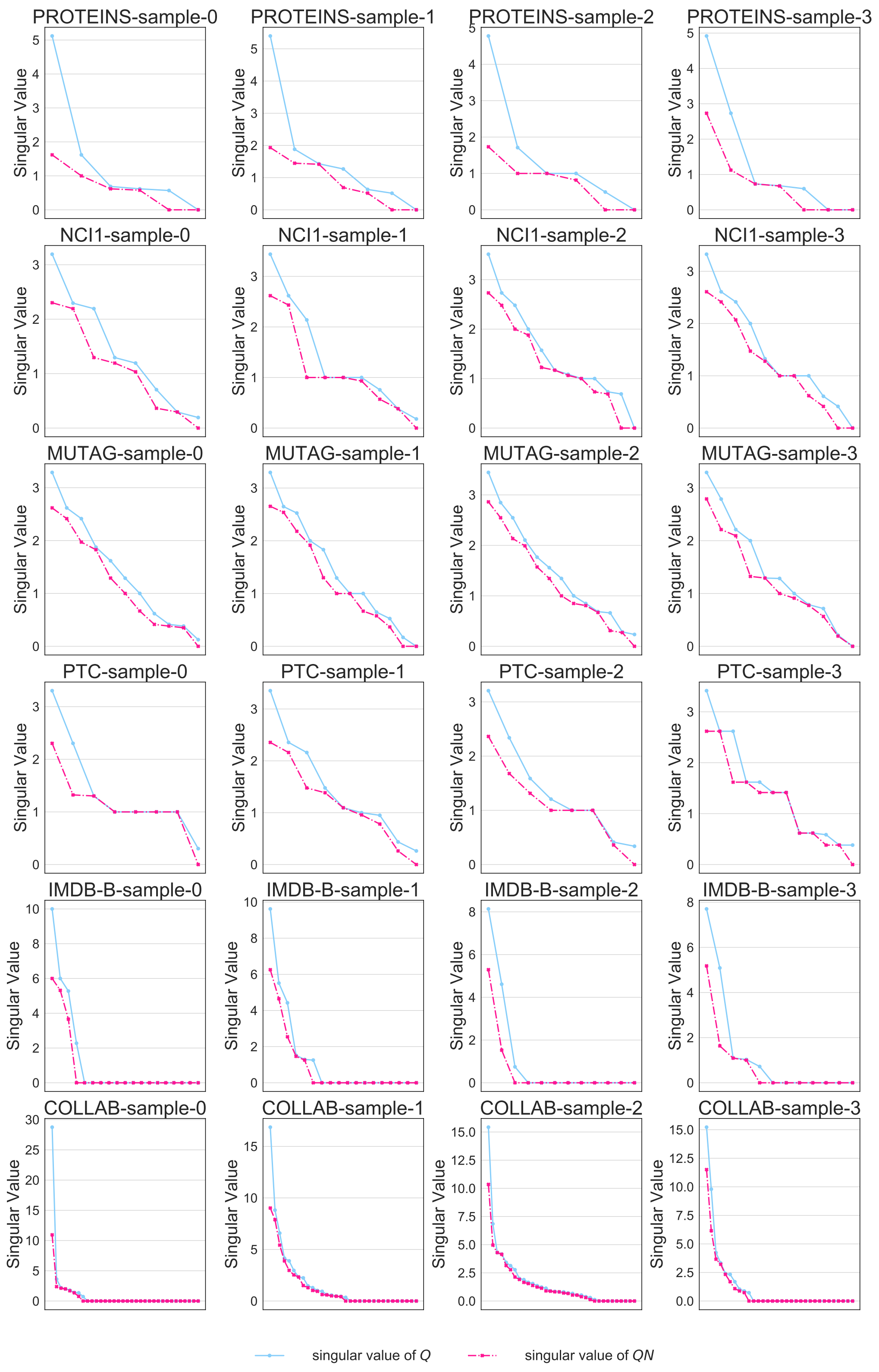

Figure 7: **Singular value distribution of Q and QN.** Graph samples from PROTEINS, NCI1, MUTAG, PTC, IMDB-BINARY, COLLAB are presented.

