LEARNING GRAPH NORMALIZATION FOR GRAPH NEURAL NETWORKS

Yihao Chen,[*] **Xin Tang**[*] , **Xianbiao Qi**[*]
Visual Computing Group, Ping An Property & Casualty Insurance
`o0o@o0oo0o.cc, tangxint@gmail.com, qixianbiao@gmail.com`

Chun-Guang Li
School of Artificial Intelligence, Beijing University of Posts and Telecommunications
`lichunguang@bupt.edu.cn`

Rong Xiao
Visual Computing Group, Ping An Property & Casualty Insurance
`xiaorong@gmail.com`

ABSTRACT

Graph Neural Networks (GNNs) have attracted considerable attention and have emerged as a new promising paradigm to process graph-structured data. GNNs are usually stacked to multiple layers and the node representations in each layer are computed through propagating and aggregating the neighboring node features with respect to the graph. By stacking to multiple layers, GNNs are able to capture the long-range dependencies among the data on the graph and thus bring performance improvements. To train a GNN with multiple layers effectively, some normalization techniques (e.g., node-wise normalization, batch-wise normalization) are necessary. However, the normalization techniques for GNNs are highly task-relevant and different application tasks prefer to different normalization techniques, which is hard to know in advance. To tackle this deficiency, in this paper, we propose to learn graph normalization by optimizing a weighted combination of normalization techniques at four different levels, including node-wise normalization, adjacency-wise normalization, graph-wise normalization, and batch-wise normalization, in which the adjacency-wise normalization and the graph-wise normalization are newly proposed in this paper to take into account the local structure and the global structure on the graph, respectively. By learning the optimal weights, we are able to automatically select a single best or a best combination of multiple normalizations for a specific task. We conduct extensive experiments on benchmark datasets for different tasks, including node classification, link prediction, graph classification and graph regression, and confirm that the learned graph normalization leads to competitive results and that the learned weights suggest the appropriate normalization techniques for the specific task. Source code is released here https://github.com/cyh1112/GraphNormalization.

1 INTRODUCTION

Graph Neural Networks (GNNs) have shown great popularity due to their efciency in learning on graphs for various application areas, such as natural language processing (Yao et al., 2019; Liu et al., 2019a; Zhang et al., 2018), computer vision (Wang et al., 2018; Li et al., 2020; Cheng et al., 2020), point cloud (Shi & Rajkumar, 2020; Liu et al., 2019b), drug discovery (Lim et al., 2019), citation networks (Kipf & Welling, 2016), social networks (Chen et al., 2018) and recommendation (Fan et al., 2019; Wu et al., 2019). A graph consists of nodes and edges, where nodes represent individual objects and edges represent relationships among those objects. In the GNN framework, the node or

---

[*]Equal Contribution.

Figure 1: Illustration for four normalization methods on graph. The input node representations are normalized on four levels. (a) Node-wise. (b) Adjacency-wise. (c) Graph-wise. and (d) Batch-wise. We also extend these four normalization methods to edge features as shown in (e), (f), (g), and (h).

edge representations are alternately updated by propagating information along the edges of a graph via non-linear transformation and aggregation functions (Wu et al., 2020; Zhang et al., 2018). GNN captures long-range node dependencies via stacking multiple message-passing layers, allowing the information to propagate for multiple-hops (Xu et al., 2018). Then, the node or edge representations can be used for downstream tasks such as node classification, link prediction, and graph regression and classification.

In essence, GNN is a new kind of neural networks which exploits neural network operations over graph structure. Among the numerous kinds of GNNs (Bruna et al., 2014; Defferrard et al., 2016; Chen et al., 2019; Maron et al., 2019; Xu et al., 2019), message-passing GNNs (Scarselli et al., 2009; Li et al., 2016; Kipf & Welling, 2016; Velickovic et al., 2018; Bresson & Laurent, 2017) have been the most widely used due to their ability to leverage the basic building blocks of deep learning such as batching, normalization and residual connections. To update the feature representation of a node, Kipf & Welling (2016) proposed a Graph ConvNets (GCN) to employ an averaging operation over the neighborhood node with the same weight value for each of its neighbors. GraphSage (Hamilton et al., 2017) samples a fixed-size neighborhood of each node and performs mean aggregator or LSTM-based aggregator over the neighbors. In Graph Attention Networks (GAT) (Velickovic et al., 2018), an attention mechanism is incorporated into the propagation step. It updates the feature representation of each code via a weighted sum of adjacent node representations. Monti et al. (2017) has present MoNet which designs a Gaussian kernel with learnable parameters to assign different weights to neighbors. GatedGCN (Bresson & Laurent, 2017) has achieved state-of-art results on many datasets (Dwivedi et al., 2020). GatedGCN explicitly introduces edge features at each layer and updates edge features by considering the feature representations of these two connected nodes of the edge. GatedGCN designs edge gates to assign different weights to different elements of each neighboring node representation. In addition, GatecGCN uses residual shortcut, batch normalization, and activation function to update the node representations. More details about GNNs are provided in Appendix A.

It is well accepted that normalization techniques (Ioffe & Szegedy, 2015; Ba et al., 2016; Wu & He, 2018; Ulyanov et al., 2016) are the critical ingredients to effectively train deep neural networks. Batch normalization (BN) (Ioffe & Szegedy, 2015) is widely used in training deep neural networks (He et al., 2016; Huang et al., 2017; Silver et al., 2017) to perform global normalization along the batch dimension. Hereafter various normalization methods have been developed from different perspectives. For instance, layer normalization (LN) (Ba et al., 2016) and group normalization (Wu & He, 2018) operate along the channel dimension, and instance normalization (Ulyanov et al., 2016) performs a BN-like normalization but only for a sample. A detailed introduction of BN and LN are provide in the Appendix B. In addition, Switchable Normalization (Luo et al., 2019) utilizes three distinct scopes to compute statistics (means and variances) including a channel, a layer, and a minibatch. In graph neural networks, (Dwivedi et al., 2020) utilizes BN for each graph propagation layer to train GNNs. Zhao & Akoglu (2020) introduces a novel normalization layer denoted as PAIRNORM to mitigate the over-smoothing problem, which prevents all node representations from

homogenization. PAIRNORM focuses on differentiating the distances between different node pairs. Unfortunately, it ignore the local neighbor structure and global graph structure. Each normalization method has its advantages and is suitable for some particular tasks. For instance, BN has achieved perfect performance in computer vision, however, LN outperforms BN in natural language processing (Vaswani et al., 2017). Moreover, in previous work, only one of the mentioned normalization methods is selected and it is used in all normalization layers of a neural network. This will limit the performance of a neural network and it is hard to decide which normalization method is suitable for a specific task.

