GraphNorm: A Principled Approach to Accelerating Graph Neural Network Training

Tianle Cai 1 2 Shengjie Luo 3 4 5 Keyulu Xu 6 Di He 7 Tie-Yan Liu 7 Liwei Wang 3 4

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; Ying et al., 2021). 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; Ying et al., 2021), 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
GraphNorm: A Principled Approach to Accelerating Graph Neural Network Training

**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 pre-conditioner 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).

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 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 graph as \( D \) is the identity matrix. Let the feature \( n \)-vector \( \{X_1, X_2, \cdots, X_n\} \) be the set \( \{v_1, v_2, \cdots, v_n\} \). Different GNNs can be obtained by choosing different AGGREGATE functions. Graph Convolutional Networks (GCN) (Kipf & Welling, 2017) can be defined in matrix form as:

\[
H^{(k)} = \text{ReLU} \left( W^{(k)} H^{(k-1)} Q_{\text{GCN}} \right),
\]

where \( \text{ReLU} \) 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

\[
H^{(k)} = \text{MLP}^{(k)} \left( W^{(k)} H^{(k-1)} Q_{\text{GIN}} \right),
\]

where \( \text{MLP} \) 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 \( h_G = \text{READOUT}\left( \{h_i^{(K)} \mid v_i \in V \} \right) \). A classifier can be applied upon \( h_G \) to predict the labels.

Normalizations. Generally, given a set of values \( \{x_1, x_2, \cdots, x_m\} \), a normalization operation first shifts each \( x_i \) by the mean \( \mu \), and then scales them down by standard deviation \( \sigma \): \( x_i \to \gamma \frac{x_i - \mu}{\sigma} + \beta \), where \( \gamma \) and \( \beta \) 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:

\[
H^{(k)} = F^{(k)} \left( \text{Norm} \left( W^{(k)} H^{(k-1)} Q \right) \right),
\]

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 \( \text{ReLU} \) and set \( Q \) to be \( Q_{\text{GCN}} \) (Eq. (2)), then Eq. (4) becomes GCN with normalization; Similarly, by setting \( F^{(k)} \) to be \( \text{MLP}^{(k)} \) and \( Q \) to be \( Q_{\text{GIN}} \) (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 \( h_{i,j,g} \)
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:

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

where the scopes of mean \( \mu \), standard deviation \( \sigma \), and affine parameters \( \gamma, \beta \) differ for different normalization methods. For BatchNorm, normalization and the computation of \( \mu \) and \( \sigma \) 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.

We first formulate our adaptation of InstanceNorm in the matrix form. Mathematically, for a graph of \( n \) nodes, denote \( N = I_n - \frac{1}{n} 1_n 1_n^\top \). \( N \) is the matrix form of the shift operation, i.e., for any vector \( z = [z_1, z_2, \cdots, z_n]^\top \in \mathbb{R}^n \), \( z^\top N = z^\top - \left( \frac{1}{n} \sum_{i=1}^n z_i \right) 1_n^\top \). Then the normalization together
with the aggregation can be represented as\footnote{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.}
\[
\text{Norm} \left( W^{(k)} H^{(k-1)} Q \right) = S \left( W^{(k)} H^{(k-1)} Q \right) N, \tag{6}
\]
where $S = \text{diag} \left( \frac{1}{\sigma_1}, \frac{1}{\sigma_2}, \ldots, \frac{1}{\sigma_d} \right)$ is the scaling, and $Q$ is the GNN aggregation matrix. Each $\sigma_i$ 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 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
\[
\lambda_1 \leq \mu_1 \leq \lambda_2 \leq \cdots \leq \lambda_{n-1} \leq \mu_{n-1} \leq \lambda_n, \tag{7}
\]
where $\lambda_i = \mu_i$ or $\lambda_i = \mu_{i-1}$ only if there exists one of the right singular vectors $\alpha_i$ of $Q$ associated with $\lambda_i$ satisfying $\alpha_i^T \alpha_i = 0$.

The proof can be found in Appendix A.1.

We hypothesize that preconditioning $Q$ 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 $w_t^{\text{Shift}}$ of the model with shift at step $t$ converges to the optimal parameter $w^*$ linearly:
\[
\| w_t^{\text{Shift}} - w^* \|_2 = O \left( \rho_1^t \right),
\]
where $\rho_1$ is the convergence rate.

Similarly, the parameter $w_t^{\text{Vanilla}}$ of the vanilla model converges linearly, but with a slower rate:
\[
\| w_t^{\text{Vanilla}} - w^* \|_2 = O \left( \rho_2^t \right) \quad \text{and} \quad \rho_1 < \rho_2,
\]
which indicates that the model with shift converges faster than the vanilla model.