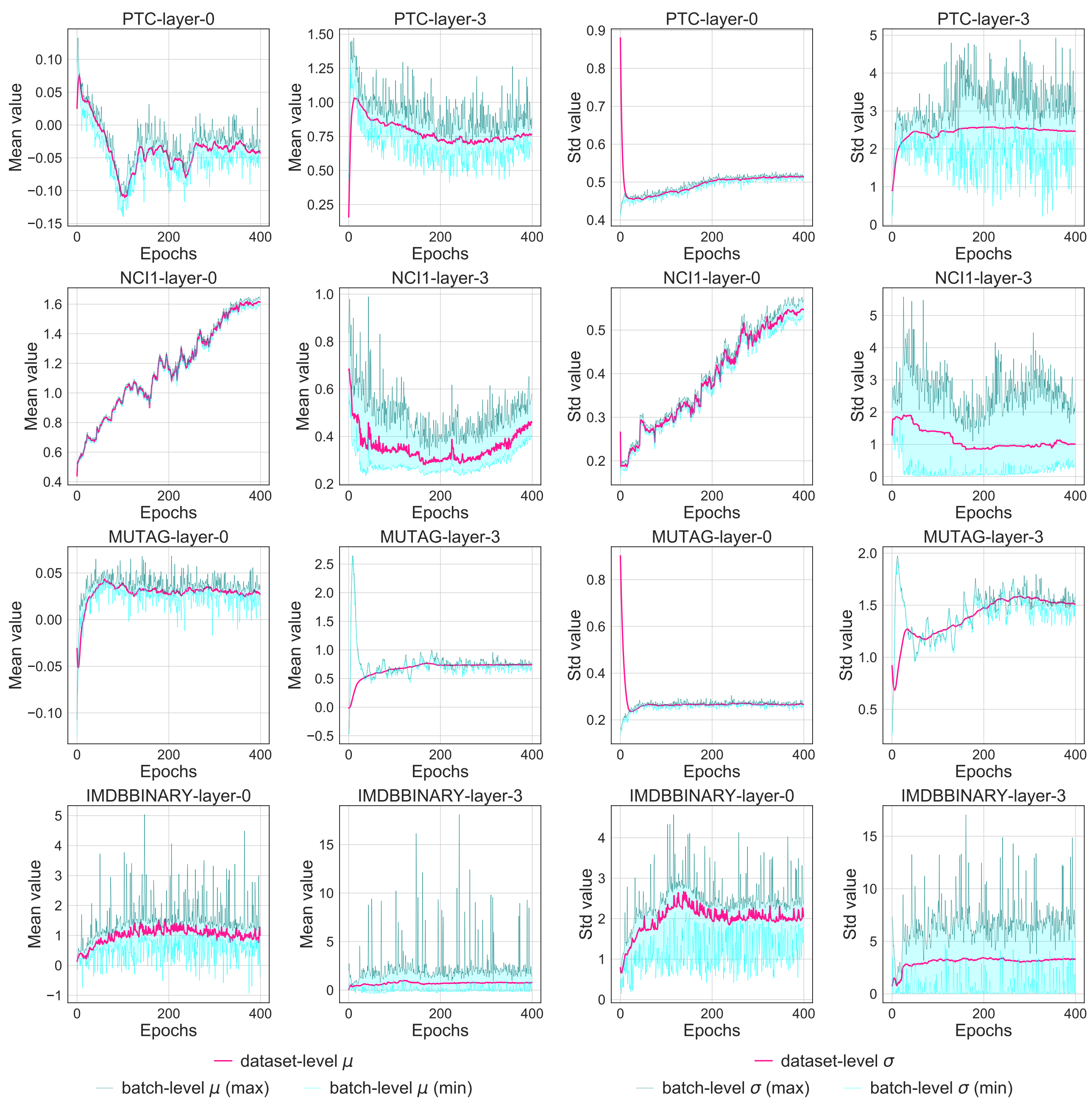

Figure 8: **Batch-level statistics are noisy for GNNs** (Examples from PTC, NCI1, MUTAG, IMDB-BINARY datasets). We plot the batch-level mean/standard deviation and dataset-level mean/standard deviation of the first (layer 0) and the last (layer 3) BatchNorm layers in different checkpoints. GIN with 5 layers is employed.