In this paper, we investigate four types of graph normalization methods, including node-wise normalization, adjacency-wise normalization, graph-wise normalization and batch-wise normalization, as illustrated in Figure 1 (a)-(d). The node-wise normalization only considers its own features and compute the statistics over the elements of its feature vector. Node-wise normalization is equivalent to layer normalization in nature. According to the adjacent structure, we normalize the node feature only using the mean and variance of its adjacent neighborhoods. We terms this normalization method as adjacency-wise normalization, as shown in Figure 1 (b). We also introduce a graph-wise normalization which normalizes the nodes using the mean and variance computed from each graph alone as shown in Figure 1 (c). BN computes the statistics (mean and variance) over a mini-batch and is named as batch-wise normalization (Figure 1 (d)) in this paper. Moreover, we also extend these normalization methods to handle the edge features as shown in Figure 1 (e)-(h). Then, we propose to learn graph normalization by optimizing a weighted combination of normalization techniques at four different levels, including node-wise normalization, adjacency-wise normalization, graph-wise normalization, and batch-wise normalization. By learning the optimal weights, we are able to automatically select the most suitable normalization or a mixture of multiple normalization methods for a specific task.

The contributions of the paper can be highlighted as follows.

- According to different statistical levels of mean and variance of graph, we formulate the graph normalization methods into four levels: node-wise, adjacency-wise, graph-wise, and batch-wise. To the best of our knowledge, the adjacency-wise normalization and graph-wise normalization are proposed for training GNNs for the first time.

- We propose a novel framework to learn graph normalization by optimizing a weighted combination of normalization methods at different levels. By learning the optimal weights, we are able to automatically select the best normalization method or the best combination of multiple normalization methods for a specific task. We observe that graph-wise normalization performs well on some node classification tasks and batch-wise normalization yields better performance on graph classification and regression tasks, which is consistent with the distribution of the learned dominant weights in our proposal.

- We conduct extensive experiments on several benchmark datasets with different tasks to quantitatively evaluate the listed four normalization methods and the learned normalization methods in various scenarios.

2 GRAPH NORMALIZATION

Suppose that we have N graphs $\mathcal{G}_1, \mathcal{G}_2, ..., \mathcal{G}_N$ in a mini-batch. Let $\mathcal{G}_k = (\mathbb{V}_k, \mathbb{E}_k)$ be the k-th graph, where $\mathbb{V}_k$ is the set of nodes and $\mathbb{E}_k$ is the set of edges. GNNs propagate information along the edges $\mathbb{E}$ of a graph to update the representation of each node for downstream tasks, such as node classification, link prediction and graph classification. We denote $v_{k,i}$ as the i-th node of the $\mathcal{G}_k$ and $h_{v_{k,i}} \in \mathbb{R}^d$ as the corresponding node feature. $h_{v_{k,i}}^j$ represents the jth element of node feature $h_{v_{k,i}}$. $\mathbb{N}(v_{k,i})$ represents the neighbors of node $v_{k,i}$ and $v_{k,i}$.

2.1 NORMALIZATION METHODS FROM DIFFERENT LEVELS

Currently, most GNNs use BN, which computes statistics over a mini-batch, after the message-passing layer to normalize feature representation. Note that graph data contains rich structural information. In this paper, by considering the structure information at different scales, we present and rewrite several normalization methods for graph-structure data.

Node-wise Normalization. For the node $v_{k,i}$, its feature $h_{v_{k,i}}$ has d elements, as Figure 1 (a). Each of them may have different distribution and we can normalize them to reduce the "covariate shift" problem. Then, according to layer normalization, $h_{v_{k,i}}$ can be normalized by itself:

$$\hat{h}_{v_{k,i}}^{(n)} = \frac{h_{v_{k,i}} - \mu_{k,i}^{(n)}\mathbf{1}}{\sigma_{k,i}^{(n)}}$$

$$\mu_{k,i}^{(n)} = \frac{1}{d}\sum_{j=1}^{d} h_{v_{k,i}}^{j}, \quad \sigma_{k,i}^{(n)} = \sqrt{\frac{1}{d}\sum_{j=1}^{d}(h_{v_{k,i}}^{j} - \mu_{k,i}^{(n)})^2}, \tag{1}$$

where $\mu_{k,i}^{(n)}$ and $\sigma_{k,i}^{(n)}$ are the mean and variance along the feature dimension for node $v_{k,i}$, and $\mathbf{1} \in \mathcal{R}^d$ represents a d-dimension vector filled with the scale value 1. If we normalize the edge features, each edge feature (Figure 1 (e)) is processed individually as Equation (1). Essentially, node-wise normalization is equivalent to applying LN to each node of a graph.

Adjacent-wise Normalization. Each node in a graph has its own neighbors. However, node-wise normalization performs normalization on each node alone and ignores the local structure of each node. For a node $v_{k,i}$ in a graph $\mathcal{G}_k$, we consider its topological structure of its adjacent nodes $\mathbb{N}(v_{k,i})$, as Figure 1 (b). The number of adjacent nodes $|\mathbb{N}(v_{k,i})|$ varies with the node $v_{k,i}$. $\mathbb{N}(v_{k,i})$ may only contain itself. Thus, the statistics (mean and variance) computed over $\mathbb{N}(v_{k,i})$ are unstable. Like LN, we can compute mean and variance across over all elements of feature vectors corresponding adjacent nodes. So, the adjacency-wise normalization for node $v_{k,i}$ is written as:

$$\hat{h}_{v_{k,i}}^{(a)} = \frac{h_{v_{k,i}} - \mu_{k,i}^{(a)}\mathbf{1}}{\sigma_{k,i}^{(a)}}$$

$$\mu_{k,i}^{(a)} = \frac{1}{|\mathbb{N}(v_{k,i})| \times d} \sum_{j' \in \mathbb{N}(v_{k,i})} \sum_{j=1}^{d} h_{v_{k,j'}}^{j},$$

$$\sigma_{k,i}^{(a)} = \sqrt{\frac{1}{|\mathbb{N}(v_{k,i})| \times d} \sum_{j' \in \mathbb{N}(v_{k,i})} \sum_{j=1}^{d}(h_{v_{k,j'}}^{j} - \mu_{k,i}^{(a)})^2}, \tag{2}$$

where $\mu_{k,i}^{(a)}$ and $\sigma_{k,i}^{(a)}$ are two scalars. For the edge $e_{k,i}$, as Figure 1 (f), the adjacent edges $\mathbb{N}(e_{k,i})$ are considered.