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 $\mu_B$ and standard deviation $\sigma_B$ 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 $\mu_D$ and standard deviation $\sigma_D$) are used instead of the batch-level statistics (Ioffe & Szegedy, 2015). To apply Theorem 3.1 to BatchNorm for the preconditioning effect, one
GraphNorm: A Principled Approach to Accelerating Graph Neural Network Training

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.

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 $\mu_B, \sigma_B$ deviate from the dataset-level statistics $\mu_D, \sigma_D$. 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 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 one-hot encodings as its 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 in-
By introducing the learnable parameter $\alpha_j$ for each feature dimension $j$, we are able to learn how much the information we need to keep in the mean. It is easy to see that GraphNorm has stronger expressive power than InstanceNorm. Formally, we have the following fact:

**Fact 1** (GraphNorm is strictly more expressive than InstanceNorm). If $\alpha_j \neq 1, \gamma_j \neq 0$, then there does not exist $\gamma_j', \beta_j'$ such that for any $\{\tilde{h}_{i,j}\}_i$, the normalization is applied to, for any $i$, $\frac{\tilde{h}_{i,j} - \alpha_j \mu_j}{\sigma_j} + \beta_j = \gamma_j' \left( \frac{\tilde{h}_{i,j} - \alpha_j \mu_j}{\sigma_j} + \beta_j \right)$.

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 $\alpha$ slightly improves the convergence on PROTEINS, while significantly boosts the training on IMDB-BINARY. This observation verifies 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
GraphNorm: A Principled Approach to Accelerating Graph Neural Network Training

Table 1. Test performance of GIN/GCN with various normalization methods on graph classification tasks.

<table>
<thead>
<tr>
<th>Datasets</th>
<th>MUTAG</th>
<th>PTC</th>
<th>PROTEINS</th>
<th>NCI1</th>
<th>IMDB-B</th>
<th>RDT-B</th>
<th>COLLAB</th>
</tr>
</thead>
<tbody>
<tr>
<td># graphs</td>
<td>188</td>
<td>344</td>
<td>1113</td>
<td>4110</td>
<td>1000</td>
<td>2000</td>
<td>5000</td>
</tr>
<tr>
<td># classes</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Avg # nodes</td>
<td>17.9</td>
<td>25.5</td>
<td>39.1</td>
<td>29.8</td>
<td>19.8</td>
<td>429.6</td>
<td>74.5</td>
</tr>
<tr>
<td>WL SUBTREE (SHERVASHIDZE ET AL., 2011)</td>
<td>90.4 ± 5.7</td>
<td>59.9 ± 4.3</td>
<td>75.0 ± 3.1</td>
<td><strong>86.0 ± 1.8</strong></td>
<td>73.8 ± 3.9</td>
<td>81.0 ± 3.1</td>
<td>78.9 ± 1.9</td>
</tr>
<tr>
<td>DCNN (ATWOOD &amp; TOWSLEY, 2016)</td>
<td>67.0</td>
<td>56.6</td>
<td>61.3</td>
<td>62.6</td>
<td>49.1</td>
<td>-</td>
<td>52.1</td>
</tr>
<tr>
<td>DGCNN (ZHANG ET AL., 2018)</td>
<td>85.8</td>
<td>58.6</td>
<td>75.5</td>
<td>74.4</td>
<td>70.0</td>
<td>-</td>
<td>73.7</td>
</tr>
<tr>
<td>AWL (IVANOV &amp; BURNAEV, 2018)</td>
<td>87.9 ± 9.8</td>
<td>-</td>
<td>-</td>
<td>74.5 ± 5.9</td>
<td>87.9 ± 2.5</td>
<td>73.9 ± 1.9</td>
<td></td>
</tr>
<tr>
<td>GIN+LAYERNORM</td>
<td>82.4 ± 6.4</td>
<td>62.8 ± 9.3</td>
<td>76.2 ± 3.0</td>
<td>78.3 ± 1.7</td>
<td>74.5 ± 4.4</td>
<td>82.8 ± 7.7</td>
<td>80.1 ± 0.8</td>
</tr>
<tr>
<td>GIN+BATCHNORM (XU ET AL., 2019)</td>
<td>89.4 ± 5.6</td>
<td>64.6 ± 7.0</td>
<td>76.2 ± 2.8</td>
<td>82.7 ± 1.7</td>
<td>75.1 ± 5.1</td>
<td>92.4 ± 2.5</td>
<td><strong>80.2 ± 1.9</strong></td>
</tr>
<tr>
<td>GIN+INSTANCENORM</td>
<td>90.5 ± 7.8</td>
<td>64.7 ± 5.9</td>
<td>76.5 ± 3.9</td>
<td>81.2 ± 1.8</td>
<td>74.8 ± 5.0</td>
<td>93.2 ± 1.7</td>
<td>80.0 ± 2.1</td>
</tr>
<tr>
<td>GIN+GraphNorm</td>
<td><strong>91.6 ± 6.5</strong></td>
<td><strong>64.9 ± 7.5</strong></td>
<td><strong>77.4 ± 4.9</strong></td>
<td>81.4 ± 2.4</td>
<td><strong>76.0 ± 3.7</strong></td>
<td><strong>93.5 ± 2.1</strong></td>
<td><strong>80.2 ± 1.0</strong></td>
</tr>
</tbody>
</table>

Table 2. Test performance on OGB.