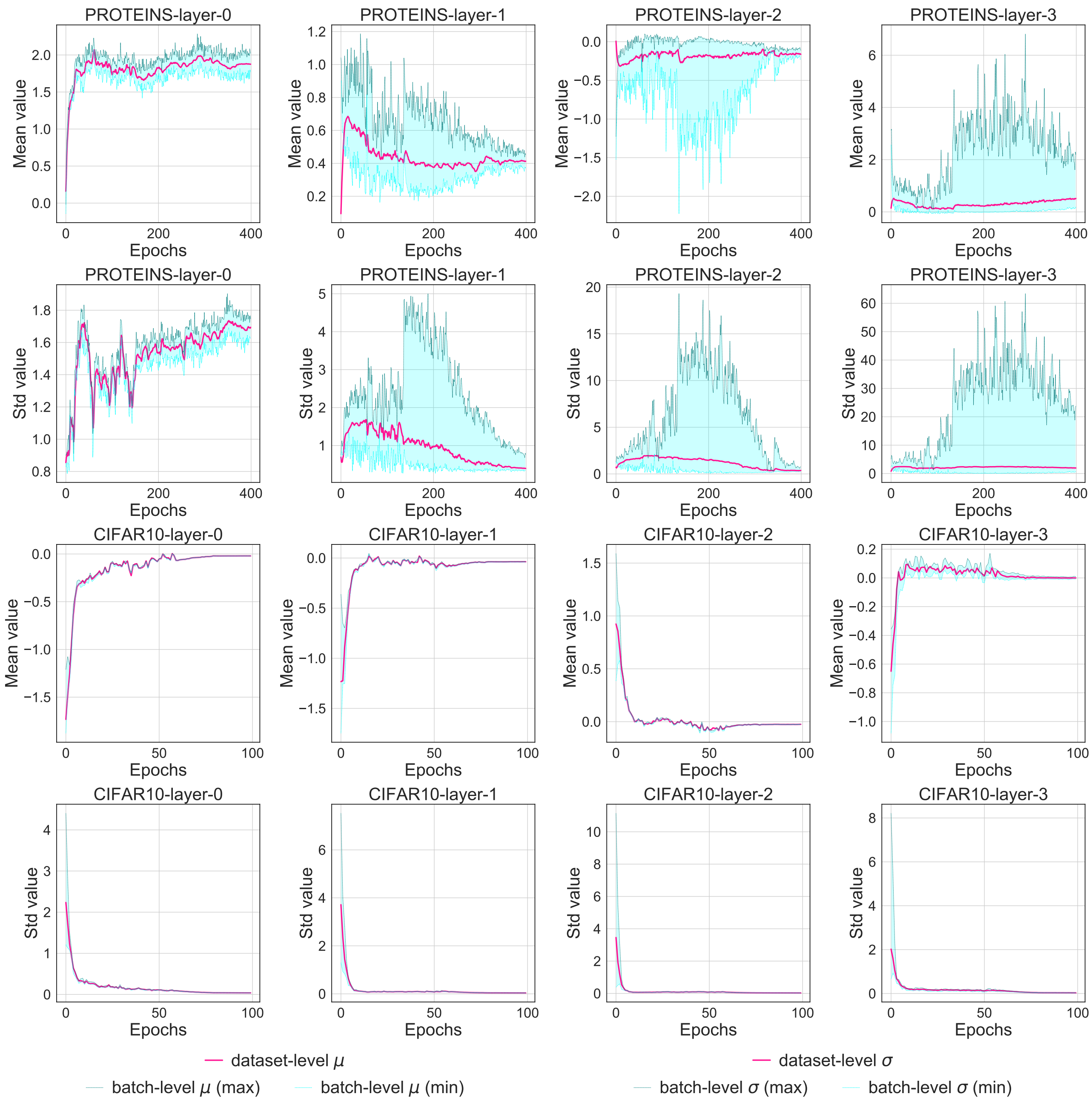

Figure 9: **Batch-level statistics are noisy for GNNs of different depth.** We plot the batch-level mean/standard deviation and dataset-level mean/standard deviation of different BatchNorm layers (from layer 0 to layer 3) in different checkpoints. We use a five-layer GIN on PROTEINS and ResNet18 on CIFAR10 for comparison.