Graph-wise Normalization. Thirdly, nodes belonging to $\mathcal{G}_k$ naturally are composed of a group. In order to preserve the global structure of a graph, the feature representation of each node is normalized based on the statistics computed over $\mathcal{G}_k$. So, we define graph-wise normalization for node $v_{k,i}$ as the following Equation (3):

$$\hat{h}_{v_{k,i}}^{(g)} = \frac{h_{v_{k,i}} - \mu_k^{(g)}}{\sigma_k^{(g)}}$$

$$\mu_k^{(g)} = \mathop{\mathbb{E}}_{v \sim \mathcal{G}_k} h_v = \frac{1}{|\mathcal{G}_k|} \sum_{v_{k,i} \in \mathcal{G}_k} h_{v_{k,i}},$$

$$\sigma_k^{(g)} = \sqrt{\mathop{\mathbb{E}}_{v \sim \mathcal{G}_k}(h_v - \mu_k^{(g)})^2} = \sqrt{\frac{1}{|\mathcal{G}_k|} \sum_{v_{k,i} \in \mathcal{G}_k}(h_{v_{k,i}} - \mu_k^{(g)})^2}, \tag{3}$$

$\mu_k^{(g)}$ and $\sigma_k^{(g)}$ in Equation (3) are the mean and standard deviation vectors of $\mathcal{G}_k$. Similarly, for the edges $\mathbb{E}_k$ of Graph $\mathcal{G}_k$ (Figure 1 (g)), we compute the mean and variance over all edges of $\mathcal{G}_k$. When the task has only one graph, graph-wise normalization is similar to BN while the later uses a smoothing average updater but graph-wise normalization does not.

Batch-wise Normlaization. To keep training stable, BN is one of the most critical components. For a mini-batch, there are N graphs. We can compute the mean and standard deviation across over the graphs of a mini-batch, then a node feature $h_{v_{k,i}}$ is normalized by the following equation, which is

batch-wise normalization (GN$_b$):

$$\hat{h}_{v_{k,i}}^{(b)} = \frac{h_{v_{k,i}} - \mu^{(b)}}{\sigma^{(b)}}$$

$$\mu^{(b)} = \text{mean}\{h_{v_{k,i}}|\{\{v_{k,i}\}_{i=1}^{\mathcal{G}_k}\}_{k=1}^{N}\} = \frac{1}{\text{n}}\sum_{k=1}^{N}\sum_{i=1}^{|\mathcal{G}_k|} h_{v_{k,i}}, \tag{4}$$

$$\sigma^{(b)} = \text{std}\{h_{v_{k,i}}|\{\{v_{k,i}\}_{i=1}^{\mathcal{G}_k}\}_{k=1}^{N}\} = \sqrt{\frac{1}{\text{n}}\sum_{k=1}^{N}\sum_{i=1}^{|\mathcal{G}_k|}(h_{v_{k,i}} - \mu^{(b)})^2},$$

where n means the total nodes of all N graphs. GN$_b$ performs normalization over nodes of all N graphs in a mini-batch and is the same as BN (Ioffe & Szegedy, 2015) in nature.

The properties of four normalization methods are summarized as follows.

- Node-wise normalization is equivalent as LN in operation. Node-wise normalization only considers each node's feature alone, but ignores the adjacent and whole graph structures.

- Adjacent-wise normalization takes the adjacent nodes into account. It reflects the difference between the node and its neighbors.

- Graph-wise normalization takes the features of all nodes in a graph into account. It embodies the difference of all the nodes in the whole graph. In ideal situation, this method can stand out the target node from the graph.

- Batch-wise normalization is the same as the standard batch normalization in nature. It expresses differences among the graphs. When the task only has one graph, then the batch-wise normalization is similar with the graph normalization except that momentum average is used in batch-wise but not in the graph-wise.

2.2 LEARNING AN UNITED GRAPH NORMALIZATION

Although we have defined several normalization methods for the graph-structured data, different tasks prefer to different normalization methods and for a specific task, it is hard to decide which normalization method should be used. Moreover, one normalization approach is utilized in all normalization layers of a GNN. This will sacrify

$$\hat{h}_{v_{k,i}} = \gamma(\lambda_n \odot \hat{h}_{v_{k,i}}^{(n)} + \lambda_a \odot \hat{h}_{v_{k,i}}^{(a)} + \lambda_g \odot \hat{h}_{v_{k,i}}^{(g)} + \lambda_b \odot \hat{h}_{v_{k,i}}^{(b)}) + \beta, \tag{5}$$

where $\lambda_n, \lambda_a, \lambda_g, \lambda_b \in \mathbb{R}^d$ are defined as the trainable gate parameters with the same dimension as $h_{v_{k,i}}$, γ and β denote the trainable scale and shift, respectively. $\lambda_n, \lambda_a, \lambda_g, \lambda_b$ indicate the contribution of the corresponding normalized feature to $\hat{h}_{v_{k,i}}$. Thus, we constrain the elements of $\lambda_n, \lambda_a, \lambda_g, \lambda_b$ to be in the range $[0,1]$ by the following scheme:

$$\begin{cases} \lambda_u^j = \text{clip}_{[0,+\infty)}(\lambda_u^j) & j \in \{1,2,...,d\} \quad and \quad u \in \{n,a,g,b\} \\ \lambda_u^j = \dfrac{\lambda_u^j}{\sum_{k\in\{n,a,g,b\}}\lambda_k^j}, & j \in \{1,2,...,d\} \quad and \quad u \in \{n,a,g,b\} \end{cases} \tag{6}$$

$\{\lambda_u\}_{u\in\{n,a,g,b\}}$ in Equation (5) are important weights used to combine the normalized feature of each normalizer and they satisfy the summation constraint, $\sum_{u\in\{n,a,g,b\}}\lambda_u^j = 1, j \in \{1,2,...,d\}$. Thus, in this mechanism, if a normalizer is better for a specific task, its corresponding weights may be higher than others. In GN, several normalization methods cooperate and compete with each other to improve the performance of GNNs.

Different normalization methods are suitable for different tasks. In GN, the weights $\lambda_n, \lambda_a, \lambda_g, \lambda_b$ are updated adaptively for a specific task. Usually, the best-performing normalization will have largest weight in the learnt weight distribution. Thus, GN can be an effective tactic to select one type of normalization method or a best combination of several normalization approaches for one specific task.

3 EXPERIMENTS

As Dwivedi et al. (2020), we evaluate GN_n, GN_a, GN_g, GN_b, and the united Graph Normliza-tion (GN) under three frameworks including Graph Convolution Network (GCN) (Kipf & Welling, 2016), Graph Attention Network (GAT) (Velickovic et al., 2018) and GatedGCN (Bresson & Laurent, 2017). We also assess the performance of GNNs without normalization layer named as "No Norm". We use the same experimental setup as Dwivedi et al. (2020). The implementation of GCN, GAT and GatedGCN are found in GNN benchmarking framwork [1], which uses Deep Graph Library (Wang et al., 2019) and PyTorch (Paszke et al., 2019).

