Node-wise Localization of Graph Neural Networks

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

Abstract

Graph neural networks (GNNs) emerge as a powerful family of representation learning models on graphs. To derive node representations, they utilize a global model that recursively aggregates information from the neighboring nodes. However, different nodes reside at different parts of the graph in different local contexts, making their distributions vary across the graph. Ideally, how a node receives its neighborhood information should be a function of its local context, to diverge from the global GNN model shared by all nodes. To utilize node locality without overfitting, we propose a node-wise localization of GNNs by accounting for both global and local aspects of the graph. Globally, all nodes on the graph depend on an underlying global GNN to encode the general patterns across the graph; locally, each node is localized into a unique model as a function of the global model and its local context. Finally, we conduct extensive experiments on four benchmark graphs, and consistently obtain promising performance surpassing the state-of-the-art GNNs.

1 Introduction

Graphs are powerful data structures to model various entities (*i.e.*, nodes) and their interactions (*i.e.*, edges) simultaneously. To learn the representations of nodes on a graph, graph neural networks (GNNs) [Wu *et al.*, 2020] have been proposed as a promising solution. Generally, state-of-the-art GNNs [Kipf and Welling, 2017; Hamilton *et al.*, 2017; Veličković *et al.*, 2018] learn the representation of each node by recursively transferring and aggregating information from its receptive field, which is often defined as its set of neighboring nodes. Consider a toy citation graph in Fig. 1(a), consisting of papers in three areas: biology (*bio*), bioinformatics (*bioinf*) and computer science (*cs*). To derive the representation of a node, say v_1, GNNs aggregate keyword features from not only v_1 itself, but also its neighbors, namely v_2, v_4, v_5 and v_6, as illustrated in Fig. 1(b). Such neighborhood aggregation can be performed recursively for the neighbors as well in more layers, to fully exploit the graph structures. To extract useful representations from the neighbors, GNNs aim to learn the model parameters formulated as a sequence of weight matrices $\mathbf{W}^1, \mathbf{W}^2, \ldots, \mathbf{W}^l$ in each layer.

However, the implicit assumption of a global weight matrix (in each layer) for the entire graph is often too strict. Nodes do not distribute uniformly over the graph, and are associated with different local contexts. For instance, in Fig. 1(c), node v_1 is associated with a *bio* context, v_2 with a *bioinf* context and v_3 with a *cs* context. Different local contexts are characterized by different keyword features, such as "gene" and "cell" in *bio*, "gene" and "svm" in *bioinf*, as well as "svm" and "vc-dim" in *cs*. Thus, a graph-level global weight matrix is inherently inadequate to express the varying importance of features at different localities. More specifically, a global weight matrix can be overly diffuse, for different nodes often have distinct optimal weight matrices, which tend to pull the model in many opposing directions. This may result in a biased model that centers its mass around the most frequent patterns while leaving others not well covered. A natural question follows: *Can we allow each node to be parameterized by its own weight matrix?* Unfortunately, this is likely to cause severe overfitting to local noises and suffer from significantly higher training cost.

In this work, to adapt to the local context of each node without overfitting, we localize the model for each node from a shared global model. As illustrated in Fig. 1(c), a localized weight matrix $\mathbf{W}^l_{v_i}$ for each node v_i can be derived from both the global weight $\mathbf{W}^l$ and the local context of v_i. That is, $\mathbf{W}^l_{v_i}$ is a function of the global information and the local context. Globally, all nodes depend on a common underlying model to encode the general patterns at the graph level. Locally, each node leverages its local context to personalize the common model into its unique localized weight at the node level. It is also useful to incorporate a finer-grained localization at the edge level, where information received through each edge of the target node can be further adjusted. The proposed localization, which we call *node-wise localized GNN* (LGNN), aims to strike a balance between the global and local aspects of the graph. Moreover, LGNN is agnostic of the underlying global model, meaning that it is able to localize any GNN that follows the paradigm of recursive neighborhood aggregation, and subsumes various conventional GNNs as its limiting cases.

In summary, our main contributions are three-fold. (1) We identify the need to adapt to the local context of each node in

