Lasagne: A Multi-Layer Graph Convolutional Network Framework via Node-aware Deep Architecture

Xupeng Miao, Wentao Zhang, Yingxia Shao, Bin Cui, *Senior Member, IEEE,* Lei Chen, *Fellow, IEEE,* Ce Zhang, and Jiawei Jiang

Abstract—Graph convolutional networks (GCNs) have been successfully applied in many different real-world tasks. However, most of the existing methods are based on shallow GCN, because multiple layers involve long-distance neighborhood information but lead to the over-smoothing problem. Actually, a similar challenge exists in the depth limitation for primitive convolutional neural networks (CNNs). As the multi-layer architecture can increase the representation ability of GCN, we study and learn from the recent progress in CNN and propose Lasagne, a novel multi-layer GCN framework, empowered by node-aware layer aggregators and factorization-based layer interactions to overcome the over-smoothing problem and realize the full potentials of the GCN model. We analyze how the node locality affects the information propagation in GCN and propose a novel node aggregation mechanism in an adaptive manner. We further demystify Lasagne from a mutual information view and evaluate it on both real-world benchmark data sets and large-scale industrial production data sets. Lasagne shows strong empirical performance on the semi-supervised node classification task and outperforms the state-of-the-art methods without considering the node locality.

Index Terms—Deep learning, Graph Convolutional Neural Network, Over-smoothing, Information Loss, Node locality.

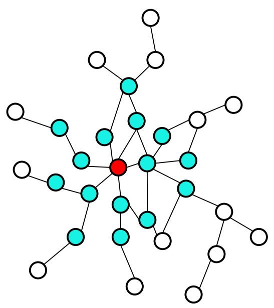
1 Introduction

Graph neural networks (GNNs) are becoming more and more attractive in the graph data management community. They have been successfully applied in graph mining tasks [1], [2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [12], and several GNN training systems have been proposed by the database community to support large scale graph mining, including G3 [13], AGL [14] and AliGraph [15]. Graph convolutional network (GCN), as one of the most representative GNNs, becomes increasingly popular in many graph-based applications, including semi-supervised node classification [16], link prediction [17] and recommendation systems [18]. GCNs generalize convolutional neural networks (CNNs) to graph-structured data by applying the "graph convolution" operation on the neighbors of a node to obtain the node embedding layer by layer. Every node gathers the embeddings of its neighbors at each layer, followed by mean pooling and nonlinearity. By stacking multiple layers, GCN can learn a node representation by utilizing information from distant neighbors. The final layer embedding can be used for subsequent learning tasks, and for example, it is passed to a classifier for node classification problems.

With the rapid growth of data, deep architecture has an advantage over shallow architectures when dealing with complex learning problems on large-scale inputs. The stacking of multiple linear and non-linear processing units in a layer-wise fashion gives deep networks the ability to learn complex representations at different levels of abstraction. For example, from CIFAR [19] to ImageNet [20], CNNs achieve great performance improvements since the innovations on deeper architectures [21], [22]. However, unlike the deep CNNs, the multi-layer architecture of GCN often achieves the best performance with limited depth even on large graphs; A GCN with more layers may perform worse than the one with fewer layers [16]. Thus, in this paper, the first question we want to address is, *can deep GCN architecture bring performance improvement on large-scale graphs*?

The receptive field for a pixel in CNN is usually a small region of the image. While for GCN, the receptive field for a node is its neighbors or a sampled subset of the neighbors [23] on the graph. An L-layer GCN can capture the node information from L-hop neighborhood, so the deep architecture helps to increase the receptive field and access more neighborhood information. Recently, there is a trend for GCN approaches to learn from the architectural innovations of deep CNNs in computer vision problems. For example, ResGCN [16] involves residual connections learned from ResNet [22] but it still does not perform better than the 2-layer GCN on many datasets, e.g., citation datasets. JK-Net [24] utilizes convolutional layer combination learned from GoogleNet to gather more information from previous layers. DenseGCN [25] further ports the dense connection

• *Xupeng Miao, Wentao Zhang, and Bin Cui are with the Key Lab of High Confidence Software Technologies (MOE), Department of Computer Science, Peking University, China. Bin Cui is also with Institute of Computational Social Science, Peking University (Qingdao), and Center for Data Science, Peking University & National Engineering Laboratory for Big Data Analysis and Applications. Yingxia Shao (corresponding author) is with School of Computer Science, Beijing University of Posts and Telecommunications. Lei Chen is with the Department of Computer Science and Engineering, HKUST. Ce Zhang and Jiawei Jiang are with Department of Computer Science, ETH Zurich, Switzerland.*
E-mail: {xupeng.miao, wentao_zhang, bin.cui}@pku.edu.cn, shaoyx @bupt.edu.cn, leichen@cse.ust.hk, {ce.zhang, jiawei.jiang}@inf.ethz.ch


(a) "central" node

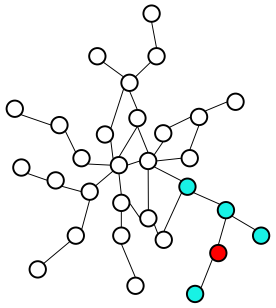

(b) "non-cnetral" node

Fig. 1. Illustration of the nodes (red) with different localities and their 2-hop neighborhoods (blue).

from DenseNet [26] to make GCN deeper. These architecture optimizations help to reduce the vanishing gradients problem and improve the depth of GCN.

