LEARNING GRAPH NORMALIZATION FOR GRAPH NEURAL NETWORKS

ABSTRACT

Graph Neural Networks (GNNs) have emerged as a useful paradigm to process graph-structured data. Usually, GNNs are 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. To effectively train a GNN with multiple layers, some normalization techniques are necessary. Though the existing normalization techniques have been shown to accelerate training GNNs, the structure information on the graph is ignored yet. In this paper, we propose two graph-aware normalization methods to effectively train GNNs. Then, by taking into account that normalization methods for GNNs are highly task-relevant and it is hard to know in advance which normalization method is the best, we propose to learn attentive graph normalization by optimizing a weighted combination of multiple graph normalization methods at different scales. By optimizing the combination weights, we can automatically select the best or the best combination of multiple normalization methods for a specific task. We conduct extensive experiments on benchmark datasets for different tasks and confirm that the graph-aware normalization methods lead to promising results and that the learned weights suggest the more appropriate normalization methods for specific task.

1 INTRODUCTION

Graph Neural Networks (GNNs) have shown great popularity due to their efficiency in learning on graphs for various application areas, such as natural language processing (Yao et al., 2019; Zhang et al., 2018), computer vision (Li et al., 2020; Cheng et al., 2020), point cloud (Shi & Rajkumar, 2020), drug discovery (Lim et al., 2019), citation networks (Kipf & Welling, 2016), and social networks (Chen et al., 2018). 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 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 over multiple-hops (Xu et al., 2018).

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; 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, many approaches are designed. For example, Graph ConvNet (GCN) (Kipf & Welling, 2016) employs 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; Graph Attention Network (GAT) (Velickovic et al., 2018) incorporates an attention mechanism into the propagation step, which updates the feature representation of each code via a weighted sum of adjacent node representations; MoNet (Monti et al., 2017) designs a Gaussian kernel with learnable parameters to assign different weights to neighbors; GatedGCN (Bresson & Laurent, 2017) explicitly introduces edge features at each layer and updates edge features by considering the feature representations of these two con-

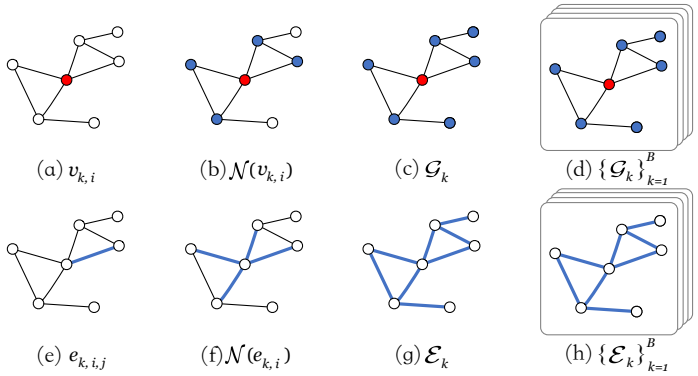

Figure 1: Illustration for normalization methods on graph. Node features are normalized at four levels: (a) Node-wise; (b) Adjacency-wise; (c) Graph-wise; and (d) Batch-wise. Similarly the four normalization methods can be extended to normalize edge features as shown in (e), (f), (g), and (h).

nected nodes of the edge and has achieved state-of-art results on several datasets (Dwivedi et al., 2020). More detailed overview about GNNs are provided in Appendix A.

It is well known that one of the critical ingredients to effectively train deep neural networks is normalization technique, *e.g.*, Batch Normalization (BN) (Ioffe & Szegedy, 2015) is widely used to accelerate the deep neural networks training. Other than BN, several normalization methods have been developed from different perspectives, *e.g.*, Layer Normalization (LN) (Ba et al., 2016) and Group Normalization (Wu & He, 2018) which operate along the channel dimension, Instance Normalization (Ulyanov et al., 2016) which performs a BN-like normalization for each sample, Switchable Normalization (Luo et al., 2019) which utilizes three distinct scopes—including channel, layer, and minibatch—to compute the first order and second order statistics. Each normalization method has its advantages and is suitable for some particular tasks. For instance, BN has achieved perfect performance in computer vision whereas LN outperforms BN in natural language processing (Vaswani et al., 2017).

As an analogue, in Dwivedi et al. (2020), BN is utilized for each graph propagation layer during training GNNs. In Zhao & Akoglu (2020), a novel normalization layer, denoted as PAIRNORM, is introduced to mitigate the over-smoothing problem and prevent all node representations from homogenization by differentiating the distances between different node pairs. Although these methods mentioned above have been demonstrated being useful in training GNNs, *the local structure and global structure of the graph* are ignored in these existing methods. Moreover, in previous work, only one of the mentioned normalization methods is selected and it is used for all normalization layers. This may limit the potential performance improvement of the normalization method and it is also hard to decide which normalization method is suitable to a specific task.

Graph data contains rich structural information. By considering the structure information in the graph, in this paper, we propose two graph-aware normalization methods at different scales: a) adjacency-wise normalization, and b) graph-wise normalization. Unlike BN and LN, the adjacency-wise normalization takes into account the local structure in the graph whereas the graph-wise normalization takes into account the global structure in the graph. On other hand, while multiple normalization methods are available for training GNNs and it is still hard to know in advance which normalization method is the most suitable to a specific task. To tackle with this deficiency, we further propose to learn attentive graph normalization by optimizing a weighted combination of multiple normalization methods. By optimizing the combination weights, we can select the best or the best combination of multiple normalization methods for training GNNs at a specific task automatically.