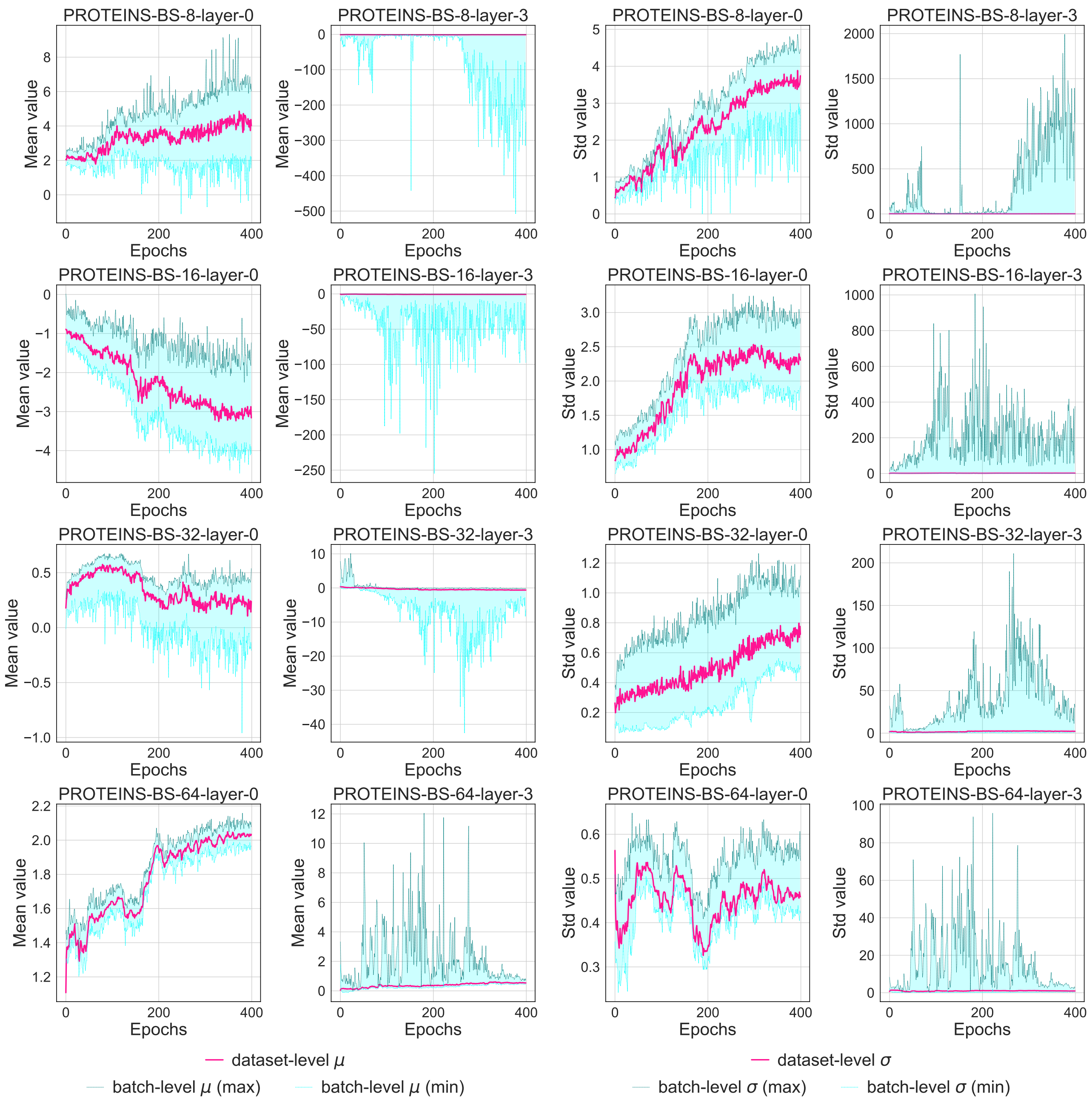

Figure 10: **Batch-level statistics are noisy for GNNs of different batch sizes.** We plot the batch-level mean/standard deviation and dataset-level mean/standard deviation of different BatchNorm layers (layer 0 and layer 3) in different checkpoints. Specifically, different batch sizes (8, 16, 32, 64) are chosed for comparison. GIN with 5 layers is employed.