Experimental datasets consist of three types of tasks including node classification, link prediction, and graph classification/regression. We use all seven datasets from (Dwivedi et al., 2020), which are PATTERN, CLUSTER, SROIE, TSP, COLLAB, MNIST, CIFAR10, and ZINC. In addition, we apply GatedGCN for key information extraction problem and evaluate the affect of different normalization methods on SROIE (Huang et al., 2019), which is used for extracting key information from receipt in ICDAR 2019 Challenge (task 3). For more information about the detailed statistics of the datasets refer to Appendix C.1.

The hyper-parameters and optimizers of the models and the details of the experimental settings are the same as GNN bechmarking framework Dwivedi et al. (2020). To evaluate the performance of GNNs with different depths, we run experiments on CLUSTER and PATTERN datasets with depth {4, 16} respectively. For the other datasets, we fix the number of GCN layer to 4.

3.1 NODE CLASSIFICATION

For node classification, its goal is to assign each node $v \in V$ to a number of classes. Hence, GNNs predict the label of each node by passing the node representation vector to a MLP. We evaluate the performance of different normalization approaches on CLUSTER and PATTERN datasets. Moreover, we apply node classification to key information extraction. SROIE consists of 626 receipts for training and 347 receipts for testing. Each image in the SROIE dataset is annotated with text bounding boxes (bbox) and the transcript of each text bbox. There are four entities to extract (Company, Date, Address and Total) from a receipt, as shown in Appendix C.2.

3.1.1 CLUSTER AND PATTERN DATASETS

For CLUSTER and PATTREN datasets, the average node-level accuracy weighted with respect to the class sizes is used to evaluate the performance of all models. For each model, we conduct 4 runs with different seeds {41, 95, 35, 12} and compute the average test accuracy as shown in Table 1.

Graph-wise normalization (GN_g) outperforms batch-wise normalization (GN_b) obviousluy in most situations. For instance, when the depth of GNNs is 4, GatedGCN with GN_g achieves 9% improve-ment over GN_b on CLUSTER. Batch-wise normalization computes the statistics over a batch data and ignores the differences between different graphs. Different from GN_b, GN_g performs normaliza-tion only for each graph. Thus, GN_g can learn the dedicated information of each graph and normalize the feature of each graph to a reasonable range. As we known, the performance of the adjacency-wise normalization (GN_a) is similar with that of the node-wise normalization (GN_n). Compared with GN_n, GN_a consider the neighbors of each node and gets higher accuracies. We can see that GN gets comparable results for different GNNs. The results of GN are close to the best results in most scenarios due to its flexibility and adaptability. GN adaptively learns weight combination of those normalization approaches which better adapt itself to the node classification task.

3.1.2 SROIE DATASET

In this experiment, we test the performance of GatedGCN with several different normalization ap-proaches for extracting key information from a receipt. For a receipt image, each text bounding box is label with five classes, which are Total, Date, Address, Company and Other. Thus, for a receipt, we treat each text bounding box as a node. Feature representation for each node will be supplied by Appendix C.2. "Company" and "Address" usually consist of multiple text bounding boxes (nodes).

---

[1] https://github.com/graphdeeplearning/benchmarking-gnns

Dataset	CLUSTER						PATTERN					
Network	GCN		GAT		GatedGCN		GCN		GAT		GatedGCN	
	Train (Acc)	Test (Acc)	Train (Acc)	Test (Acc)	Train (Acc)	Test (Acc)	Train (Acc)	Test (Acc)	Train (Acc)	Test (Acc)	Train (Acc)	Test (Acc)
Layer=4 No Norm	54.3±1.9	54.2±1.9	59.7±0.4	59.0±0.3	58.0±2.8	57.4±2.6	61.9±0.2	61.4±0.1	81.8±0.7	81.0±0.8	82.5±3.2	82.5±3.4
GN_n	57.2±0.1	57.0±0.1	59.6±0.2	59.0±0.2	61.1±0.9	60.5±0.7	64.9±0.1	64.0±0.1	79.5±0.3	78.7±0.4	82.7±2.7	82.6±2.8
GN_a	58.9±0.6	58.7±0.6	68.5±0.5	67.9±0.5	63.5±0.9	63.0±0.9	66.5±1.4	65.3±1.2	82.0±0.4	81.1±0.4	84.3±0.0	84.5±0.0
GN_g	68.7±0.3	67.0±0.1	69.5±0.1	68.1±0.1	70.6±0.1	69.3±0.0	80.2±0.1	77.3±0.1	83.6±0.1	79.2±0.2	85.1±0.0	85.1±0.0
GN_b	55.1±1.8	54.3±1.5	59.3±0.4	58.6±0.3	61.4±0.2	60.3±0.1	64.8±0.2	63.8±0.1	78.3±1.2	76.3±1.0	84.5±0.1	84.5±0.1
GN	68.3±0.3	67.4±0.2	68.9±0.2	68.2±0.2	69.8±0.3	69.3±0.1	79.0±0.4	76.5±0.4	81.7±0.7	79.2±0.7	85.3±0.3	85.2±0.2
Layer=16 No Norm	85.3±0.5	72.4±0.1	63.6±2.5	63.4±2.3	80.6±1.3	71.2±0.4	82.8±0.4	82.9±0.4	73.4±0.4	69.7±0.2	85.6±0.0	85.7±0.0
GN_n	63.6±7.0	63.2±6.9	83.8±0.5	72.2±0.4	84.6±0.8	73.8±0.2	76.7±0.8	71.4±0.3	87.7±0.7	81.8±0.5	85.6±0.0	85.8±0.0
GN_a	66.7±1.5	66.0±1.5	85.6±0.6	73.2±0.2	84.7±0.6	74.1±0.3	74.9±1.7	70.7±0.7	84.8±0.3	82.8±0.5	85.6±0.0	85.8±0.0
GN_g	87.2±0.4	72.5±0.2	91.9±0.3	73.4±0.1	90.9±0.5	74.5±0.1	98.9±0.1	76.3±0.2	92.8±0.1	81.2±0.2	86.7±0.2	85.3±0.1
GN_b	67.6±3.7	65.1±2.6	83.9±0.6	72.2±0.3	88.2±1.0	73.7±0.3	79.0±1.6	72.0±0.3	91.9±0.6	80.2±0.2	86.1±0.2	85.7±0.1
GN	85.8±0.4	73.8±0.2	87.3±2.1	72.6±0.6	88.6±0.4	75.8±0.2	93.6±1.9	77.8±0.3	95.6±0.5	79.2±0.4	87.3±0.3	85.7±0.1

Table 1: Node classification results on the CLUSTER and PATTERN. Results are averaged over four different random seeds. Red: the best model, Violet: good models.

If and only if all nodes of each entity are classified correctly, this entity is extracted successfully. We compute the mean accuracies for each text field and the average accuracies for each receipt as shown in Table 2.