The contributions of the paper are highlighted as follows.

- We propose two graph-aware normalization methods: adjacency-wise normalization and graph-wise normalization. To the best of our knowledge, it is for the first time that the graph-aware normalization method is proposed for training GNNs.

- We present to learn attentive graph normalization by optimizing a weighted combination of different normalization methods. By learning the combination weights, we can automatically select the

best normalization method or the best combination of multiple normalization methods for training GNNs at a specific task.

- We conduct extensive experiments on benchmark datasets for different tasks and confirm that the graph-aware normalization methods leads to promising results and that the learned weights suggest the more appropriate normalization methods for specific task.

2 GRAPH-AWARE NORMALIZATION AT DIFFERENT SCALES

Suppose that we have N graphs $\mathcal{G}_1, \mathcal{G}_2, ..., \mathcal{G}_N$ in a mini-batch. Let $\mathcal{G}_k = (\mathcal{V}_k, \mathcal{E}_k)$ be the k-th graph, where $\mathcal{V}_k$ is the set of nodes and $\mathcal{E}_k$ is the set of edges. We use $v_{k,i}$ to denote the i-th node of graph $\mathcal{G}_k$ and use $e_{k,i,j}$ to denote the edge between nodes $v_{k,i}$ and $v_{k,j}$ of graph $\mathcal{G}_k$. Moreover, we use $\boldsymbol{h}_{v_{k,i}} \in \mathbb{R}^d$ to represent the feature of node $v_{k,i}$ and $h^j_{v_{k,i}}$ to represent the j-th element of $\boldsymbol{h}_{v_{k,i}}$. We use $\mathcal{N}(v_{k,i})$ to represent the neighbors of node $v_{k,i}$ (including node $v_{k,i}$ itself).

For clarity, we formulate the normalization methods for training GNNs from different scales, as illustrated in Figure 1 (a)-(d), including node-wise normalization, adjacency-wise normalization, graph-wise normalization and batch-wise normalization.

Node-wise Normalization. Node-wise normalization on graph, denoted as GN_n, considers to normalize the feature vector $\boldsymbol{h}_{v_{k,i}}$ of each node $v_{k,i}$ and compute the first and the second statistics over the d entries of the feature vector $\boldsymbol{h}_{v_{k,i}}$ as follows:

$$\hat{\boldsymbol{h}}^{(n)}_{v_{k,i}} = \frac{\boldsymbol{h}_{v_{k,i}} - \mu^{(n)}_{k,i}\mathbf{1}}{\sigma^{(n)}_{k,i}}, \qquad \mu^{(n)}_{k,i} = \frac{1}{d}\sum_{j=1}^{d} h^j_{v_{k,i}}, \quad \sigma^{(n)}_{k,i} = \sqrt{\frac{1}{d}\sum_{j=1}^{d}(h^j_{v_{k,i}} - \mu^{(n)}_{k,i})^2}, \tag{1}$$

where $\mu^{(n)}_{k,i}$ and $\sigma^{(n)}_{k,i}$ are the mean and the standard deviation along the feature dimension for node $v_{k,i}$, and $\mathbf{1} \in \mathcal{R}^d$ represents a d-dimension vector of all 1. Note that node-wise normalization is equivalent to applying LN to each node of the graph to reduce the "covariate shift" problem[1].

Adjacency-wise Normalization. Each node in a graph has its neighbors. However, node-wise normalization performs normalization on each node individually and ignores the local structure in the graph. Here, we propose to take into account the adjacency structure in the graph and normalize the node features of the adjacent neighbors. We term it as adjacency-wise normalization on graph, denoted as GN_a. For each node $v_{k,i}$ in graph $\mathcal{G}_k$, we consider its adjacent nodes $\mathcal{N}(v_{k,i})$, as illustrated in Figure 1 (b). Specifically, the adjacency-wise normalization for node $v_{k,i}$ is defined as follows:

$$\hat{\boldsymbol{h}}^{(a)}_{v_{k,i}} = \frac{\boldsymbol{h}_{v_{k,i}} - \mu^{(a)}_{k,i}\mathbf{1}}{\sigma^{(a)}_{k,i}}, \tag{2}$$

$$\mu^{(a)}_{k,i} = \frac{1}{|\mathcal{N}(v_{k,i})| \times d}\sum_{j' \in \mathcal{N}(v_{k,i})}\sum_{j=1}^{d} \boldsymbol{h}^j_{v_{k,j'}}, \tag{3}$$

$$\sigma^{(a)}_{k,i} = \sqrt{\frac{1}{|\mathcal{N}(v_{k,i})| \times d}\sum_{j' \in \mathcal{N}(v_{k,i})}\sum_{j=1}^{d}(h^j_{v_{k,j'}} - \mu^{(a)}_{k,i})^2}, \tag{4}$$

where $\mu^{(a)}_{k,i}$ and $\sigma^{(a)}_{k,i}$ are the first order and second order statistics over the adjacent nodes.[2]

Graph-wise Normalization. Note that nodes belonging to graph $\mathcal{G}_k$ naturally form a group. In order to preserve the global structure of a graph, we propose to normalize the node feature based on the first and the second order statistics computed over graph $\mathcal{G}_k$. Specifically, we define a graph-wise

[1]The node-wise normalization method in Equation (1) can also be used to normalize the feature at each edge, as illustrated in Figure 1 (e).