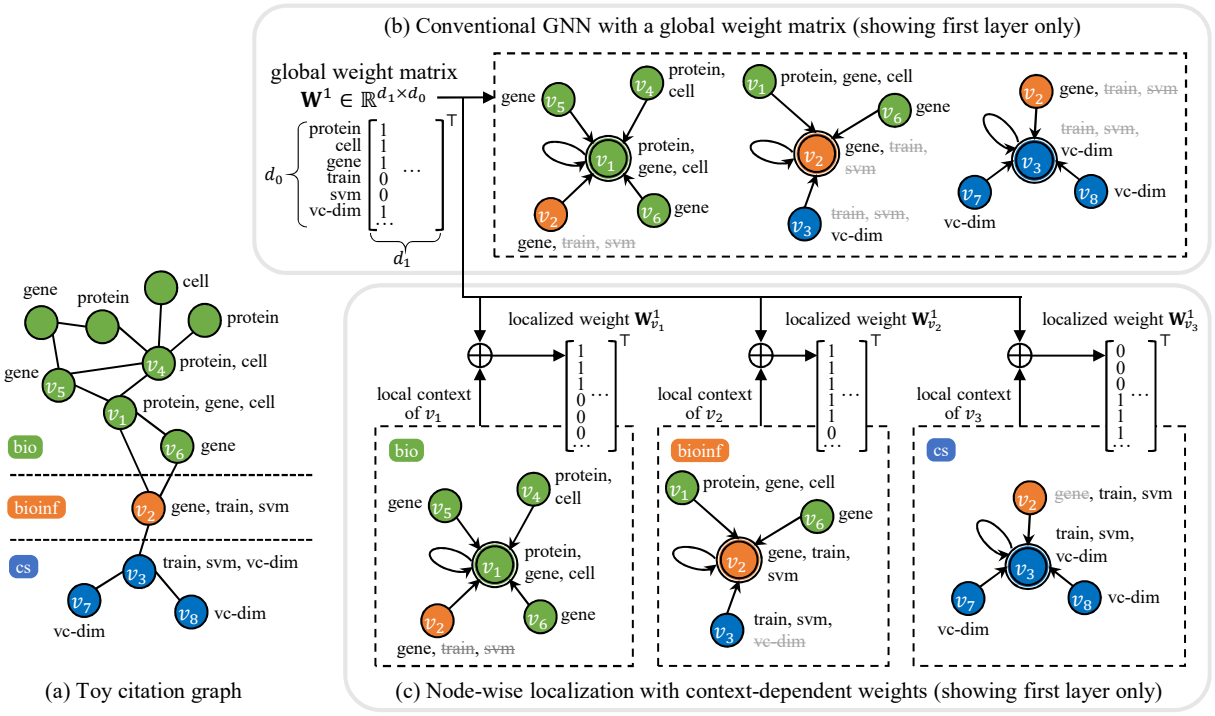

Figure 1: Comparison of conventional GNNs and the proposed localization (best viewed in color).

GNNs. (2) We propose a node-wise localization of GNNs to capture both global and local information on the graph, and further discuss its connection to other works. (3) We conduct extensive experiments on four benchmark datasets and show that LGNN consistently outperforms prior art.

2 Proposed Approach

In this section, we introduce the proposed approach LGNN, and discuss its connection to other works.

2.1 General Formulation of Localization

We start with an abstract definition of localizing a global model to fit local contexts. Specifically, the localized model of an instance v is a function of both the global model and the local context. Let Θ denote the global model, and C_v the local context of v. Then the localized model for v is

$$\Theta_v = f(\Theta, C_v), \tag{1}$$

where f can be any function such as a neural network. While we focus on graph data where v is a node, the general formulation is also pertinent to other kinds of data when there are varying, non-uniform contexts associated with instances. In particular, On a graph $G = (V, E)$ with a set of nodes V and edges E, the local context of a node v can be materialized as the neighbors of v, in addition to v itself. That is,

$$C_v = \{v\} \cup \{u \in V : \langle v, u \rangle \in E\}. \tag{2}$$

2.2 Localization of GNNs

A typical GNN consists of multiple layers of recursive neighborhood aggregation. In the l-th layer, each node v receives and aggregates information from its neighbors to derive its hidden representation $\mathbf{h}_v^l \in \mathbb{R}^{d_l}$, with d_l being the dimension of the l-the layer representation. Note that the initial representation $\mathbf{h}_v^0$ is simply the input feature vector of v. The aggregation is performed on the local context of v which also covers the neighbors of v, as follows.

$$\mathbf{h}_v^l = \sigma\left(\textsc{Aggr}\left(\left\{\mathbf{W}^l \mathbf{h}_u^{l-1} : \forall u \in C_v\right\}\right)\right), \tag{3}$$

where $\mathbf{W}^l \in \mathbb{R}^{d_l \times d_{l-1}}$ is a weight matrix in the l-th layer, $\sigma(\cdot)$ is an activation function, and $\textsc{Aggr}(\cdot)$ is an aggregation function. Different GNNs differ in the choice of the aggregation function. For instance, GCN [Kipf and Welling, 2017] uses an aggregator roughly equivalent to mean pooling [Hamilton *et al.*, 2017], GAT [Veličković *et al.*, 2018] uses an attention-weighted mean aggregator, and GIN [Xu *et al.*, 2019] uses a multi-layer perceptron (MLP).

Node-level localization. In the above setup, we have a global model $\mathbf{W}^l$ for all nodes in the l-th layer. At the node level, the global model is localized to adapt to the local context of each node. Specifically, we modulate the global weight $\mathbf{W}^l$ by a node-specific transformation through scaling and shifting. The localized weight matrix of node v is given by

$$\mathbf{W}_v^l = \mathbf{W}^l \odot \left[\left(\mathbf{a}_v^l\right)_{\times d_l}\right]^\top + \left[\left(\mathbf{b}_v^l\right)_{\times d_l}\right]^\top, \tag{4}$$

where $\mathbf{a}_v^l, \mathbf{b}_v^l \in \mathbb{R}^{d_{l-1}}$ are v-specific vectors for scaling and shifting, respectively. Here the notation $[(\mathbf{x})_{\times n}]$ represents a matrix of n columns all of which are identical to the vector $\mathbf{x}$, and $\odot$ denotes element-wise multiplication. We essentially

transform each row of the global matrix $\mathbf{W}^l$ by $\mathbf{a}_v^l$ and $\mathbf{b}_v^l$ in an element-wise manner, to generate the localized weight $\mathbf{W}_v^l$, so that various importance levels are assigned to each feature dimension of the node embedding $\mathbf{h}_u^{l-1}$.