Text Field	Normalization Method					
	No Norm	GN_n	GN_a	GN_g	GN_b	GN
Total	87.5	91.9	74.5	**96.8**	94.8	94.5
Date	96.5	98.0	95.9	**98.8**	97.4	97.4
Address	91.6	92.0	80.0	**94.5**	93.9	93.6
Company	92.2	93.3	87.8	94.5	93.0	**94.8**
Average	92.0	94.0	84.6	**96.2**	94.8	95.1

Table 2: Performance (accuracy) comparison of different normalization approaches.

We can observe that GN_g achieves the best performance among all compared normalization methods. In the receipt, there are many nodes with only numeric texts. It is hard to differentiate the "Total" field from other nodes with numeric text. GN_g performs well in this field and outperforms the second best by 2.0%. We believe that the graph-wise normalization can make the 'Total' field stand out from the other bounding boxes with numeric text by aggregating the relevant anchor point information from its neighbors and removing the mean number information. Similarly, graph-wise normalization can promote extracting information for the other three key fields. It's interesting that the graph of each receipt is special with Neighboring nodes usually belonging to different classes. Thus, the performance of adjacency-wise normalization is worse than node-wise normalization.

3.2 LINK PREDICTION

Link prediction is to predict whether there is a link between two nodes v_i and v_j in a graph. Two node features of v_i and v_j, at both ends of edge $e_{i,j}$, are concatenated to make a prediction. Then the concatenated feature is fed into a MLP for prediction. Experimental results are shown in Table 3.

All the five normalization methods achieve similar performance on TSP dataset. Compared with other normalization methods, the results of GN are very stable. For each GNN, the result of GN is comparable with the best result. In addition, GNNs without normalization layer obtains good performance.

COLAB dataset contains only a graph with 235,868 nodes. When we use adjacency-wise normalization, we encounter out-of-memory problem. Thus, we don't report the results of GN_a and GN. Compared with GNNs with normalization layer, the results of GNNs without normalization layer (No Norm) seriously decrease. GN_g performs better than GN_b. GatedGCN with GN_g achieves the best result.

3.3 GRAPH CLASSIFICATION AND GRAPH REGRESSION

Graph classification is to assign one label to each graph. We conduce experiments on CIFAR10 and MNIST. Average class accuracies are reported in Table 4. ZINC is a dataset for graph regression.

Network	GCN		GAT		GatedGCN	
Dataset	TSP					
	Train (F1)	Test (F1)	Train (F1)	Test (F1)	Train (F1)	Test (F1)
No Norm	0.628±0.001	0.627±0.001	0.677±0.002	0.675±0.002	0.805±0.005	0.804±0.005
GN_n	0.635±0.001	0.634±0.001	0.663±0.008	0.662±0.008	0.810±0.003	0.808±0.003
GN_a	0.633±0.004	0.631±0.004	0.678±0.003	0.676±0.003	0.805±0.005	0.803±0.004
GN_g	0.630±0.001	0.629±0.001	0.669±0.003	0.668±0.001	0.890±0.001	0.806±0.001
GN_b	0.633±0.001	0.632±0.001	0.673±0.004	0.671±0.004	0.791±0.002	0.789±0.002
GN	0.635±0.001	0.633±0.001	0.673±0.001	0.671±0.001	0.804±0.001	0.802±0.001
Dataset	COLAB					
	Train (Hits)	Test (Hits)	Train (Hits)	Test (Hits)	Train (Hits)	Test (Hits)
No Norm	73.02±7.03	38.32±4.13	64.19±4.02	32.69±4.48	38.55±8.13	22.60±3.40
GN_n	81.81±7.40	45.75±4.14	95.93±0.54	51.76±0.68	91.72±3.40	51.55±1.44
GN_g	93.67±0.71	52.27±1.28	97.11±0.65	51.36±1.15	97.50±2.52	52.71±0.36
GN_b	91.88±0.04	51.16±0.10	97.11±0.43	51.54±0.90	95.31±3.56	51.87±0.41

Table 3: Link prediction results on the TSP and COLAB. Red: the best model,Violet: good models.

the mean absolute error (MAE) between the predicted value and the ground truth is calculated for each group. Average MAEs also are reported in Table 4. We can see that GN_b outperforms others in most cases. GN_g doesn't work well on graph classification and regression. Furthermore, GN_g affects the performance of GN. GN integrates the normalized features of GN_n, GN_a, GN_g, and GN_b and adptively pays more attention to GN_b due to its outstanding performance. Therefore, its performance is comparable with GN_b.

Network	GCN		GAT		GatedGCN	
	Train (Acc)	Test (Acc)	Train (Acc)	Test (Acc)	Train (Acc)	Test (Acc)
Dataset	MNIST					
No Norm	93.62±0.72	90.10±0.25	100.00±0.00	95.39±0.16	100.00±0.00	96.61±0.09
GN_n	96.63±0.91	90.53±0.22	100.00±0.00	95.66±0.14	100.00±0.00	97.23±0.12
GN_a	95.65±0.75	89.68±0.20	100.00±0.00	95.41±0.22	100.00±0.00	96.87±0.21
GN_g	96.90±0.52	86.18±0.30	100.00±0.00	94.74±0.13	100.00±0.00	96.17±0.16
GN_b	97.16±1.06	90.51±0.22	99.99±0.00	95.77±0.19	100.00±0.00	97.47±0.11
GN	97.34±0.66	90.46±0.17	100.00±0.00	95.75±0.22	100.00±0.00	97.41±0.17
Dataset	CIFAR10					
No Norm	65.87±1.68	54.56±0.53	88.80±1.31	62.13±0.31	82.81±1.15	63.44±0.22
GN_n	73.64±1.42	55.77±0.31	87.67±0.81	63.04±0.60	90.14±2.05	67.86±0.65
GN_a	71.48±1.27	54.83±0.32	86.80±0.70	62.72±0.26	90.85±0.32	67.21±0.44
GN_g	71.75±2.48	46.41±0.29	89.20±0.41	54.44±0.28	81.29±6.37	52.69±3.28
GN_b	69.34±2.47	55.14±0.26	89.56±1.41	64.54±0.24	95.75±0.12	67.83±0.68
GN	80.33±3.10	54.73±0.68	94.48±1.58	62.98±0.47	98.80±0.28	66.84±0.16
	Train (MAE)	Test (MAE)	Train (MAE)	Test (MAE)	Train (MAE)	Test (MAE)
Dataset	ZINC					
No Norm	0.368±0.022	0.472±0.005	0.270±0.029	0.490±0.001	0.292±0.003	0.456±0.004
GN_n	0.349±0.019	0.455±0.007	0.295±0.014	0.456±0.001	0.260±0.021	0.428±0.005
GN_a	0.351±0.013	0.458±0.003	0.291±0.013	0.458±0.001	0.274±0.023	0.437±0.001
GN_g	0.263±0.033	0.547±0.029	0.228±0.010	0.519±0.001	0.216±0.019	0.507±0.003
GN_b	0.346±0.019	0.465±0.009	0.308±0.028	0.480±0.003	0.280±0.013	0.431±0.007
GN	0.357±0.017	0.486±0.007	0.298±0.018	0.483±0.005	0.275±0.011	0.458±0.003

Table 4: Experimental results on MNIST, CIFAR10 and ZINC. Red: the best model,Violet: good models.