[2]For the edge $e_{k,i,j}$, as Figure 1 (f), the adjacent edges $\mathcal{N}(e_{k,i})$ can be considered in a similar way.

normalization on graph, denoted as GN_g, for node $v_{k,i}$ as follows:

$$\hat{\boldsymbol{h}}^{(g)}_{v_{k,i}} = (\boldsymbol{h}_{v_{k,i}} - \boldsymbol{\mu}^{(g)}_k)\Lambda_k^{-1}, \tag{5}$$

$$\boldsymbol{\mu}^{(g)}_k = \frac{1}{|\mathcal{G}_k|}\sum_{v_{k,i}\in\mathcal{G}_k} \boldsymbol{h}_{v_{k,i}}, \tag{6}$$

where $\boldsymbol{\mu}^{(g)}_k$ and Λ_k are the first order and the second order statistics in graph $\mathcal{G}_k$ in which Λ_k is a diagonal matrix with diagonal entry Λ^{jj}_k is defined as

$$\Lambda^{jj}_k = \sqrt{\frac{1}{|\mathcal{G}_k|}\sum_{v_{k,i}\in\mathcal{G}_k}(h^j_{v_{k,i}} - \mu^{(g),j}_k)^2}. \tag{7}$$

If the task has only a single graph, then graph-wise normalization is similar to BN. However, unlike in BN, graph-wise normalization does not use a smoothing average updater.[3]

Batch-wise Normalization. To keep training stable, BN is one of the most critical components. For a mini-batch, there are N graphs. We compute the mean and standard deviation across over the graphs of a mini-batch, then each node feature $\boldsymbol{h}_{v_{k,i}}$ is normalized as follows:

$$\hat{\boldsymbol{h}}^{(b)}_{v_{k,i}} = (\boldsymbol{h}_{v_{k,i}} - \boldsymbol{\mu}^{(b)})\Lambda^{-1}, \tag{8}$$

$$\boldsymbol{\mu}^{(b)} = \frac{1}{T}\sum_{k=1}^{N}\sum_{i=1}^{|\mathcal{G}_k|}\boldsymbol{h}_{v_{k,i}}, \tag{9}$$

where $T = \sum_{k=1}^{N}|\mathcal{G}_k|$ means the total number of the nodes in the N graphs and Λ is a diagonal matrix to keep the standard deviation of the note features over N graphs in which the diagonal entry Λ^{jj} is defined

$$\Lambda^{jj} = \sqrt{\frac{1}{T}\sum_{k=1}^{N}\sum_{i=1}^{|\mathcal{G}_k|}(h^j_{v_{k,i}} - \mu^{(b),j})^2}. \tag{10}$$

Note that batch-wise normalization on graph, named as GN_b, is effectively BN (Ioffe & Szegedy, 2015), which performs normalization over all nodes of the N graphs in a mini-batch.

The normalization methods applying to node features $\boldsymbol{h}_{v_{k,i}}$ can also be extended to edge features $\boldsymbol{h}_{e_{k,i,j}}$ where $\boldsymbol{h}_{e_{k,i,j}}$ denotes the feature of edge $e_{i,j}$ in graph $\mathcal{G}_k$, as illustrated in Figure 1 (e)-(h).

Remark. The properties of the four normalization methods are summarized as follows.

- Node-wise normalization only considers to normalize the feature of each node individually but ignores the adjacency structure and the whole graph structures. It is equivalent to LN (Ba et al., 2016) in operation.
- Adjacency-wise normalization takes the adjacent nodes into account, whereas graph-wise normalization takes into account the features of all nodes in a graph.
- Batch-wise normalization is the same as the standard batch normalization (Ioffe & Szegedy, 2015). If the task only involves a single graph, then the batch-wise normalization is similar to the graph normalization except that momentum average used in batch-wise normalization is not used in the graph-wise normalization.

3 Learning Attentive 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 sacrifice the performance of a GNN.

To remedy these issues, we propose to learn *attentive graph normalization* for training GNNs by optimizing a weighted combination of the normalization methods. Specifically, we combine the node feature $\hat{\boldsymbol{h}}_{v_{k,i}}$ under different normalization methods as follows:

$$\hat{\boldsymbol{h}}_{v_{k,i}} = \gamma(\boldsymbol{\alpha}^{(n)}\odot\hat{\boldsymbol{h}}^{(n)}_{v_{k,i}} + \boldsymbol{\alpha}^{(a)}\odot\hat{\boldsymbol{h}}^{(a)}_{v_{k,i}} + \boldsymbol{\alpha}^{(g)}\odot\hat{\boldsymbol{h}}^{(g)}_{v_{k,i}} + \boldsymbol{\alpha}^{(b)}\odot\hat{\boldsymbol{h}}^{(b)}_{v_{k,i}}) + \boldsymbol{\beta}, \tag{11}$$

[3]For the edges $\mathcal{E}_k$ of graph $\mathcal{G}_k$ (Figure 1 (g)), we can also define the same normalization.

where $\boldsymbol{\alpha}^{(n)}, \boldsymbol{\alpha}^{(a)}, \boldsymbol{\alpha}^{(g)}$ and $\boldsymbol{\alpha}^{(b)} \in \mathbb{R}^d$ are trainable gate parameters with the same dimension as $\boldsymbol{h}_{v_{k,i}}$,$\gamma \in \mathbb{R}$ and $\boldsymbol{\beta} \in \mathbb{R}^d$ are the trainable scale and shift parameters, respectively.