It is important to recognize that the node-specific transformation should not be directly learned, for two reasons. First, directly learning them significantly increases the number of model parameters especially in the presence of a large number of nodes, causing overfitting. Second, node-wise localization is a function of both the global model and the local context as formulated in Eq. (1), in order to better capture the local information surrounding each node. Thus, for node v, we propose to generate the v-specific transformation in the l-th layer from its local contextual information $\mathbf{c}_v^l$. A straightforward recipe for $\mathbf{c}_v^l$ is to pool the $(l-1)$-th layer representations of the nodes in the local context C_v:

$$\mathbf{c}_v^l = \textsc{Mean}\left(\left\{\mathbf{h}_u^{l-1} : \forall u \in C_v\right\}\right), \tag{5}$$

in which we adopt the mean pooling although other forms of pooling can be substituted for it. Given the local contextual information $\mathbf{c}_v^l \in \mathbb{R}^{d_{l-1}}$, we further utilize a dense layer, shared by all nodes, to generate the v-specific transformation as follows.

$$\mathbf{a}_v^l = \sigma\left(\mathbf{M}_a^l \mathbf{c}_v^l\right) + \mathbf{1}, \quad \mathbf{b}_v^l = \sigma\left(\mathbf{M}_b^l \mathbf{c}_v^l\right), \tag{6}$$

where $\mathbf{M}_a^l, \mathbf{M}_b^l \in \mathbb{R}^{d_{l-1} \times d_{l-1}}$ are learnable parameters shared by all nodes, and $\mathbf{1}$ is a vector of ones to ensure that the scaling factors are centered around one. Note that if the dimension d_{l-1} is too large, such as that of the high-dimensional raw features handled in the first layer of a GNN, we could employ two dense layers and set the first one with a smaller number of neurons.

Edge-level localization. The node-level localization of the global weight matrix is coarse-grained, as the same localized weight is applied on all neighbors of the target node. That is, the target node receives information through each of its edges uniformly. To enable a finer-grained localization, we further modulate how information propagates at the edge level.

Consider a node v with edges $\{\langle u, v\rangle : \forall u \in C_v\}$. In the l-th layer of the GNN, let $\mathbf{c}_{u,v}^l$ denote the local contextual information of the edge $\langle u, v\rangle$, which is given by its two ends. Specifically, we concatenate the $(l-1)$-th layer representations of nodes u and v, as follows.

$$\mathbf{c}_{u,v}^l = \textsc{Concat}\left(\mathbf{h}_v^{l-1}, \mathbf{h}_u^{l-1}\right). \tag{7}$$

To implement the edge-level localization, we similarly adopt an edge-specific transformation through scaling and shifting. Specifically, to derive the l-th layer representation of v, we scale and shift the information from each edge $\langle u, v\rangle$ during aggregation, by rewriting Eq. (3) as follows.

$$\mathbf{h}_v^l = \sigma\left(\textsc{Aggr}\left(\left\{\mathbf{W}_v^l \mathbf{h}_u^{l-1} \odot \mathbf{a}_{u,v}^l + \mathbf{b}_{u,v}^l : \forall u \in C_v\right\}\right)\right), \tag{8}$$

where $\mathbf{a}_{u,v}^l, \mathbf{b}_{u,v}^l \in \mathbb{R}^{d_l}$ are edge $\langle u, v\rangle$-specific vectors for scaling and shifting, respectively. Like the node-specific transformation, we generate the edge-specific transformation with a dense layer given by

$$\mathbf{a}_{u,v}^l = \sigma\left(\mathbf{N}_a^l \mathbf{c}_{u,v}^l\right) + \mathbf{1}, \quad \mathbf{b}_{u,v}^l = \sigma\left(\mathbf{N}_b^l \mathbf{c}_{u,v}^l\right), \tag{9}$$

where $\mathbf{N}_a^l, \mathbf{N}_b^l \in \mathbb{R}^{d_l \times 2d_{l-1}}$ are learnable parameters shared by all edges.

2.3 Semi-supervised Node Classification

In the benchmark task of semi-supervised node classification, each node belongs to one of the pre-defined classes $\{1, 2, \ldots, K\}$. However, we only know the class labels of a subset of nodes $V_Y \subset V$, called the labeled nodes. The goal is to predict the class labels of the remaining nodes. Following standard practice [Kipf and Welling, 2017], for K-way classification we set the dimension of the output layer to K, and apply a softmax function to the representation of each node. That is, given a total of ℓ layers,

$$\mathbf{z}_{v,k} = \textsc{Softmax}\left(\mathbf{h}_{v,k}^\ell\right) = \frac{\exp\left(\mathbf{h}_{v,k}^\ell\right)}{\sum_{k'=1}^K \exp\left(\mathbf{h}_{v,k'}^\ell\right)}. \tag{10}$$