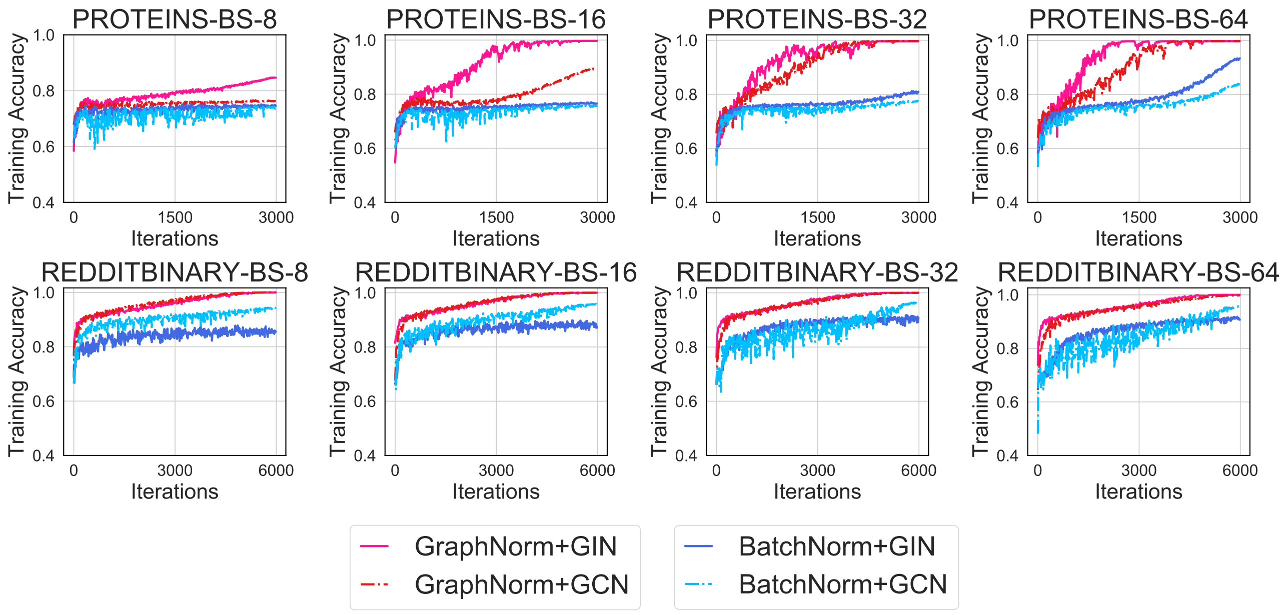

Figure 11: **Training performance** of GIN/GCN with GraphNorm and BatchNorm with batch sizes of (8, 16, 32, 64) on PROTEINS and REDDITBINARY datasets.

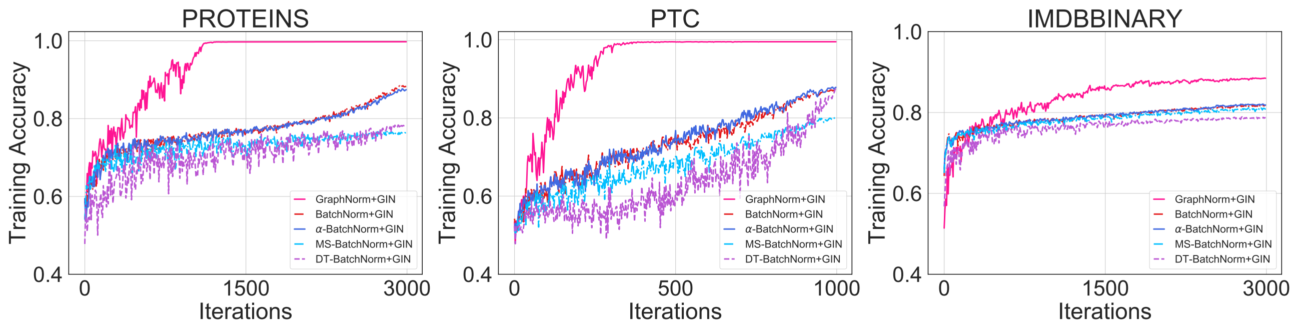

Figure 12: **Training performance** of GIN with GraphNorm and variant BatchNorms (α-BatchNorm, MS-BatchNorm and DT-BatchNorm) on PROTEINS, PTC and IMDB-BINARY datasets.