However, these approaches neglect that the main defect of deep GCN originates from over-smoothing [27]. Although deep GCN involves more information from "neighboring" nodes, it may lead to more information loss at the same time. The degradation of learning for deep GCN occurs due to the over-smoothing problem, in which the output features may be over-smoothed and nodes from different clusters may become indistinguishable. Due to the neighborhood expansion problem [28], the over-smoothing could be more serious on large graphs. Some recent studies (e.g., GraphSAGE [23], DropEdge [29]) take efforts to alleviate this problem with sampling techniques on large graphs, but the performance improvement is still not satisfactory due to the lack of complete information. In this work, we surprisingly notice that the *node locality* on the graph can be used to reduce the information loss caused by the over-smoothing problem. As shown in Figure 1, in real-world graphs, most of the nodes have few connections, whereas some "central" nodes (hubs) are connected to many other nodes. By stacking multiple layers, the node can aggregate information from multi-hop neighborhoods. For "central" nodes in a cluster of the graph, deep GCN leads to a rapid expansion, which may expand too broadly and beyond the range of the cluster. These nodes involve an excess of neighborhoods and thereby the embeddings are over-smoothed and lose information. While non-central nodes rely on the deep architecture to obtain a sufficient neighborhood for stabilizing predictions [24]. Existing methods (e.g., ResGCN, DenseGCN, and JK-Net) ignore node locality and treat each node in the same way, resulting in the over-smoothing problem for deep architectures. Therefore, we raise the second question, *how to alleviate the over-smoothing problem of GCN when learning from the successful architecture of deep CNNs*? Besides, existing approaches directly port the experience from computer vision to graph embedding without an explanation about why it works. Consequently, we further raise the third question, based on the multi-layer architecture, *how to interpret and understand the effectiveness of these GCN layers*?

In this paper, to solve the above three problems, we first investigate the mutual information and analyze the information diminishing for existing deep GCN approaches. Then we study some popular architecture optimizations inspired by the architecture innovations on CNN, e.g., the concatenation from GoogleNet [30] and the dense connection from DenseNet [26]. Considering the over-smoothing problem for deep GCNs, we propose Lasagne, a multi-layer GCN framework. Although Lasagne also requires the information from previous layers, unlike previous studies forcing all nodes to pass the same number of GC layers, we enable different nodes to aggregate information from different layers. More concretely, we propose the node-aware architecture to *adaptively aggregate the node embeddings for different nodes in a layer level*, which makes the architecture fundamentally different from DenseGCN. With these optimizations, different nodes can benefit from different layers' information respectively. On top of the optimized multi-layer GCN architecture, we further demystify the model from a mutual information view. The observation confirms that our approach preserves more useful information across layers than vanilla GCN and its variants.

We summarize our contributions as four folds.

- We learn successful architectures from the CNNs, utilize multiple layers' information and propose Lasagne with node-aware architectures for a deep GCN model. To the best of our knowledge, Lasagne is *the first deep GCN architecture capturing the node locality* to alleviate the over-smoothing problem.
- We investigate the information theoretical approach to compare different models. The mutual information helps to interpret and understand the effectiveness of multi-layer architecture.
- Experimental results on benchmark datasets demonstrate that Lasagne effectively utilizes multiple layers' information and outperforms twenty four recent state-of-the-art GNN methods.
- We apply Lasagne to a real-world large-scale production dataset from our industry partner — Tencent Inc., and achieves significant performance improvement, compared to other GNN models without considering the node locality.

2 Related Works

In this section, we review some previous work on GCN for semi-supervised node classification. In addition, we also study some popular techniques which help CNN to obtain deeper architecture and better performance.

2.1 Graph Convolutional Networks

The original graph neural networks proposed in [31] collectively aggregate information from graph structure. There is an increasing interest in generalizing convolutions to the graph domain. The advances in this direction can be roughly categorized into two branches: spectral approaches and spatial approaches.