For a labeled node $v \in V_Y$, let $Y_{v,k} \in \{0, 1\}$ be 1 if and only if node v belongs to class k. The overall loss is then formulated using a cross-entropy loss with regularization as

$$\begin{aligned} &-\textstyle\sum_{v \in V_Y} \sum_{k=1}^K Y_{v,k} \ln \mathbf{z}_{v,k} + \lambda_G \|\Theta_G\|_2^2 + \lambda_L \|\Theta_L\|_2^2 \\ &+\lambda\left(\|A - 1\|_2^2/|A| + \|B\|_2^2/|B|\right), \end{aligned} \tag{11}$$

where $\Theta_G = \{\mathbf{W}^l : l \le \ell\}$ contains the parameters of the global model, $\Theta_L = \{\mathbf{M}_a^l, \mathbf{M}_b^l, \mathbf{N}_a^l, \mathbf{N}_b^l : l \le \ell\}$ contains the parameters of localization, and A, B are sets respectively containing the transformation vectors for scaling ($\mathbf{a}_v^l$'s and $\mathbf{a}_{u,v}^l$'s) and shifting ($\mathbf{b}_v^l$'s and $\mathbf{b}_{u,v}^l$'s). Note that the norms $\|\cdot\|_2^2$ are computed over all tensor elements in the sets. While A and B are not learnable, they are functions of the learnable Θ_L. We explicitly constrain them to favor small local changes, *i.e.*, close-to-one scaling and close-to-zero shifting. Moreover, as A and B grow with the size of the graph, their norms are further normalized by the total number of tensor elements in them, denoted by $|A|$ and $|B|$, respectively. λ_G, λ_L and λ are non-negative hyperparameters.

2.4 Connections to Other Works

Next, we discuss how LGNN is related to other lines of work in GNNs, network embedding and hypernetworks.

Relationship with existing GNNs. The proposed LGNN is able to localize any GNN model that follows the scheme of recursive neighborhood aggregation in Eq. (3). In particular, LGNN is a generalized GNN that subsumes, as special limiting cases, several conventional GNNs by adopting an appropriate aggregation function and regularization.

Specifically, as $\lambda \to \infty$ in Eq. (11), the scaling and shifting would approach 1's and 0's in the limiting case, respectively. This is equivalent to removing all node-wise localization at both node and edge levels, where all nodes assume a global model. Thus, LGNN would be asymptotically equivalent to GCN [Kipf and Welling, 2017] and GIN [Xu *et al.*, 2019], when using a graph convolution aggregator (roughly equivalent to the mean pooling) and an MLP aggregator on the sum-pooled representations, respectively. Furthermore, with an appropriate regularization, LGNN can be asymptotically reduced to GAT[1] [Veličković *et al.*, 2018]. Consider the vectors for scaling in A, which can be split into the node-specific A_V (containing $\mathbf{a}_v^l$'s) and edge-specific A_E (containing $\mathbf{a}_{u,v}^l$'s).

[1]Here we only discuss GAT with one attention head.

For A_V, we maintain the same regularization $\|A_V - 1\|_2^2$; for A_E, we adopt the regularization $\|A_E - \alpha\|_2^2$ such that α contains the corresponding attention coefficients on all edges. That is,

$$\mathbf{a}_{u,v}^l \to \left[\left(\alpha_{u,v}^l\right)_{\times d_l}\right]^\top, \tag{12}$$

where $\alpha_{u,v}^l \in \mathbb{R}$ is the attention coefficient on edge $\langle u, v\rangle$. Coupled with a mean aggregator, we obtain GAT as the limiting case when $\lambda \to \infty$. Alternatively, we can also maintain the same regularization for both node- and edge-specific transformations, and adopt a weighted mean aggregator in accordance to the attention coefficients. Note that in GAT the attention coefficient on each edge is a scalar, whereas in our edge-level localization, the edge-specific scaling is represented by a vector, which is able to flexibly vary the contribution from different dimensions of the information received through the edges.

Relationship with network embedding. Network embedding [Cai *et al.*, 2018] is a popular alternative to GNNs to learn node representations on a graph. In these methods, each node is directly encoded with a learnable parameter vector, which is taken as the representation of the node. In contrast, in GNNs, the representations of the nodes are derived from a shared learnable model. Thus, network embedding can be viewed as a set of individual local models that are loosely coupled by local graph structures, whereas GNNs capture a more global view of the structures. In contrast, the proposed LGNN achieves a balance between the local and global views, to allow localized variations grounded on a global model.

Relationship with hypernetworks. Our localization strategy can be deemed a form of hypernetworks [Ha *et al.*, 2017], which use a secondary neural network to generate weights for the target network. In our case, we employ dense layers as a secondary network to generate the vectors for scaling and shifting, which are leveraged to ultimately generate the localized weights for the target GNN. Moreover, our approach boils down to the feature-wise modulation of information received from the conditioning local context (*i.e.*, neighboring nodes) at both node and edge levels, which is inspired by the feature-wise modulation of neural activations in FiLMs [Perez *et al.*, 2018; Birnbaum *et al.*, 2019].