3.4 ANALYSIS

The above experimental results indicates that GN_g outperforms batch normalization on most node classification tasks. For each single normalization method, it performs very well on some datasets, while its performance may decrease sharply on other datasets. Meanwhile, our proposed GN, which

	CLUSTER (Acc)	PATTERN (Acc)	SROIE (Acc)	TSP (F1)	MNIST (Acc)	CIFAR10 (Acc)	ZINC (MAE)
GN_n	–	–	–	80.83	97.23	67.86	0.4283
GN_a	63.02	84.53	–	–	–	–	–
GN_g	69.31	85.07	96.2	80.61	–	–	–
GN_b	–	–	94.8	–	97.47	67.83	0.4311
Combine	69.16	84.64	95.4	81.11	97.52	67.88	0.4371

Table 5: Performance of different normalization methods on several datasets. For each dataset, we give the performance of two best normalization methods and a new normalization method combined these two best normalizer like Equation (5).

integrates several normalization methods into a framework, achieves competitive results compared with the best single normalization method on various datasets.

In this part, we analyze the effect of each normalization method on different datasets. GN adaptively combines the results of several normalization methods and $\{\lambda_u\}_{u \in n,a,g,b}$ in Equation (5) indicate the importance weight vector of the corresponding normalizer, respectively. We initialize the important weights $\{\lambda_u\}_{u \in n,a,g,b}$ in each layer to the equal values (*i.e.* 1/4). In the training phase, the values of $\{\lambda_u\}_{u \in n,a,g,b}$ changes between 0 and 1. We investigate the averaged weights in different layers of GatedGCN on several datasets. We get the important weights of each normalizer in each layer from the optimized model. Since $\lambda_u \in \mathbb{R}^d$, then the averaged weights of each normalizer is calculated over all of the d elements of λ_u. According to Figure 2, the weights of each normalizer not only are distinct for different datasets, but also vary for different layers. It implies that distinct layers may have their own preference of normalization methods to achieve good performance. It's interesting that the weights of GN_g are larger than others on node classification tasks while GN_b is more importance on other tasks. GN has the ability to automatically choose the suitable normalizers for a specific task.

Furthermore, for each dataset, we select the two best normalizers and integrate them into a new normalization method like Equation (5). From Table 5, the combined normalizer achieves the comparable results with the best normalization method for each dataset. Therefore, these results show that the important weights indicate whether the corresponding normalization method is suitable for the current task.

Figure 2: Learnt weight distributions of four normalization methods along with layers on different tasks.

4 CONCLUSION

In this paper we formulate four graph normalization methods, node-wise, adjacent-wise, graph-wise, and graph-wise according to different statistical levels of mean and variance. Node-wise normalization only considers its own statistical information. Adjacent-wise and graph-wise takes local and global graph structures into account. BN computes the statistics in a mini-batch level.

Different normalization methods perform variously in different tasks. We observe graph-wise and adjacent-wise normalizations perform well on some node classification tasks, batch-wise normalization shows better performance on graph classification and regression tasks. Therefore, we propose to learn an effective Graph Normalization (GN) by optimizing a weighted combination of node-wise normalization, adjacency-wise normalization, graph-wise normalization, and batch-wise normalization. Through analyzing the distributions of weights, we can select one or a combinations of several normalization for one specific task.

REFERENCES

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. 2016.

Xavier Bresson and Thomas Laurent. Residual gated graph convnets. *CoRR*, abs/1711.07553, 2017. URL http://arxiv.org/abs/1711.07553.

Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. *CoRR*, abs/1312.6203, 2014.

Jie Chen, Tengfei Ma, and Cao Xiao. Fastgcn: fast learning with graph convolutional networks via importance sampling. *arXiv preprint arXiv:1801.10247*, 2018.

Zhengdao Chen, Soledad Villar, Lei Chen, and Joan Bruna. On the equivalence between graph isomorphism testing and function approximation with gnns. *ArXiv*, abs/1905.12560, 2019.

Ke Cheng, Yifan Zhang, Xiangyu He, Weihan Chen, Jian Cheng, and Hanqing Lu. Skeleton-based action recognition with shift graph convolutional network. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 183–192, 2020.

Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *NIPS*, 2016.

Neofytos Dimitriou and Ognjen Arandjelovic. A new look at ghost normalization. *arXiv preprint arXiv:2007.08554*, 2020.

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

Wenqi Fan, Yao Ma, Qing Li, Yuan He, Eric Zhao, Jiliang Tang, and Dawei Yin. Graph neural networks for social recommendation. In *The World Wide Web Conference*, pp. 417–426, 2019.

William L. Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NIPS*, 2017.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. Densely connected convolutional networks. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2261–2269, 2017.

Zheng Huang, Kai Chen, Jianhua He, Xiang Bai, Dimosthenis Karatzas, Shijian Lu, and CV Jawahar. Icdar2019 competition on scanned receipt ocr and information extraction. In *2019 International Conference on Document Analysis and Recognition (ICDAR)*, pp. 1516–1520. IEEE, 2019.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Francis Bach and David Blei (eds.), *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pp. 448–456, Lille, France, 07–09 Jul 2015. PMLR. URL http://proceedings.mlr.press/v37/ioffe15.html.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *CoRR*, abs/1609.02907, 2016. URL `http://arxiv.org/abs/1609.02907`.

Xia Li, Yibo Yang, Qijie Zhao, Tiancheng Shen, Zhouchen Lin, and Hong Liu. Spatial pyramid based graph reasoning for semantic segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8950–8959, 2020.

Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard S. Zemel. Gated graph sequence neural networks. *CoRR*, abs/1511.05493, 2016.

Jaechang Lim, Seongok Ryu, Kyubyong Park, Yo Joong Choe, Jiyeon Ham, and Woo Youn Kim. Predicting drug–target interaction using a novel graph neural network with 3d structure-embedded graph representation. *Journal of chemical information and modeling*, 59(9):3981–3988, 2019.

Xiaojing Liu, Feiyu Gao, Qiong Zhang, and Huasha Zhao. Graph convolution for multimodal information extraction from visually rich documents. *arXiv preprint arXiv:1903.11279*, 2019a.