The spectral approaches define parameterized filters based on spectral graph theory. Spectral network [32] first introduces the convolutional operation in the Fourier domain by computing the eigendecomposition of the graph Laplacian. However, the graph Fourier transform is computationally expensive and limits the application for large graphs. ChebNet [33] approximates the K-polynomial filters utilizing a Chebyshev expansion of the graph Laplacian which helps to avoid the computation of the eigenvectors of the Laplacian. GCN [16] further simplifies the ChebNet and limits the layer-wise convolution operation to $K = 1$.

The spatial approaches generate node embedding by combining the neighborhood information in the vertex domain. Patchy-SAN [34] extracts and normalizes a neighborhood of exactly k nodes for each node. Then the neighborhood serves as a receptive field of the convolutional operation on the input graph. MoNet [35] integrates the local signals by designing a universe patch operator, which provides a unified generalization of CNN architectures to graphs. GraphSAGE [23] samples a fixed number of neighbors and employs several aggregation functions, such as concatenation, max-pooling, and LSTM aggregator. GAT [36] further introduces the attention mechanism to model different influences of neighbors with learnable parameters. However, the above two branches of GCNs are usually shallow (at most 2-4 layers), suffer from the over-smoothing problem with a deep depth of convolution architecture, and cannot obtain adequate global information.

2.2 Deep CNN Techniques and GCN

Before we introduce the deep GCN studies, we first revisit the history of CNN. The main boom in the use of CNN for image classification and segmentation occurred after it was observed that the representational capacity of CNN can be enhanced with an increased depth in AlexNet [37]. After that, the research in CNN achieves significant performance improvements because of architectural innovations. Visualization approach was used to improve the feature extraction approach by reducing the size of filters in [38]. In VGG [21], the depth is increased from 9 layers to 16, by making the volume of feature maps doubles at each layer. GoogleNet [30] merges multilevel transformations and helps CNN in tackling details of images at various levels. Representational depth improves generalization by defining a diverse level of features ranging from simple to complex. To solve the problem of vanishing gradients, ResNet [22] involves the concept of residual blocks or skip connections in CNN architecture for the training of 150 layers deep network. This technique is further extended to multi-layer connectivity by different researchers to improve representation (e.g., DenseNet [26], Stochastic Depth ResNet [39]. The breakthrough in CNN performance suggests that the representation depth is beneficial in improving the generalization of classifier.

The breakthrough in CNN performance suggests that the representation depth is beneficial in improving the generalization of the classifier. Recently, there is a trend to extend the deep architecture improvements on CNN to GCN. JK-Net [24] notices the importance of making use of multilevel transformation and combines multi-layers' output in GCN before the classifier. This design is similar to that of GoogleNet. And in ResGCN [16], the residual connection is also used between hidden layers to facilitate the training of deeper models by enabling GCN to carry over information from the previous layer's input. However, GCN generally achieves the best performance with 2 or 3 layers. The structure is further extended to DenseGCN [25] to overcome vanishing gradients problems and obtain better performance for deep GCNs. However, these approaches directly apply the CNNs architectures and ignore that the main obstacle for deep GCN is over-smoothing [27], rather than vanishing gradients. Our approach targets on mining the significant potential of improvement for deep GCN by making use of existing deep CNN techniques with node-aware designs.

TABLE 1
NOTATIONS

Symbols	Definitions
v_i	The ith node
$\boldsymbol{h}_i^{(l)}$	The embedding of the ith node at the lth layer
$\boldsymbol{H}^{(l)}$	The node embedding matrix at the lth layer
$\boldsymbol{W}^{(i)}$	The ith weight matrix
$\boldsymbol{A}$	The adjacency matrix
$\boldsymbol{D}$	The node degree diagonal matrix
$\boldsymbol{X}$	The node feature matrix
N	The size of the training set
M	The dimension of the node features
F	The class of data
$\boldsymbol{C}$	The larger aggregation weight matrix in Lasagne
$\boldsymbol{P}$	The sampling weight matrix in Stochastic aggregator

2.3 Tackling Over-Smoothing in Deep GCN

Besides involving deep CNN techniques, several recent attempts have also been proposed to tackle the over-smoothing problem in deep GCNs. APPNP [40] solves over-smoothing by using personalized PageRank that additionally involves the rooted node into the message passing loop. MixHop [41] mixes the node representations of long distance neighbors by combining the powers of the adjacency. ADSF [42] targets GAT and proposes a different way of computing the attention scores over the neighboring nodes, which considers not just their feature similarity, but also extra structure information up to k-hop neighbors. MADReg [43] and Pairnorm [44] involve regularization or normalization techniques to prevent the over-smoothing between close nodes. Unlike these methods, our approach proposes an adaptive node aggregation mechanism to capture the node locality, which makes Lasagne fundamentally different from existing techniques. Our method could be orthogonal to these previous efforts and provides a novel solution on the over-smoothing problem.