On graphs, GNN-FiLM [Brockschmidt, 2020] also uses a form of FiLM on GNNs. However, there are several fundamental differences between GNN-FiLM and our LGNN. First, in terms of *problem and motivation*, GNN-FiLM aims to model edge labels on relational networks for message passing. In contrast, our LGNN aims to model local contexts for node-wise localization, and works on general graphs. Second, in terms of *model*, GNN-FiLM models relation-specific transformations conditioned only on a node's self-information, which does not sufficiently reflect the full local context of the node as LGNN is conditioned on. Furthermore, when there is no edge labels, GNN-FiLM reduces to a "uniform edge" model. In LGNN, we still have both node and edge level modulation to achieve localization. Third, in terms of *empirical performance*, as discussed therein [Brockschmidt, 2020], GNN-FiLM only achieves at best comparable performance to existing GNNs on citation networks (such as *Cora* and *Citeseer*) where there is no edge label. Our own experiments also have reproduced similar results in the next section.

Dataset	# Nodes	# Edges	# Classes	# Features
Cora	2,708	5,429	7	1,433
Citeseer	3,327	4,732	6	3,703
Amazon	13,381	245,778	10	767
Chameleon	2,277	36,101	5	2,325

Table 1: Summary of datasets.

3 Experiments

In this section, we evaluate and analyze the empirical performance of our proposed approach LGNN.

3.1 Experimental Setup[2]

Datasets. We utilize four benchmark datasets. They include two academic citation networks, namely *Cora* and *Citeseer* [Yang *et al.*, 2016], in which nodes correspond to papers and edges correspond to citations between papers. The input node features are bag-of-word vectors, indicating the presence of each keyword. A similar citation network for Wikipedia articles called *Chameleon* [Pei *et al.*, 2020] is also used. Finally, we use an e-commerce co-purchasing network called *Amazon* [Hou *et al.*, 2020], in which the nodes correspond to computer products and the edges correspond to co-purchasing relationships between products. The input node feature vectors are constructed from product images. The statistics of the datasets are summarized in Table 1.

Baselines and our approach. We compare against three categories of baselines. (1) Embedding models: *DeepWalk* [Perozzi *et al.*, 2014] and *Planetoid* [Yang *et al.*, 2016]. Both employ a direct embedding lookup and adopt random walks to sample node pairs. However, DeepWalk is unsupervised such that node classification is performed as a downstream task on the learned representations, whereas Planetoid is an end-to-end model that jointly learns the representations and the classifier. (2) GNN models: *GCN* [Kipf and Welling, 2017], *GAT* [Veličković *et al.*, 2018] and *GIN* [Xu *et al.*, 2019]. (3) GNN-FiLM: The original GNN-FiLM works with a GCN-style model, which we call *GCN-FiLM*. We further extend it to two other GNN architectures GAT and GIN, resulting in two more versions *GAT-FiLM* and *GIN-FiLM*.

On the other hand, our approach LGNN can also be implemented on different GNN architectures. Specifically, we employ GCN, GAT and GIN as the global model, and obtain localized versions *LGCN*, *LGAT* and *LGIN*, respectively.

Main settings. For DeepWalk, we sample 10 random walks per node with walk length 100 and windows size 5, and set the embedding dimension to 128. We further train a logistic regression as the downstream classifier. For Planetoid, we

[2] Additional implementation details and experimental settings are included in Sections A and B of the supplemental material.