Note that we attempt to use the learned $\boldsymbol{\alpha}^{(n)}, \boldsymbol{\alpha}^{(a)}, \boldsymbol{\alpha}^{(g)}$ and $\boldsymbol{\alpha}^{(b)}$ indicate the contribution of the corresponding normalized feature to $\hat{\boldsymbol{h}}_{v_{k,i}}$. Thus, we impose normalization constraints on each dimension of $\boldsymbol{\alpha}^{(n)}, \boldsymbol{\alpha}^{(a)}, \boldsymbol{\alpha}^{(g)}$ and $\boldsymbol{\alpha}^{(b)}$ that $\alpha_j^{(u)} \in [0, 1]$ where $u \in \{n, a, g, b\}$ and $j = 1, \cdots, d$, and $\sum_{u \in \{n,a,g,b\}} \alpha_j^{(u)} = 1$ where $j = 1, \cdots, d$. In this way, if a normalization method is better for a specific task, the learned corresponding weights will be higher than others. Thus, we term the learned attentive graph normalization method in Equation (11) as *Automatic Graph Normalization* (AGN). In AGN, multiple normalization methods collaborate and compete with each others to improve the performance of GNNs.

Different normalization methods are suitable for different tasks. In AGN, the attention weights $\boldsymbol{\alpha}^{(n)}, \boldsymbol{\alpha}^{(a)}, \boldsymbol{\alpha}^{(g)}$ and $\boldsymbol{\alpha}^{(b)}$ are optimized for a specific task and thus the best-performing normalization method will have a set of significant weights. Therefore AGN can serve as an effective strategy to select one of the best-performing normalization method or the best combination of multiple normalization methods for a specific task.

4 EXPERIMENTS

We evaluate GN_n, GN_a, GN_g, GN_b, and AGN under three GNN frameworks, including Graph Convolution Network (GCN) , Graph Attention Network (GAT) and GatedGCN. We also assess the performance of GNNs without normalization layer named as "No Norm". The benchmark 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 effect 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). The detailed statistics of the datasets are presented in Appendix C.1.

The implementations of GCN, GAT and GatedGCN are from GNN benchmarking framwork[4]. The hyper-parameters and optimizers of the models and the details of the experimental settings are kept the same as in (Dwivedi et al., 2020). We run experiments on CLUSTER and PATTERN datasets with GNNS of depth of layers $L = \{4, 16\}$, respectively. For the other datasets, we fix the number of GCN layers to $L = 4$.

4.1 NODE CLASSIFICATION

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

As can be read, graph-wise normalization (GN_g) outperforms batch-wise normalization (GN_b) obviously in most situations. For instance, when the depth of GNNs is 4, GatedGCN with GN_g achieves 9% improvement 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 normalization only for each graph. Thus, GN_g can learn the dedicated information of each graph and normalize the feature of each graph into 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. AGN gets comparable results for different GNNs and the results of AGN are close to the best results in most cases due to its flexibility and adaptability. AGN can adaptively learn the optimal combination of the normalization methods which better adapt to the node classification task.

Moreover, we apply node classification to key information extraction on SROIE which consists of 626 receipts for training and 347 receipts for testing. Each image is annotated with text bounding

[4]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
	AGN	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
	AGN	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: Results on CLUSTER and PATTERN. Red: the best model,Violet: good models.

boxes (bbox) and the transcript of each text bbox. There are four entities to extract (i.e., Company, Date, Address and Total) from a receipt, as shown in Appendix C.2. For a receipt image, each text bounding box is label with five classes (i.e., Total, Date, Address, Company and Other). Then, the key information extraction is treated as node classification and we treat each text bounding box in a receipt image 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). The entity is recorded as "extracted successful" if and only if all nodes of each entity are classified correctly. In this experiment, we use GatedGCN. We compute the mean accuracy for each text field and the average accuracy for each receipt and show the results in Table 2.

Text Field	No Norm	GN_n	GN_a	GN_g	GN_b	AGN
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 is interesting that the graph of each receipt is special with neighboring nodes that usually belong to different classes. Thus, the performance of adjacency-wise normalization is worse than node-wise normalization.

4.2 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. Experimental results are shown in Table 3. All the five normalization methods achieve similar performance on dataset TSP. Compared with others, the results of AGN are very stable. For each GNN, the result of AGN is comparable with the best result.

Dataset COLAB contains only a graph with 235,868 nodes. Due to out-of-memory problem, we do not report the results of GN_a and AGN. Compared with GNNs with normalization layer, the results of GNNs without normalization layer (No Norm) seriously degenerates. GN_g performs better than GN_b. GatedGCN with GN_g achieves the best result.

4.3 Graph Classification and Graph Regression

Graph classification is to assign one label to each graph. We conduce experiments on CIFAR10 and MNIST. Average class accuracy is reported in Table 4. ZINC is a dataset for graph regression. The mean absolute error (MAE) between the predicted value and the ground truth is calculated for each

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
AGN	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.

group. Average MAE also is reported in Table 4. We can see that GN_b outperforms others in most cases. GN_g does not work well on graph classification and regression. Furthermore, GN_g affects the performance of AGN. In AGN, the normalized features of GN_n, GN_a, GN_g, and GN_b are integrated and automatically paid more attention to GN_b due to its outstanding performance. Therefore, the performance of AGN 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
AGN	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
AGN	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
AGN	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: Results on MNIST, CIFAR10 and ZINC. Red: the best model,Violet: good models.

4.4 Analysis and Further Investigations

The above experimental results indicate 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 AGN which integrates several normalization methods into a unified optimization framework achieves competitive results compared with the best single normalization method on various datasets.

