Node-wise Localization of Graph Neural Networks (Supplemental Material)

Zemin Liu[1] , **Yuan Fang**[1] , **Chenghao Liu**[2] and **Steven C.H. Hoi**[1,2]
[1]Singapore Management University, Singapore
[2]Salesforce Research Asia, Singapore
{zmliu, yfang}@smu.edu.sg, {chenghao.liu, shoi}@salesforce.com

A Additional Implementation Details

We implemented the proposed LGNN using TensorFlow 1.13.1 in Python 3.6.5. All experiments were conducted on a Linux workstation with a 6-core 3.6GHz CPU, 128GB DDR4 memory and two RTX 2080Ti GPUs.

For the baselines DeepWalk[1] and Planetoid[2], we used their respective authors' implementations. Note that DeepWalk is unsupervised and does not consider node features. Therefore, after obtaining the node embeddings using DeepWalk, we follow the standard practice to further train a logistic regression classifier for node classification, using the concatenation of node embeddings and node features as the final feature vectors. For the baselines GCN[3], GAT[4] and GIN[5], their authors' implementations load in the entire graph during training. To enable more scalable and efficient training, we adapted their original code to work with mini-batches of nodes and their neighborhoods during training. Our adapted implementations achieved near-identical performance on the benchmark datasets as reported in the original papers. For GNN-FiLM[6], using the authors' code as a reference, we implemented it on three different GNN architectures, namely, GCN, GAT and GIN, and obtained three corresponding versions GCN-FiLM, GAT-FiLM and GIN-FiLM.

B Additional Experimental Settings

We further discuss more settings omitted in the main paper due to space constraint. For the optimization of the baseline GNNs and our LGNN, we used the Adam optimizer with learning rate 0.005. The regularization for GNN parameters were all set to $\lambda_G = 0.0005$. We used the Exponential Linear Unit (ELU) as the activation function. We also applied dropout with a rate of 0.6 for the baselines, a rate of 0.4 for our LGCN and LGIN, and 0.6 for our LGAT (0.7 only for LGAT on Citeseer dataset), based on empirical tuning on the validation set. Moreover, for GAT, we adopted 8 heads for the attention mechanism in the first aggregation layer; for GIN, we adopted an MLP with one hidden layer; for LGNN, as the input dimension of the node features is fairly large, we utilized two-dense layers to generate the node-specific vectors for scaling and shifting. Furthermore, we employed LeakyReLU as the activation function for the generation of all the transformation vectors.

[1]https://github.com/phanein/deepwalk
[2]https://github.com/kimiyoung/planetoid
[3]https://github.com/tkipf/gcn
[4]https://github.com/PetarV-/GAT
[5]https://github.com/weihua916/powerful-gnns
[6]https://github.com/Microsoft/tf-gnn-samples

C Analysis of Number of Parameters

To validate that the power of the proposed LGNN is not simply derived from an increased number of model parameters, we calculated the number of parameters of different models. The details of the calculation are as follows.

- DeepWalk: This model adopts a direct embedding lookup. Each node has two learnable parameter vectors, one serving as the node representation, and the other serving as the weight vector when it appears as a context node. There are no other learnable parameters. The total number of parameters are therefore $2d \cdot |V|$, where d is the dimension of the vectors, and $|V|$ denotes the number of nodes in the graph.
- Planetoid: Similar to DeepWalk, a direct embedding lookup is used. However, it utilizes an additional hidden layer to embed the input features and transform the node embeddings, as well as a classification layer for the final output. The total number of parameters are therefore $2d \cdot |V| + d_1 \cdot d_0 + d_1 \cdot d + 2K \cdot d_1$, where d_0 is the dimension of the input features, d_1 is the dimension of the hidden layer, and K is the number of classes. Note that $|V|$ and d_0 can be large whereas d_1, d and K are typically small constants. Thus, the total number of parameters is dominated by $2d \cdot |V| + d_1 \cdot d_0$.
- GCN: The learnable parameters are the weights $\mathbf{W}^1, \mathbf{W}^2, ...\mathbf{W}^l$ in each layer. As we adopt a two-layer architecture where the dimension of the last layer is commonly set to the number of classes, the number of parameters amounts to $d_1 \cdot d_0 + K \cdot d_1$.
- GAT: There are h weight matrices and h attention vectors in the first layer, where h is the number of attention heads. The size of each matrix is $d_1 \times d_0$ and that of each vector is $2d_1$. Overall, the number of parameters is given by $h(d_1 \cdot d_0 + 2d_1) + h \cdot K \cdot d_1 + 2K$. Note that d_1 and K are small constants, which means the parameters are roughly h times as many as those of GCN.

Table I: Average classification performance with standard deviation (percent) over 10 runs, using a validation set of 100 nodes only. Improvements of LGNN are relative to the best performing baseline with the corresponding GNN architecture.