Methods	# Params (Cora)	Cora		Citeseer		Amazon		Chameleon	
		Accuracy	Micro-F	Accuracy	Micro-F	Accuracy	Micro-F	Accuracy	Micro-F
DeepWalk	693K	73.8±0.3	74.9±0.1	61.6±0.2	60.5±1.0	80.1±1.6	77.3±1.3	41.2±1.3	40.1±1.1
Planetoid	345K	66.1±0.4	64.5±0.5	64.5±0.3	62.9±0.4	69.8±1.7	64.5±1.5	39.3±1.8	37.7±1.7
GCN	11K	81.5±0.7	80.8±0.5	70.4±0.5	68.3±0.7	81.9±0.5	81.0±0.8	46.7±4.3	46.4±2.4
GCN-64	92K	82.0±0.3	80.9±0.3	71.1±0.3	69.2±0.4	82.1±0.5	81.2±0.8	48.3±3.3	46.3±1.8
GCN-96	138K	81.9±0.2	80.8±0.3	71.3±0.4	69.4±0.5	82.2±0.4	81.5±0.7	45.5±2.4	43.8±2.5
GCN-FiLM	35K	78.1±0.6	76.9±0.5	69.8±1.1	67.9±1.0	79.2±1.0	77.1±1.5	42.8±1.1	39.9±1.3
LGCN	104K	**83.5**±0.3	**82.1**±0.4	**72.2**±0.4	**70.2**±0.4	**83.7**±1.5	**82.3**±2.0	**50.9**±1.1	**49.7**±0.7
(improv.)	-	(1.8%)	(1.5%)	(1.3%)	(1.2%)	(1.8%)	(1.0%)	(5.4%)	(7.1%)
GAT	92K	82.9±0.6	82.0±0.6	72.4±0.7	70.4±0.8	82.4±1.3	80.1±1.9	47.2±1.1	46.2±2.1
GAT-64	738K	83.1±0.4	81.9±0.6	71.6±1.5	69.8±1.6	83.0±0.9	81.2±1.4	51.2±1.5	50.2±1.3
GAT-96	1108K	83.2±0.6	81.9±0.6	71.4±0.9	69.6±0.9	83.1±1.0	81.5±1.4	51.9±1.2	50.2±1.8
GAT-FiLM	277K	82.0±0.5	80.6±0.6	71.2±1.0	69.2±1.1	83.3±0.6	81.9±0.8	46.8±5.7	45.1±5.2
LGAT	836K	**83.6**±0.4	**82.3**±0.4	**72.8**±0.4	**70.8**±0.5	**83.7**±0.7	**82.3**±0.8	**52.6**±1.0	**51.1**±0.9
(improv.)	-	(0.5%)	(0.4%)	(0.6%)	(0.6%)	(0.5%)	(0.5%)	(1.3%)	(1.8%)
GIN	11K	80.2±0.5	78.8±0.3	68.5±0.7	66.5±1.0	79.6±1.7	78.5±2.6	45.8±3.0	41.2±4.0
GIN-64	92K	80.3±1.1	79.1±1.0	67.8±1.5	66.1±1.1	79.8±1.1	79.0±1.4	45.7±4.5	40.7±5.7
GIN-96	138K	79.9±1.1	78.9±1.0	68.6±1.4	66.6±1.6	80.2±2.1	79.0±3.2	45.9±3.5	41.5±4.1
GIN-FiLM	35K	79.8±0.7	78.5±0.5	67.7±1.4	65.8±1.5	78.6±2.8	77.2±3.3	38.8±2.6	34.2±2.9
LGIN	126K	**82.6**±0.8	**81.6**±0.8	**71.3**±0.4	**69.5**±0.5	**84.0**±1.2	**82.7**±1.7	**48.3**±1.9	**47.3**±1.9
(improv.)	-	(2.9%)	(3.2%)	(3.9%)	(4.4%)	(4.7%)	(4.7%)	(5.2%)	(14.0%)

Table 2: Average classification performance with standard deviation (percent) over 10 runs. Improvements of LGNN are relative to the best performing baseline with the corresponding GNN architecture.

set the path length to 10, window size to 3 and the embedding dimension to 50. We select the best results from their transductive and inductive versions for report. For all GNN-based approaches, we adopt two aggregation layers, noting that deeper layers often bring in noises from distant nodes [Pei *et al.*, 2020] and cause "over-smoothing" such that all nodes obtain very similar representations [Li *et al.*, 2018; Xu *et al.*, 2018]. The dimension of the hidden layer defaults to 8, while we also present results using larger dimensions. The regularization of GNN parameters is set to $\lambda_G = 0.0005$. These settings are chosen via empirical validation, and are largely consistent with the literature [Perozzi *et al.*, 2014; Kipf and Welling, 2017; Veličković *et al.*, 2018]. For our models, the additional regularizations are set to $\lambda_L = \lambda = 1$ (except for $\lambda = 0.1$ in LGAT).

Training and testing. For all datasets, we follow the standard split in the literature [Yang *et al.*, 2016; Kipf and Welling, 2017; Veličković *et al.*, 2018], which uses 20 labeled nodes *per class* for training, 500 nodes for validation and 1000 nodes for testing. We repeat 10 runs for each experiment, and report the average accuracy and micro-F score.

3.2 Performance Comparison

Table 2 shows the comparison of the classification performance on the four datasets. For each GNN architecture, we also include additional models by increasing the dimension of its hidden layer. For instance, GCN-64 denotes that the hidden layer of GCN has 64 neurons. The results reveal a few important observations.

Firstly, we observe that LGNN consistently achieves significant performance boosts w.r.t. three state-of-the-art GNNs on four datasets. This suggests that, our localization approach is agnostic of the particular GNN architecture of the global model, and can be used universally on different GNNs. Furthermore, LGNNs also outperform GNN-FiLMs, as GNN-FiLMs only achieve at best comparable performance with the base GNNs. A potential reason is that GNN-FiLM is only conditioned on the target node's self-information to modulate message passing for different edge labels, which may not be useful on graphs without edge labels. In contrast, LGNN is conditioned on the full local context of the target node to achieve node-wise localization that could be useful to the classification of nodes residing across different localities on the graph. *Secondly*, GAT-based models generally attain better performance than GCN- and GIN-based models, and the performance gain in LGAT w.r.t. GAT is smaller. This is not surprising as the attention coefficients on the edges can be understood as a limited form of localization to differentiate the weight of different neighbors. *Thirdly*, the results imply that increasing the number of parameters alone cannot achieve the effect of localization. It may be argued that more parameters lead to higher model capacity, which can potentially assign some space to encode the local aspects. However, when the parameters are still shared globally, there is no explicit node-wise constraint and thus the model will still try to find a middle ground. Nevertheless, for a fair comparison, we use more parameters on each GNN baseline to match or exceed the number of parameters on the corresponding LGNN, as listed in Table 2[3]. As expected, more parameters only result in marginal improvements to the baselines. Thus, the efficacy of LGNN is derived from our localization strategy rather

[3]Details of the calculation are included in Section C of the supplemental material.