Yongcheng Liu, Bin Fan, Shiming Xiang, and Chunhong Pan. Relation-shape convolutional neural network for point cloud analysis. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 8895–8904, 2019b.

Ping Luo, Ruimao Zhang, Jiamin Ren, Zhanglin Peng, and Jingyu Li. Switchable normalization for learning-to-normalize deep representation. *IEEE transactions on pattern analysis and machine intelligence*, 2019.

Haggai Maron, Heli Ben-Hamu, Hadar Serviansky, and Yaron Lipman. Provably powerful graph networks. In *NeurIPS*, 2019.

Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodolà, Jan Svoboda, and Michael M. Bronstein. Geometric deep learning on graphs and manifolds using mixture model cnns. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5425–5434, 2017.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems*, pp. 8026–8037, 2019.

F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

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

Weijing Shi and Raj Rajkumar. Point-gnn: Graph neural network for 3d object detection in a point cloud. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1711–1719, 2020.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

Dmitry Ulyanov, Andrea Vedaldi, and Victor S. Lempitsky. Instance normalization: The missing ingredient for fast stylization. *ArXiv*, abs/1607.08022, 2016.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017. URL `http://arxiv.org/abs/1706.03762`.

Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. *ArXiv*, abs/1710.10903, 2018.

Minjie Wang, Lingfan Yu, Da Zheng, Quan Gan, Yu Gai, Zihao Ye, Mufei Li, Jinjing Zhou, Qi Huang, Chao Ma, Ziyue Huang, Qipeng Guo, Hao Zhang, Haibin Lin, Junbo Zhao, Jinyang Li, Alexander J Smola, and Zheng Zhang. Deep graph library: Towards efficient and scalable deep learning on graphs. *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

Xiaolong Wang, Yufei Ye, and Abhinav Gupta. Zero-shot recognition via semantic embeddings and knowledge graphs. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 6857–6866, 2018.

Shu Wu, Yuyuan Tang, Yanqiao Zhu, Liang Wang, Xing Xie, and Tieniu Tan. Session-based recommendation with graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 346–353, 2019.

Yuxin Wu and Kaiming He. Group normalization. In *ECCV*, 2018.

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

Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. *arXiv preprint arXiv:1806.03536*, 2018.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *ArXiv*, abs/1810.00826, 2019.

Liang Yao, Chengsheng Mao, and Yuan Luo. Graph convolutional networks for text classification. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 7370–7377, 2019.

Si Zhang, Hanghang Tong, Jiejun Xu, and Ross Maciejewski. Graph convolutional networks: Algorithms, applications and open challenges. In *International Conference on Computational Social Networks*, pp. 79–91. Springer, 2018.

Lingxiao Zhao and Leman Akoglu. Pairnorm: Tackling oversmoothing in gnns. *ArXiv*, abs/1909.12223, 2020.

A   GRAPH NEURAL NETWORKS

Graph neural networks (Kipf & Welling, 2016; Velickovic et al., 2018) are effective in learning graph representations. For node v, GNNs update its representation by utilizing itself and its adjacent neighbors. To capture high-order structure information of the graph, GNNs learn a new feature representation of each node over multiple layers. In a layer of GNNs, each node v sends a "message"- its feature representation, to the nodes in $\mathcal{N}(v)$; and then the feature representation of v is updated according to all collected information from the neighborhood $\mathcal{N}(v)$. Mathematically, at the ℓ-th layer, we have,

$$h_v^{\ell+1} = \psi^{\ell+1}(\mathcal{C}\{h_v^\ell, \mathcal{M}\{\phi^{\ell+1}(h_u^\ell)|u \in \mathcal{N}(v)\}\}) \tag{7}$$

where h_u^ℓ denote the feature vector at layer ℓ of node $u \in \mathcal{N}(v)$, ψ and ϕ are learnable functions, $\mathcal{M}$ is the aggregation function for nodes in $\mathcal{N}(v)$, and $\mathcal{C}$ is utilized to combine the feature of node v and its neighbors. Especially, the initial node representation $h_v^0 = x_v$ represents the original input feature vector of node v.

Graph ConvNets(Kipf & Welling, 2016) treats each neighbor node u equally to update the representation of a node v as:

$$h_v^{\ell+1} = \text{ReLU}(\frac{1}{deg_v} \sum_{u \in \mathcal{N}(v)} W^\ell h_u^\ell), \tag{8}$$

where $W \in \mathcal{R}^{d \times d}$, deg_v is the in-degree of node v. One graph convolutional layer only considers immediate neighbors. To use neighbors within K hops, in practice, multiple GCN layers are stacked. All neighbors contribute equally in the information passing of GCN. One of key issue of the GCN

is its over-smoothing problem. This issue can be partially eased by residual shortcut across layers. One effective approach is to use spatial GNNs, such as GAT (Velickovic et al., 2018) and GatedGCN (Bresson & Laurent, 2017).

GAT (Velickovic et al., 2018) learns to assign different weight to adjacent nodes by adopting attention mechanism. In GAT, the feature representation of v can be updated by:

$$h_v^{\ell+1} = \sigma(\sum_{u \in \mathcal{N}(v)} e_{u,v}^{\ell} W^{\ell} h_u^{\ell}), \tag{9}$$

where $e_{u,v}^{\ell}$ measures the contribution of node u's feature to node v:

$$e_{u,v}^{\ell} = \frac{exp(g(\alpha^T [W^{\ell} h_u^{\ell} || W^{\ell} h_v^{\ell}]))}{\sum_{k \in \mathcal{N}(v)} exp(g(\alpha^T [W^{\ell} h_k^{\ell} || W^{\ell} h_v^{\ell}]))}, \tag{10}$$

where $g(\cdot)$ is a LeaklyReLU activation function, α is a weight vector and $||$ is the concatenation operation. Similar to Vaswani et al. (2017), to expand GAT's expressive capability and stabilize the learning process, multi-head attention is employed in GAT. GAT has achieved an impressive improvement over GCN on node classification tasks. However, as the number of graph convolutional layers increases, nodes representations will converge to the same value. The over-smoothing problem still exists.