Behaviours of Learned Weights in AGN. To gain more insight into AGN, we conduct a set of experiments to analyze the effect of each normalization method on different datasets. Note that AGN combines the results of several normalization methods and $\{\boldsymbol{\alpha}^{(u)}\}_{u\in n,a,g,b}$ in Equation (11) indicate the importance of the corresponding normalization methods, respectively. We initialize the weights $\{\boldsymbol{\alpha}^{(u)}\}_{u\in n,a,g,b}$ in each layer with the equal values, i.e., $\alpha_j^{(m)} = 0.25$ for $j = 1, \cdots, d$ and $m \in \{n, a, g, b\}$. In the training phase, the value of each component of $\{\boldsymbol{\alpha}^{(u)}\}_{u\in n,a,g,b}$ changes between 0 and 1. On each dataset, we investigate the learned optimal weights on average at different layers of GatedGCN. Particularly, We collect the learned weights of each normalization method in each layer and calculate the averaged weights of each normalization method over all of the d entries of $\alpha^{(u)}$. We show the learned weights on average in Figure 2. As can be observed, the learned weights of each normalization method on average not only change for different dataset but also

	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
Combined	69.16	84.64	95.4	81.11	97.52	67.88	0.4371

Table 5: Performance of different normalization methods on seven benchmark datasets. For each dataset, we give the performance of the best two normalization methods and a new normalization method which is combined the two best-performing normalization methods as Equation (11).

vary for different layers. This implies that different layers prefer to different normalization method in order to yield good performance. We can also observe that the weights on GN_g are larger than others on node classification tasks and GN_b is more important on others. Our proposed AGN has the ability to automatically choose the suitable normalization method for a specific task.

Evaluation on Selected Normalization Methods via AGN. To further evaluate the performance of the selected normalization methods, we select the two best-performing normalization methods, combine them into a new normalization method as in Equation (11), and conduct experiments on each dataset. The experimental results are listed in Table 5. We can read that the combined normalization method obtains the comparable results with the best normalization method. Therefore these results show that the learnt weights indicate whether the corresponding normalization method is suitable for the current task.

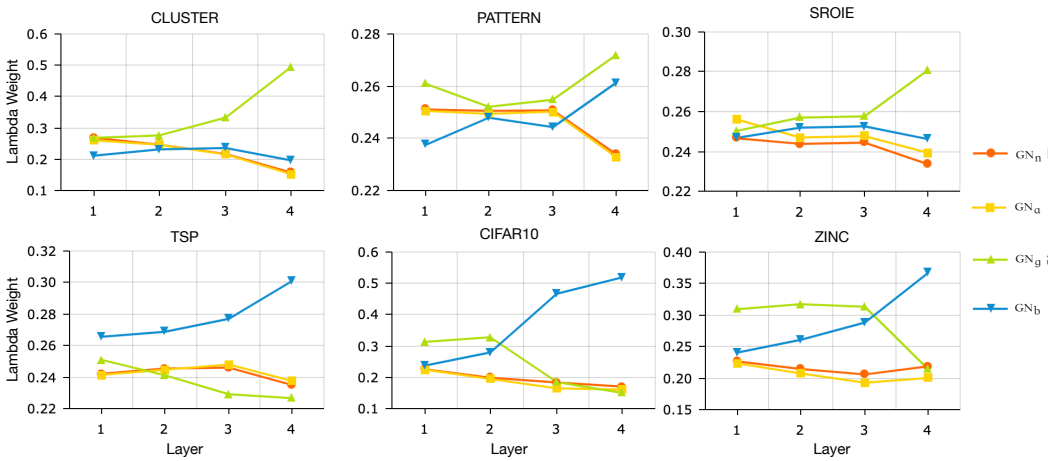

Figure 2: Learnt weight distributions of normalization methods along with layers on different tasks.

Training Loss and Test Accuracy Curves vs. Iteration Steps. To show the effect of different normalization methods, we draw the curves of training loss and test result with respect to the iteration steps in Figures 4 and 5 in Appendix C.3. We see that when a proper normalization method is used, the training loss converges faster and better test accuracy can be obtained.

5 Conclusions

We formulated four normalization methods for training GNNs at different scales: node-wise normalization, adjacency-wise normalization, graph-wise normalization, and batch-wise normalization. Particularly, the adjacency-wise normalization and graph-wise normalization are graph-aware normalization methods, which are designed with respect to the local and the global structure of the graph, respectively. Moreover, we proposed a novel optimization framework, called Automatic Graph Normalization, to learn attentive graph normalization by optimizing an attentively weighted combination of multiple graph normalization methods. We conducted extensive experiments on seven benchmark datasets at different tasks and confirmed that the graph-aware normalization methods and the automatically learned graph normalization method lead to promising results and that the learned optimal weights suggest more appropriate normalization methods for specific tasks.

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.

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.

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.

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.

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.

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.

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.

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

$$\boldsymbol{h}_v^{\ell+1} = \psi^{\ell+1}(\mathcal{C}\{\boldsymbol{h}_v^{\ell}, \mathcal{M}\{\phi^{\ell+1}(\boldsymbol{h}_u^{\ell}) | u \in \mathcal{N}(v)\}\}) \tag{12}$$

where $\boldsymbol{h}_u^\ell$ denote the feature vector at the ℓ-th layer 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 $\boldsymbol{h}_v^0 = \boldsymbol{x}_v$ 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:

$$\boldsymbol{h}_v^{\ell+1} = \mathrm{ReLU}(\frac{1}{deg_v} \sum_{u \in \mathcal{N}(v)} W^\ell \boldsymbol{h}_u^\ell), \tag{13}$$