<table>
<thead>
<tr>
<th>Datasets</th>
<th>OGB-MOLHIV</th>
</tr>
</thead>
<tbody>
<tr>
<td># graphs</td>
<td>41,127</td>
</tr>
<tr>
<td># classes</td>
<td>2</td>
</tr>
<tr>
<td>Avg # nodes</td>
<td>25.5</td>
</tr>
<tr>
<td>GCN (Hu et al., 2020)</td>
<td>76.06 ± 0.97</td>
</tr>
<tr>
<td>GIN (Hu et al., 2020)</td>
<td>75.58 ± 1.40</td>
</tr>
<tr>
<td>GCN+LayerNorm</td>
<td>75.04 ± 0.48</td>
</tr>
<tr>
<td>GCN+BATCHNORM</td>
<td>76.22 ± 0.95</td>
</tr>
<tr>
<td>GCN+INSTANCENORM</td>
<td>78.18 ± 0.42</td>
</tr>
<tr>
<td>GCN+GraphNorm</td>
<td><strong>78.30 ± 0.69</strong></td>
</tr>
<tr>
<td>GIN+LayerNorm</td>
<td>74.79 ± 0.92</td>
</tr>
<tr>
<td>GIN+BATCHNORM</td>
<td>76.61 ± 0.97</td>
</tr>
<tr>
<td>GIN+INSTANCENORM</td>
<td>77.54 ± 1.27</td>
</tr>
<tr>
<td>GIN+GraphNorm</td>
<td><strong>77.73 ± 1.29</strong></td>
</tr>
</tbody>
</table>

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.

For reference, we explain the possible reasons of higher test accuracy in two folds. First, as shown in Figure 2, using proper normalization helps the model find a minimum with a higher training accuracy. Second, as suggested by Hardt et al. (2016), faster training leads to smaller generalization gap. Since the test accuracy equals the training accuracy plus the generalization, these two views together suggest better normalization leads to better test performance.

5.1. Ablation Study

In this subsection, we summarize the results of some ablation studies, including BatchNorm with learnable shift, BatchNorm with running statistics and the effect of batch size. Due to the space limitation, the detailed results can be found in Appendix D.

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

\(^2\)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.
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 Graph-
Norm 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 normal-
ization methods, i.e., BatchNorm, LayerNorm, and Instan-
cenorm to GNNs. We give explanations for the successes
and failures of these adaptations. Based on our understand-
ing of the strengths and limitations of existing adaptations,
we propose Graph Normalization, that builds upon the adap-
tation 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 per-
formance 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.

Acknowledgements

We thank Mozhi Zhang and Ruosong Wang for helpful
suggestions on the paper; Zhiyuan Li and Kaifeng Lyu
for helpful discussion on the literature of normalization
methods; and Prof. Yang Yuan for support of computational
resources. This work was supported by National Key R&D
Program of China (2018YFB1402600), Key-Area Research
and Development Program of Guangdong Province (No.
2019B121204008), BJNSF (L172037), Beijing Academy
of Artificial Intelligence, Project 2020BD006 supported by
PKU-Baidu Fund, NSF CAREER award (1553284) and
NSF III (1900933).

References

Allen-Zhu, Z., Li, Y., and Song, Z. A convergence theory for
depth 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 conver-
gence analysis of gradient descent for deep linear neural
networks. In International Conference on Learning Rep-
resentations, 2018a.

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

Atwood, J. and Towsley, D. Diffusion-convolutional neural
networks. In Advances in neural information processing

Axelsson, O. A survey of preconditioned iterative methods
for linear systems of algebraic equations. BIT Numerical


Bruna, J., Zaremba, W., Szlam, A., and LeCun, Y. Spec-
tral networks and locally connected networks on graphs.

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

Chen, Y., Tang, X., Qi, X., Li, C.-G., and Xiao, R. Learning

Defferrard, M., Bresson, X., and Vandergheynst, P. Con-
volutional neural networks on graphs with fast localized
spectral filtering. In Advances in neural information pro-

Demmel, J. W. Applied numerical linear algebra, vol-

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 Rep-
resentations, 2018.


GraphNorm: A Principled Approach to Accelerating Graph Neural Network Training