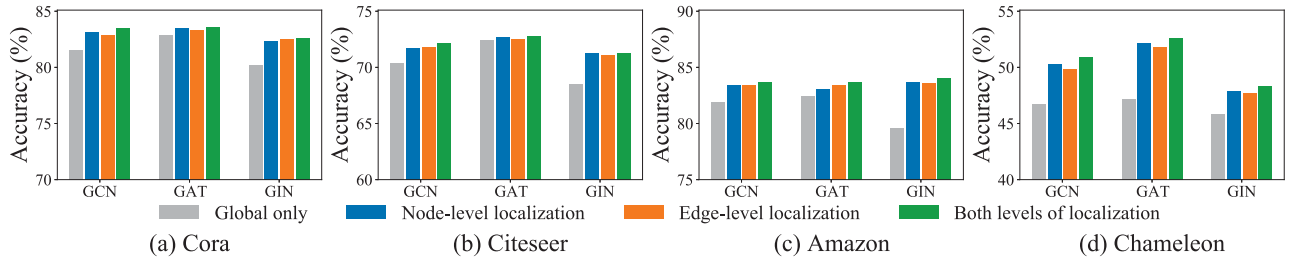

Figure 2: Ablation study on the effect of each localization module.

than just more parameters. *Fourthly*, LGNN is robust and stable given their relatively small standard deviations in many cases. We hypothesize that, in conventional GNNs, a bad initialization would risk a suboptimal global model shared by all nodes without recourse. In contrast, node-wise localization offers an opportunity to adjust the suboptimal model for some nodes, and is thus less susceptible to the initialization.

It has also been discussed that in GNNs overfitting to the validation set is an issue [Shchur *et al.*, 2018], and thus models with more parameters tend to perform better given a larger validation set. Therefore, we further compare different methods using a smaller validation set of only 100 nodes[4]. Our proposed LGNN still outperforms all the baselines across the four datasets, demonstrating its power and stability.

3.3 Further Model Analyses

Due to space constraint, we only present an ablation study here[5]. In the ablation study, we investigate the effectiveness of the node- and edge-level localization modules in LGNN. To validate the contribution of each module, we compare the accuracy of four variants in Fig. 2: (1) global only without any localization; (2) node-level localization only; (3) edge-level localization only; and (4) the full model with both localization modules. We observe that utilizing only one module, whether at the node or edge level, consistently outperform the global model. Between the two modules, the node-level localization tends to perform better. Nevertheless, modeling both jointly results in the best performance, which implies that both modules are effective and possess complementary localization power.

4 Related Work

Graph representation learning has received significant attention in recent years. Earlier network embedding approaches [Perozzi *et al.*, 2014; Tang *et al.*, 2015; Grover and Leskovec, 2016] employ a direct embedding lookup for node representations, which are often coupled via local structures such as skip-grams. To better capture global graph structures, GNNs [Kipf and Welling, 2017; Hamilton *et al.*, 2017; Veličković *et al.*, 2018; Xu *et al.*, 2019] open up promising opportunities. They generally follow a paradigm of recursive neighborhood aggregation, in which each node receives information from its neighbors on the graph. More complex structural information is also exploited by recent variants [Zhang *et al.*, 2020; Pei *et al.*, 2020; Chen *et al.*, 2020], but none of them deals with the node-wise localization.

While our approach can be deemed a form of hypernetworks [Ha *et al.*, 2017; Perez *et al.*, 2018] as discussed in Sect. 2.4, several different forms of local models have been explored in related problems. In manifold learning [Yu *et al.*, 2009; Ladicky and Torr, 2011], local codings are used to approximate any point on the manifold as a linear combination of its surrounding anchor points. In low-rank matrix approximation [Lee *et al.*, 2013], multiple low-rank approximations are constructed for different regions of the matrix before aggregation. These methods are tightly coupled with their problems, and cannot be easily extended to localize GNNs. Some meta-learning frameworks [Vinyals *et al.*, 2016; Snell *et al.*, 2017; Finn *et al.*, 2017] can also be viewed as adapting to local contexts. Given a set of training tasks, meta-learning aims to learn a prior that can be adapted to new unseen tasks, often for addressing few-shot learning problems. In particular, existing meta-learning models on graphs are mostly designed for few-shot node classification [Zhou *et al.*, 2019; Yao *et al.*, 2020; Liu *et al.*, 2021] or regression [Liu *et al.*, 2020]. Such few-shot settings are fundamentally different from our localization objective, in which all nodes are seen during training and we leverage the local contexts of the nodes in order to enhance their representations.

5 Conclusions

In this work, we identified the need to localize GNNs for different nodes that reside in non-uniform local contexts across the graph. This motivated us to propose a node-wise localization approach, named LGNN, in order to adapt to the locality of each node. On one hand, we encode graph-level general patterns using a global weight matrix. On the other hand, we modulate the global model to generate localized weights specific to each node, and further perform an edge-level modulation to enable finer-grained localization. Thus, the proposed LGNN can capture both local and global aspects of the graph well. Finally, our extensive experiments demonstrate that LGNN significantly outperforms state-of-the-art GNNs.

Acknowledgments

This research is supported by the Agency for Science, Technology and Research (A*STAR) under its AME Programmatic Funds (Grant No. A20H6b0151).