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 key issue of the GCN is an over-smoothing problem, which can be partially eased by residual shortcut across layers. Another 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:

$$\boldsymbol{h}_v^{\ell+1} = \sigma(\sum_{u \in \mathcal{N}(v)} a_{u,v}^\ell W^\ell \boldsymbol{h}_u^\ell), \tag{14}$$

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

$$a_{u,v}^\ell = \frac{\exp(g(\boldsymbol{\alpha}^T[W^\ell \boldsymbol{h}_u^\ell || W^\ell \boldsymbol{h}_v^\ell]))}{\sum_{k \in \mathcal{N}(v)} \exp(g(\boldsymbol{\alpha}^T[W^\ell \boldsymbol{h}_k^\ell || W^\ell \boldsymbol{h}_v^\ell]))}, \tag{15}$$

where $g(\cdot)$ is a $\mathrm{LeaklyReLU}$ activation function, $\boldsymbol{\alpha}$ 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. Unfortunately, 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. Thus, for node v, the formulation for updating the feature representation is:

$$\boldsymbol{h}_v^{\ell+1} = \boldsymbol{h}_v^\ell + \mathrm{ReLU}(\mathrm{BN}(W^\ell \boldsymbol{h}_v^\ell + \sum_{u \in \mathcal{N}(v)} e_{vu}^\ell \odot U^\ell \boldsymbol{h}_u^\ell)), \tag{16}$$

where $W^\ell, U^\ell \in \mathcal{R}^{d \times d}$, $\odot$ is the Hadamard product, and the edge gates $\boldsymbol{e}_{v,u}^\ell$ are defined as:

$$\begin{aligned} \boldsymbol{e}_{v,u}^\ell &= \frac{\sigma(\hat{\boldsymbol{e}}_{v,u}^\ell)}{\sum_{u' \in \mathcal{N}(v)} \sigma(\hat{\boldsymbol{e}}_{v,u'}^\ell) + c}, \\ \hat{\boldsymbol{e}}_{v,u}^\ell &= \hat{\boldsymbol{e}}_{v,u}^{\ell-1} + \mathrm{ReLU}(\mathrm{BN}(A^\ell \boldsymbol{h}_v^{\ell-1} + B^\ell \boldsymbol{h}_u^{\ell-1} + C^\ell \hat{\boldsymbol{e}}_{v,u}^{\ell-1})), \end{aligned} \tag{17}$$

where $\sigma(\cdot)$ is a sigmoid function, c 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 $\hat{\boldsymbol{e}}_{v,u}$ at each layer.

B NORMALIZATION METHODS

B.1 BATCH NORMALIZATION

Batch normalization (BN) (Ioffe & Szegedy, 2015) has become one of the critical components in training a deep neural network, which normalizes the features by using the first order and the second order statistics computed within a mini-batch. BN can reduce the internal covariate shift problem and accelerate the training process. We briefly introduce the formulation of BN. Firstly, $\mathcal{H} = \{\boldsymbol{h}_1, \boldsymbol{h}_2, ..., \boldsymbol{h}_m\} \in \mathbb{R}^{d \times m}$ is denoted as the input of a normalization layer, where m is the batch size and $\boldsymbol{h}_i$ represents a sample. Then, $\boldsymbol{\mu}^{(m)} \in \mathbb{R}^d$ and $\boldsymbol{\sigma}^{(m)} \in \mathbb{R}^d$ denote the mean vector and the variance vector of the m sample in $\mathcal{H}$, respectively. BN normalizes each dimension of features

using $\boldsymbol{\mu}^{(m)}$ and $\boldsymbol{\sigma}^{(m)}$ as:

$$\begin{aligned}\hat{\boldsymbol{h}} &= \gamma(\boldsymbol{h} - \boldsymbol{\mu}^{(m)})./\boldsymbol{\sigma}^{(m)} + \boldsymbol{\beta}, \\ \boldsymbol{\mu}^{(m)} = \frac{1}{m}\sum_{j=1}^{m}\boldsymbol{h}_j, \quad \sigma_i^{(m)} &= \sqrt{\frac{1}{m}\sum_{j=1}^{m}(h_{ij} - \mu_i^{(m)})^2}, \\ \boldsymbol{\mu} &= \alpha\boldsymbol{\mu} + (1-\alpha)\boldsymbol{\mu}^{(m)}, \quad \sigma^2 = \alpha\sigma^2 + (1-\alpha)(\sigma^{(m)})^2,\end{aligned} \tag{18}$$

where $./$ means element-wise division, γ and $\boldsymbol{\beta}$ are trainable scale and shift parameters, respectively. In Equation (18), $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ denote the running mean vector and the variance vector to approximate the mean vector and the variance vector of the dataset. 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 statistics. 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 $\boldsymbol{h}_j \in \mathbb{R}^d$, LN computes mean $\mu_j^{(L)}$ and variance $\sigma_j^{(L)}$ across the feature dimension. The normalization equations of LN are as follows:

$$\begin{aligned}\hat{\boldsymbol{h}}_j &= \boldsymbol{\gamma} \odot \frac{\boldsymbol{h}_j - \mu_j^{(L)}\mathbf{1}}{\sigma_j^{(L)}} + \boldsymbol{\beta}, \\ \mu_j^{(L)} = \frac{1}{d}\sum_{i=1}^{d} h_{ij}, \quad \sigma_j^{(L)} &= \sqrt{\frac{1}{d}\sum_{i=1}^{d}(h_{ij} - \mu_j^{(L)})^2},\end{aligned} \tag{19}$$

where $\hat{\boldsymbol{h}}_j \in \mathbb{R}^d$ is the normalized feature vector, $\mathbf{1} \in \mathbb{R}^d$ is a d dimension vector of 1's, $\boldsymbol{\gamma} \in \mathbb{R}^d$ and $\boldsymbol{\beta} \in \mathbb{R}^d$ are scale and shift parameters of 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. Thus, 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.