Methods	Cora		Citeseer		Amazon		Chameleon	
	Accuracy	Micro-F	Accuracy	Micro-F	Accuracy	Micro-F	Accuracy	Micro-F
GCN	79.9±1.8	79.2±1.7	69.2±1.2	67.3±1.1	82.4±1.2	81.0±1.6	33.8±3.7	31.4±5.4
GCN-FiLM	74.3±1.1	72.9±1.2	66.6±1.3	64.6±1.6	78.9±2.3	76.5±3.0	42.1±1.7	38.3±2.5
LGCN	**83.0**±0.8	**81.7**±0.9	**71.6**±0.6	**69.7**±0.6	**83.2**±1.7	**81.5**±2.7	**50.5**±1.1	**49.7**±0.8
(improv.)	(3.9%)	(3.2%)	(3.5%)	(3.6%)	(1.0%)	(0.6%)	(19.9%)	(29.8%)
GAT	82.6±0.8	81.8±0.7	71.0±1.1	69.3±1.3	82.8±0.8	80.7±1.4	47.6±1.8	47.1±2.4
GAT-FiLM	82.5±1.1	81.4±0.9	71.2±0.9	69.3±0.8	83.1±0.8	81.2±1.3	39.8±5.5	38.1±7.1
LGAT	**83.1**±0.8	**82.1**±0.5	**71.3**±1.3	**69.4**±1.2	**83.5**±0.6	**82.0**±0.8	**51.2**±1.9	**50.0**±1.9
(improv.)	(0.6%)	(0.4%)	(0.1%)	(0.1%)	(0.5%)	(1.0%)	(7.6%)	(6.2%)
GIN	79.4±1.1	78.1±0.9	66.0±1.2	63.0±1.2	78.0±2.0	77.1±2.6	38.2±4.8	33.1±4.9
GIN-FiLM	77.5±1.0	76.6±0.9	63.0±3.5	60.6±4.0	79.9±2.3	78.2±2.9	37.5±3.0	32.0±3.0
LGIN	**82.1**±0.7	**81.1**±0.5	**71.1**±0.7	**69.2**±0.6	**83.2**±1.5	**81.4**±2.4	**47.5**±1.3	**46.6**±1.2
(improv.)	(3.4%)	(3.8%)	(7.7%)	(9.8%)	(4.1%)	(4.1%)	(24.3%)	(40.8%)

- GIN: Similar to GCN, there is a series of one-layer MLPs in each aggregation layer. Each one-layer MLP is parameterized by one weight matrix of size $d_l \times d_{l-1}$ and a bias vector of size d_l. The total number of parameters is thus $d_1 \cdot d_0 + d_1 + K \cdot d_1 + K$. Note that this number is dominated by $d_1 \cdot d_0 + K \cdot d_1$, which is the same as GCN.
- GNN-FiLM: It contains all the parameters of the corresponding base GNN model. Additionally, it further generates the node-specific scaling and shifting vectors in each layer, and each type of transformation vector corresponds to a parameter matrix. Therefore, it has additional $2d_0 \cdot d_1 + 2K \cdot d_1$ parameters.
- LGNN: Similar to GNN-FiLM, our model naturally has all the parameters of the global GNN model. Additionally, we need to generate four types of vectors: node-specific scaling and shifting vectors, and edge-specific scaling and shifting vectors. For each type of node-specific vectors, there are d_0^2 parameters in the first GNN layer, and d_1^2 parameters in the second GNN layer. However, due to the potentially high-dimensional node features (*i.e.*, large d_0), we employed dense layers where the first dense layer has dimension d_1 in order to generate the node-specific vectors in the first GNN layer. That means, there are instead only $2d_0 \cdot d_1$ parameters in the first GNN layer. For each type of edge-specific vectors, there are $2d_0 \cdot d_1$ parameters in the first GNN layer, and $2d_1 \cdot K$ parameters in the second GNN layer. Overall, we have additional $8d_0 \cdot d_1 + 2d_1^2 + 4d_1 \cdot K$ parameters.

D Evaluation on Smaller Validation Sets

GNNs run the risk of overfitting to the validation set (Shchur et al. 2018). In other words, models with more parameters tend to perform better given a larger validation set. To evaluate the influence of validation set on GNN-based models, we further conduct experiments using a smaller validation set of only 100 nodes (the standard splits used in the main paper consist of 500 nodes for validation). We report the results in Table I, and we can draw the following conclusions.

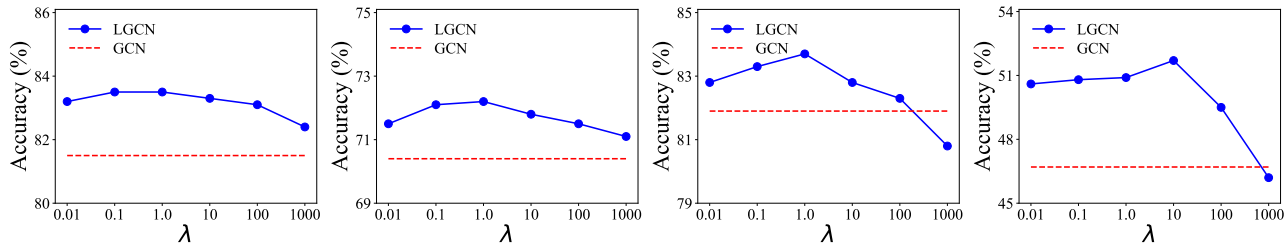
Table II: Time cost comparison. Train/test refers to wall-clock time in ms, and train time is the average of one epoch.