To mitigate the over-smoothing problem, GatedGCN (Bresson & Laurent, 2017) integrates gated mechanism (Hochreiter & Schmidhuber, 1997), batch normalization (Ioffe & Szegedy, 2015), and residual connections (He et al., 2016) into the network design. Unlike GCNs which treats all edges equally, GatedGCN uses an edge gated mechanism to give different weights to different nodes. So, for node v, the formulation of updating the feature representation is:

$$h_v^{\ell+1} = h_v^{\ell} + \text{ReLU}(\text{BN}(W^{\ell} h_v^{\ell} + \sum_{u \in \mathcal{N}(v)} e_{vu}^{\ell} \odot U^{\ell} h_u^{\ell})), \tag{11}$$

where $W^{\ell}, U^{\ell} \in \mathcal{R}^{d \times d}$, $\odot$ is the Hadamard product, and the edge gates e_{vu}^{ℓ} are defined as:

$$e_{v,u}^{\ell} = \frac{\sigma(\hat{e}_{v,u}^{\ell})}{\sum_{u' \in \mathcal{N}(v)} \sigma(\hat{e}_{v,u'}^{\ell}) + \varepsilon}, \tag{12}$$

$$\hat{e}_{v,u}^{\ell} = \hat{e}_{v,u}^{\ell-1} + \text{ReLU}(\text{BN}(A^{\ell} h_v^{\ell-1} + B^{\ell} h_u^{\ell-1} + C^{\ell} \hat{e}_{v,u}^{\ell-1})),$$

where σ is the sigmoid function, ε is a small fixed constant, $A^{\ell}, B^{\ell}, C^{\ell} \in R^{d \times d}$. Different from traditional GNNs, GatedGCN explicitly considers edge feature $\hat{e}_{v,u}$ at each layer.

B   Normalization methods

B.1   Batch Normalization

Batch normalization (Ioffe & Szegedy, 2015) (BN) has become one of the critical components in a deep neural network, which normalizes the features by using the statics computed within a mini-batch. BN can reduce the internal covariate shift problem and accelerate training. We briefly introduce the formulation of BN. Firstly, $H = [h_1, h_2, ..., h_B]^T \in \mathcal{R}^{B \times d}$ is denoted as the input of a normalization layer, where B is the batch size and h_i represents a sample. Then, $\mu^{(B)} \in \mathcal{R}^d$ and $\sigma^{(B)} \in \mathcal{R}^d$ mean the mean and the variance of H along the batch dimension, respectively. BN normalizes each dimension of features using $\mu^{(B)}$ and $\sigma^{(B)}$ as:

$$\hat{H} = \gamma \frac{H - \mu^{(B)}}{\sigma^{(B)}} + \beta,$$

$$\mu^{(B)} = \frac{1}{B} \sum_{i=1}^{B} h_i, \quad \sigma^{(B)} = \sqrt{\frac{1}{B} \sum_{i=1}^{B} (h_i - \mu^{(B)})^2}, \tag{13}$$

$$\mu = \alpha\mu + (1-\alpha) * \mu^{(B)}, \quad \sigma^2 = \alpha\sigma^2 + (1-\alpha)(\sigma_B)^2$$

where γ and β are trainable scale and shift parameters, respectively. In Equation (13), μ and σ denote the running mean and variance to approximate the mean and the variance of the dataset. So, during testing, they are used for normalization.

B.2 Layer Normalization

Layer Normalization (LN) (Ba et al., 2016) is widely adopted in Natural Language Processing, specially Transformer (Vaswani et al., 2017) incorporates LN as a standard normalization scheme. BN computes a mean and a variance over a mini-batch and the stability of training is highly dependent on these two statics. Shen et al. (2020) has showed that transformer with BN leads to poor performance because of the large fluctuations of batch statistics throughout training. Layer normalization computes the mean and variance along the feature dimension for each training case. Different from BN, for each sample h_i, LN computes mean $\mu_i^{(L)}$ and variance $\sigma_i^{(L)}$ across the feature dimension. The normalization equations of LN are as follows:

$$\hat{h}_i = \gamma \odot \frac{h_i - \mu_i^{(L)}}{\sigma_i^{(L)}} + \beta,$$

$$\mu_i^{(L)} = \frac{1}{d} \sum_{j=1}^{d} h_{i_j}, \quad \sigma_i^{(L)} = \sqrt{\frac{1}{d} \sum_{j=1}^{d} (h_{i_j} - \mu_i^{(L)})^2}, \tag{14}$$

where $\hat{h}_i = [\hat{h}_{i_1}, \hat{h}_{i_2}, ..., \hat{h}_{i_d}]$ is the feature-normalized response. γ and β are parameters with dimension d.

Overall, there are many normalization approaches (Ulyanov et al., 2016; Wu & He, 2018; Shen et al., 2020; Dimitriou & Arandjelovic, 2020). Shen et al. (2020) has indicated that BN is suitable for computer vision tasks, while LN achieves better results on NLP. For a normalization approach, its performance may vary a lot in different tasks. So, it is very important to investigate the performance of normalization approaches in GNNs.

C Datasets and experimental details

C.1 Dataset Statistics

Table C.1 summarizes the statistics of the datasets used for our experiments.

Dataset	Graphs	Nodes	Total Nodes	Edges	Total Edges	Avg Edges	Task	Classes
PATTERN	14K	44-188	166,449	752-14,864	85,099,952	51.1	N.C.	2
CLUSTER	12K	41-190	140,643	488-10,820	51,620,680	36.7	N.C.	6
SROIE	971	18-153	52,183	70-2,031	420,903	8.1	N.C.	5
TSP	12K	50-499	3,309,140	1,250-12,475	82,728,500	25	E.C.	2
COLLAB	1	235,868	235,868	2,358,104	2,358,104	10	E.C.	2
MNIST	70K	40-75	4,939,668	320-600	39,517,344	8	G.C.	10
CIFAR10	60K	85-150	7,058,005	680-1,200	56,464,040	8	G.C.	10
ZINC	12K	9-37	277,864	16-84	597,970	2.1	G.R.	-

Table 6: Summary statistics of datasets used in our experiments. The 7th column (AvgEdges) represents the average number of edges per node in a graph. N.C., E.C., G.C., G.R. mean node classification, edge classification, graph classification and graph regression independently.

C.2 SROIE

For a receipt, each text bbox can be viewed as a node of a graph. Its positions, the attributes of bounding box, and the corresponding text are used as the node feature. To describe the relationships among all the text bounding boxes of a receipt, we consider the distance between two nodes v_i and v_j. If the distance between two nodes is less than a threshold θ, v_i and v_j are connected by an edge $e_{i,j}$. The relative positions of two text bounding boxes are important for node classification, hence, we encode the relative coordinates of v_i and v_j to represent the edge e_{ij}. This information

extraction problem can be treated as a node classification problem based on the graph. Our goal is to label each node (text bounding box) with five different classes, including Company, Date, Address, Total and Other. Since GatedGCN explicitly exploit edge features and has achieved state-of-the-art performance on various tasks, for this task, GatedGCN with 8 GCN layers is used.

(a) A receipt image.

Figure 3: Sample images of the SROIE dataset. Four entities are highlighted in different colors. "Company", "Address", "Date", and "Total" are marked with Red, Blue, Yellow, and Purple individually. The "Company" and the "Address" entities usually consist of several text lines.

D ACKNOWLEDGEMENT