C.2 SROIE

For a receipt, each text bounding box (bbox) is viewed as a node of a graph. The positions and the attributes of the bounding box, and the corresponding text are used as the node feature. To describe the relationships among all the text bounding boxes on 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, we connect v_i and v_j by an edge $e_{i,j}$. Since that the relative positions of two text bounding boxes are important for node classification, we encode the relative coordinates of v_i and v_j to represent the edge e_{ij}. In this way, such an information extraction task from a receipt can be treated as a node classification task on a graph. Our goal is to label each node (i.e., text bounding box) with five different classes: "Company", "Date", "Address", "Total" and "Other". Since that GatedGCN explicitly exploits edge features and has achieved state-of-the-art performance on various tasks, we use GatedGCN with 8 GCN layers for this task.

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.

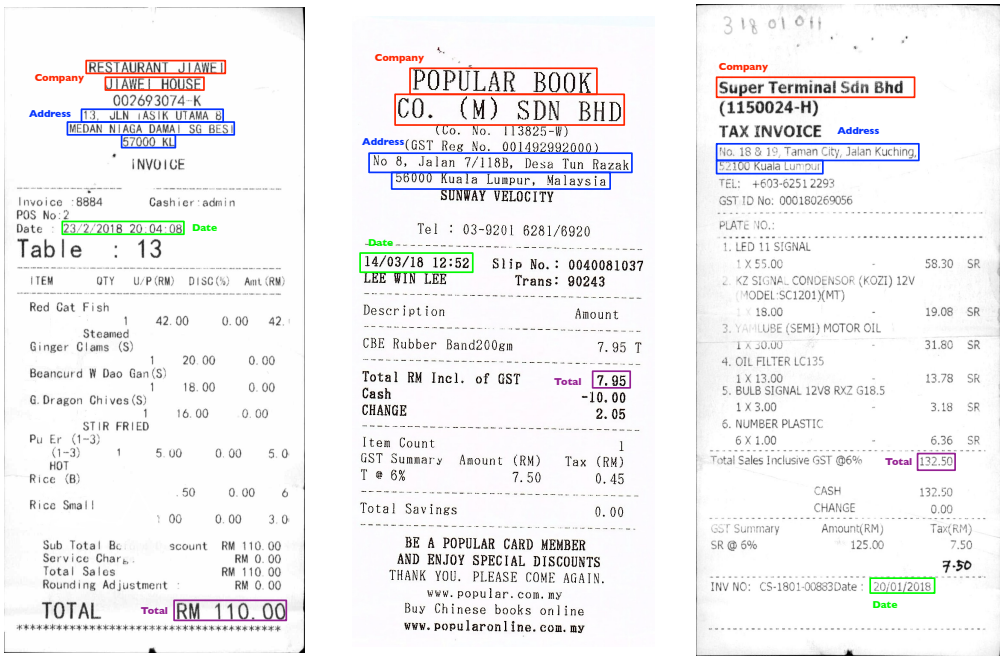

Figure 3: Sample images of the SROIE dataset. Four entities are highlighted in different colors. "Company", "Address", "Date", and "Total" are marked with Red, Blue, Yellow, and Purple individually. The "Company" and the "Address" entities usually consist of several text lines.

C.3 Training and test curves for node-wise, adjacency-wise, graph-wise, batch-wise normalization and auto graph normalization.

D Acknowledgement

We would like to thank Vijay *et al.* to release their benchmarking code for our research. We also want to thank the DGL team for their excellent toolbox.