	Cora			Amazon		
	FLOPs	Train	Test	FLOPs	Train	Test
GCN	821M	32.4	37.2	2,188M	37.5	44.2
LGCN	15,158M	65.3	130.6	42,608M	109.1	196.5
GAT	5,268M	54.8	37.2	14,220M	55.3	222.2
LGAT	119,410M	324.8	130.6	336,161M	563.4	1076.3
GIN	821M	23.6	206.2	2,190M	25.9	50.3
LGIN	15,792M	80.4	758.4	44,390M	125.2	228.0

Firstly, with a smaller validation set, all the models including GNNs, GNN-FiLMs and LGNNs generally achieve worse performance than using a large validation set (as shown in Table 3 of the main paper). This implies that the validation set is an important factor on the performance, and a larger validation set would improve model generalization to some extent. *Secondly*, our model could still outperform all the baselines across four datasets with a smaller validation set, despite having more parameters. By employing appropriate regularizations on the parameters as well as the transformation vectors in Eq. (11) of the main paper, LGNNs are robust with both large and small validation sets.

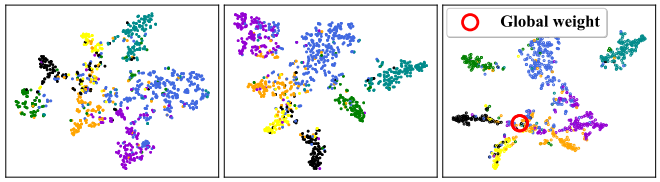
E Complexity Study

Our node-wise localization increases the computational cost. Taking GCN as an example, with everything else being the same, we compare the complexity of neighborhood aggregation for one node in GCN and LGCN. Given a node with degree d, the complexity of GCN is $O(d)$ for neighborhood aggregation. On the other hand, LGCN has two main parts. (1) Node level: generate the localized weight matrix in Eqs. (4), (5) and (6), with complexity $O(d)$; then to calculate $\mathbf{W}_v^l \mathbf{h}_u^{l-1}$ in Eq. (8) with complexity $O(d)$. (2) Edge level: calculate the transformation for all edges with complexity $O(d)$ in Eqs. (7), (9) and (8). Overall, the complexity of LGCN for neighborhood aggregation of one node is $O(d)$, belonging to the same



(a) Cora (b) Citeseer (c) Amazon (d) Chameleon

Figure I: Effect of regularization on the magnitude of localization.



(a) Nodes of GCN (b) Nodes of LGCN (c) Weights of LGCN

Figure II: Visualizations on the Cora dataset. Each color represents a class. (a) Node representations generated by GCN; (b) Node representations generated by LGCN, our proposed localization of GCN; (c) Global (large red circle) and node-wise localized weights (small circles) of LGCN.

complexity class as its base model GCN and only differing by a constant factor. This comparison is also appropriate for other GNNs and LGNNs. In Table II, we report the number of floating point operations (FLOPs) and wall-clock time costs on the smaller Cora and larger Amazon datasets. The FLOPs indeed differ by a constant factor, while the wall-clock time difference is even smaller due to parallelization.

F Effect of Regularization

In Eq. (11), λ regularizes the magnitude of localization in LGNN. We test the impact of λ on the performance of LGCN in Fig. I, w.r.t. GCN. Note the GCN shows a straight line as it has no such regularization. On one hand, LGCN is less sensitive for $\lambda \leq 1$. Some decrease in accuracy is observed on small λ's, which means the localization should not diverge from the global model too drastically. Nevertheless, setting λ around 1 is robust across datasets. On the other hand, when λ becomes larger, the accuracy decreases more significantly to become similar to that of GCN. This is an expected behavior as larger λ's discourage localization and favor a global model in line with our analysis in connections to existing GNNs.

G Visualization of Nodes and Weights

In this case study, we visualize the node representations learned by GCN and LGCN on the Cora dataset using the t-SNE algorithm, as shown in Figs. II(a) and (b), respectively. We observe that the margins between different classes in LGCN are larger than the margins in GCN, and the nodes of each class form more compact clusters in LGCN. The findings imply that our proposed approach learns more powerful representations.

Next, to demonstrate that LGCN generates localized weights for different nodes, we visualize the weights in the last layer ℓ. Specifically, we concatenate the rows of a weight matrix into a single vector, and visualize it using t-SNE. In Fig. II(c), we plot the global weight matrix $\mathbf{W}^\ell$ (indicated by the large red circle), as well as the localized weight matrices $\mathbf{W}_v^\ell$ for all nodes $v \in V$ (indicated by the small circles). It is not surprising that the global model resides around the center of mass of all localized models, as it tries to simultaneously optimize for different local contexts across the graph. On the other hand, the localized models appear in clear clusters that correspond to the node classes, which naturally suggest that nodes of the same class have similar local contexts.