[4]These results are in Section D of the supplemental material.

[5]More analyses on the complexity and effect of regularization, as well as model visualization, can be found in Sections E, F and G of the supplemental material.

References

[Birnbaum *et al.*, 2019] Sawyer Birnbaum, Volodymyr Kuleshov, Zayd Enam, Pang Wei W Koh, and Stefano Ermon. Temporal FiLM: Capturing long-range sequence dependencies with feature-wise modulations. In *NeurIPS*, pages 10287–10298, 2019.

[Brockschmidt, 2020] Marc Brockschmidt. GNN-FiLM: Graph neural networks with feature-wise linear modulation. In *ICML*, pages 1144–1152, 2020.

[Cai *et al.*, 2018] Hongyun Cai, Vincent W Zheng, and Kevin Chang. A comprehensive survey of graph embedding: problems, techniques and applications. *TKDE*, 30(9):1616–1637, 2018.

[Chen *et al.*, 2020] Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. Simple and deep graph convolutional networks. In *ICML*, pages 1725–1735, 2020.

[Finn *et al.*, 2017] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *ICML*, pages 1126–1135, 2017.

[Grover and Leskovec, 2016] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *KDD*, pages 855–864, 2016.

[Ha *et al.*, 2017] David Ha, Andrew Dai, and Quoc V Le. Hypernetworks. In *ICLR*, 2017.

[Hamilton *et al.*, 2017] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NeurIPS*, pages 1024–1034, 2017.

[Hou *et al.*, 2020] Yifan Hou, Jian Zhang, James Cheng, Kaili Ma, Richard TB Ma, Hongzhi Chen, and M Yang. Measuring and improving the use of graph information in graph neural networks. In *ICLR*, 2020.

[Kipf and Welling, 2017] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.

[Ladicky and Torr, 2011] Lubor Ladicky and Philip HS Torr. Locally linear support vector machines. In *ICML*, pages 985–992, 2011.

[Lee *et al.*, 2013] Joonseok Lee, Seungyeon Kim, Guy Lebanon, and Yoram Singer. Local low-rank matrix approximation. In *ICML*, pages 82–90, 2013.

[Li *et al.*, 2018] Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *AAAI*, pages 3538–3545, 2018.

[Liu *et al.*, 2020] Zemin Liu, Wentao Zhang, Yuan Fang, Xinming Zhang, and Steven C.H. Hoi. Towards locality-aware meta-learning of tail node embeddings on networks. In *CIKM*, pages 975–984, 2020.

[Liu *et al.*, 2021] Zemin Liu, Yuan Fang, Chenghao Liu, and Steven C.H. Hoi. Relative and absolute location embedding for few-shot node classification on graph. In *AAAI*, 2021.

[Pei *et al.*, 2020] Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. Geom-GCN: Geometric graph convolutional networks. In *ICLR*, 2020.

[Perez *et al.*, 2018] Ethan Perez, Florian Strub, Harm De Vries, Vincent Dumoulin, and Aaron Courville. FiLM: Visual reasoning with a general conditioning layer. In *AAAI*, pages 3942–3951, 2018.

[Perozzi *et al.*, 2014] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. DeepWalk: Online learning of social representations. In *KDD*, pages 701–710, 2014.

[Shchur *et al.*, 2018] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868*, 2018.

[Snell *et al.*, 2017] Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. In *NeurIPS*, pages 4077–4087, 2017.

[Tang *et al.*, 2015] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. LINE: Large-scale information network embedding. In *TheWebConf*, pages 1067–1077, 2015.

[Veličković *et al.*, 2018] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. In *ICLR*, 2018.

[Vinyals *et al.*, 2016] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. In *NeurIPS*, pages 3630–3638, 2016.

[Wu *et al.*, 2020] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *TNNLS*, 32(1):4–24, 2020.

[Xu *et al.*, 2018] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. In *ICML*, pages 5453–5462, 2018.

[Xu *et al.*, 2019] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *ICLR*, 2019.

[Yang *et al.*, 2016] Zhilin Yang, William W Cohen, and Ruslan Salakhutdinov. Revisiting semi-supervised learning with graph embeddings. In *ICML*, pages 40–48, 2016.

[Yao *et al.*, 2020] Huaxiu Yao, Chuxu Zhang, Ying Wei, Meng Jiang, Suhang Wang, Junzhou Huang, Nitesh V Chawla, and Zhenhui Li. Graph few-shot learning via knowledge transfer. In *AAAI*, pages 6656–6663, 2020.

[Yu *et al.*, 2009] Kai Yu, Tong Zhang, and Yihong Gong. Nonlinear learning using local coordinate coding. In *NeurIPS*, pages 2223–2231, 2009.

[Zhang *et al.*, 2020] Kai Zhang, Yaokang Zhu, Jun Wang, and Jie Zhang. Adaptive structural fingerprints for graph attention networks. In *ICLR*, 2020.

[Zhou *et al.*, 2019] Fan Zhou, Chengtai Cao, Kunpeng Zhang, Goce Trajcevski, Ting Zhong, and Ji Geng. Meta-GNN: On few-shot node classification in graph meta-learning. In *CIKM*, pages 137–144, 2019.