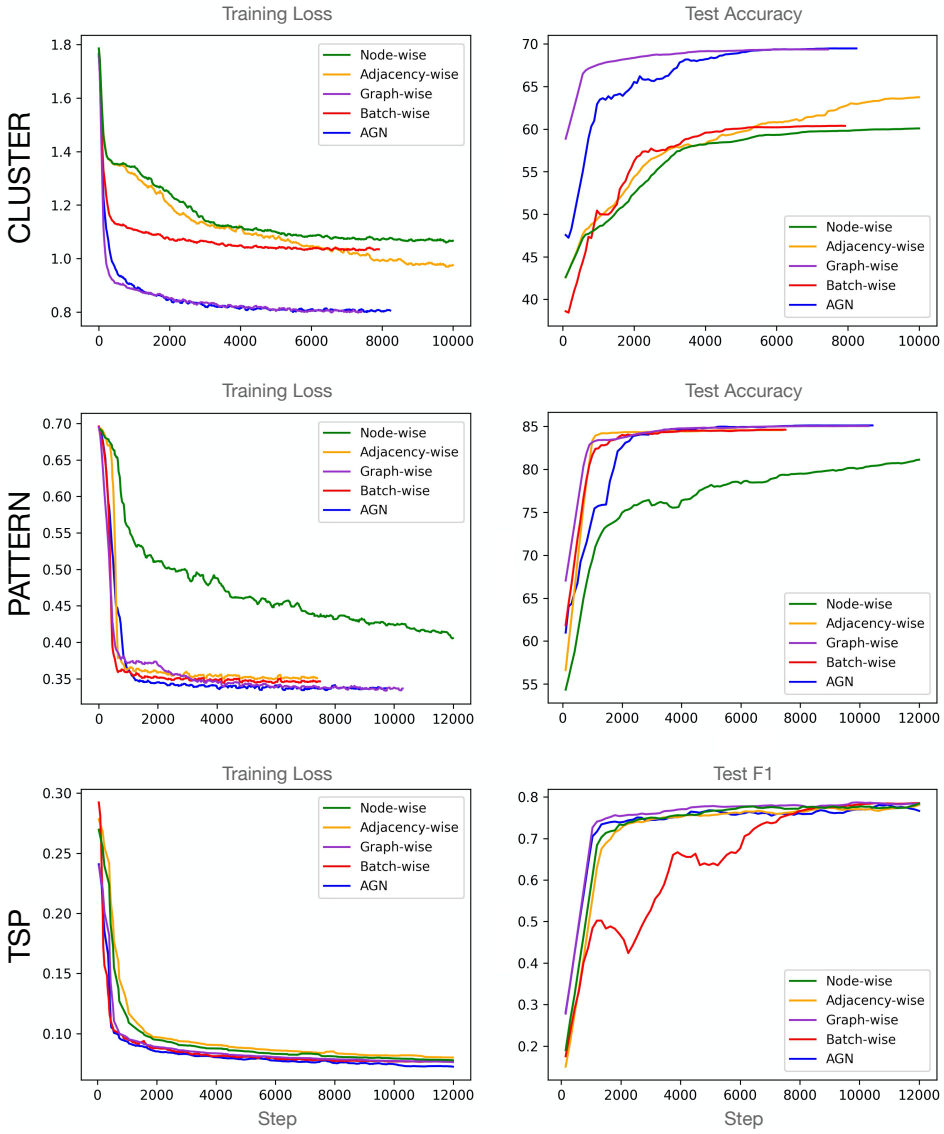

Figure 4: Training loss and test result of GatedGCN on CLUSTER, PATTERN and TSP vs. the number of steps, with different normalization methods.

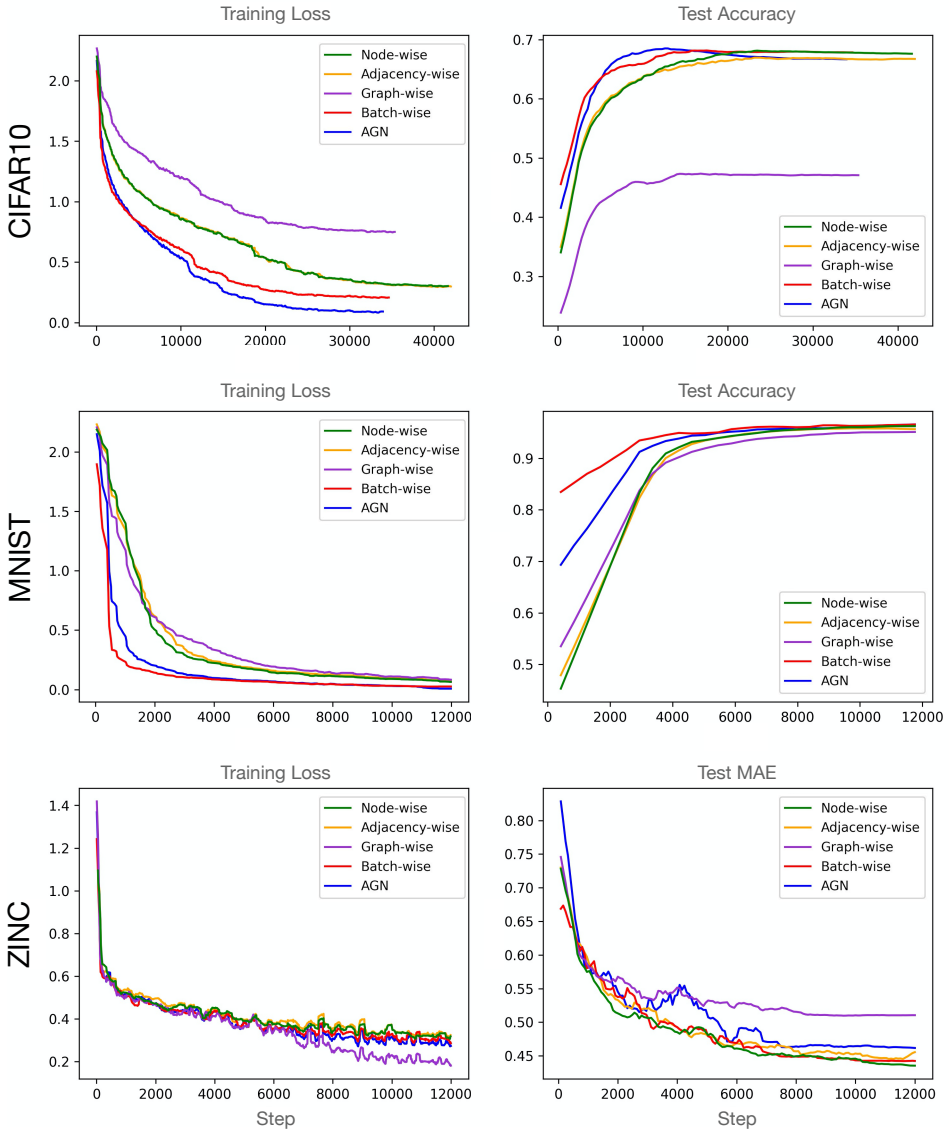

Figure 5: Training loss and test result of GatedGCN on CIFAR, MNIST and ZINC vs. the number of steps, with different normalization methods.