3 Preliminaries

3.1 Notations

To help the readers understand this work, we begin by introducing some notations related to Lasagne in Table 1. For the input undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with N nodes $v_i \in \mathcal{V}$, edges $(v_i, v_j) \in \mathcal{E}$, let $\boldsymbol{A} \in \mathbb{R}^{N\times N}$ be the adjacency matrix and $\boldsymbol{X} \in \mathbb{R}^{N\times M}$ be the node feature matrix, where

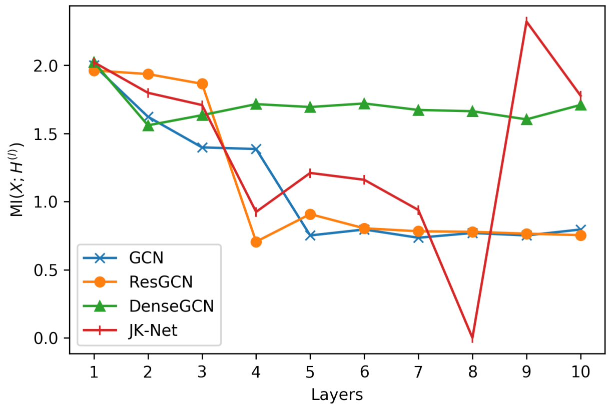


Fig. 2. The MI for 10-layer GCN models on Cora

M is the dimension of the attributive features. A multi-layer GCN follows the layer-wise propagation rule. At layer l, the output is the hidden representation $\boldsymbol{H}^{(l)}$:

$$\boldsymbol{H}^{(l)} = \delta(\widetilde{\boldsymbol{D}}^{-\frac{1}{2}}\widetilde{\boldsymbol{A}}\widetilde{\boldsymbol{D}}^{-\frac{1}{2}}\boldsymbol{H}^{(l-1)}\boldsymbol{W}^{(l)}), \tag{1}$$

where $\widetilde{\boldsymbol{A}} = \boldsymbol{A} + \boldsymbol{I}_N$ is the adjacency matrix of the undirected graph $\mathcal{G}$ with added self-connections. $\boldsymbol{I}_N$ is the identity, $\widetilde{\boldsymbol{D}}_{ii} = \sum_j \widetilde{\boldsymbol{A}}_{ij}$ and $\boldsymbol{W}^{(l)}$ is a layer-specific trainable weight matrix. $\delta(\cdot)$ denotes an activation function, such as $\mathrm{ReLU}(\cdot) = \max(0, \cdot)$. $\boldsymbol{H}^{(l-1)}$ is the input of lth layer and $\boldsymbol{H}^{(0)} = \boldsymbol{X}$. For an L-layer GCN on semi-supervised node classification on a graph with a symmetric adjacency matrix $\boldsymbol{A}$, we first calculate $\hat{\boldsymbol{A}} = \widetilde{\boldsymbol{D}}^{-\frac{1}{2}}\widetilde{\boldsymbol{A}}\widetilde{\boldsymbol{D}}^{-\frac{1}{2}}$ in a pre-processing step and the forward model takes the simple form:

$$\begin{aligned} &\boldsymbol{H}^{(l)} = \mathrm{ReLU}(\hat{\boldsymbol{A}}\boldsymbol{H}^{(l-1)}\boldsymbol{W}^{(l)}), l = 1..., L, \\ &\boldsymbol{Z} = f(\boldsymbol{X}, \boldsymbol{A}) = \mathrm{softmax}(\boldsymbol{H}^{(L)}). \end{aligned} \tag{2}$$

$\boldsymbol{H}^{(l)} \in \mathbb{R}^{N \times \boldsymbol{D}^{(l)}}$, where $\boldsymbol{D}^{(l)}$ is the parameter for the embedding dimension for the lth layer, $\boldsymbol{D}^{(0)} = M$ and $\boldsymbol{D}^{(L)} = F$ (F is the number of classes). The softmax activation function, defined as $\mathrm{softmax}(\boldsymbol{x}_i) = \frac{1}{\boldsymbol{S}}\exp(\boldsymbol{x}_i)$ with $\boldsymbol{S} = \sum_i \exp(\boldsymbol{x}_i)$, is applied row-wise. We then evaluate the cross-entropy error over all labeled examples:

$$L = \sum_{i \in \mathcal{Y}} \sum_{f=1}^{F} \boldsymbol{Y}_{if} \ln \boldsymbol{Z}_{if}, \tag{3}$$

where $\mathcal{Y}$ is the set of node indices that have labels.

3.2 Mutual Information

Deep models improve model expressiveness, but also introduce new challenges: *vanishing gradients* in backward propagation and *diminishing feature reuse* in forward propagation. Vanishing Gradients is a well-known nuisance in neural networks with many layers and has been successfully solved by many studies (e.g., batch normalization [45], residual blocks [22]). Diminishing feature reuse (also known as loss in information flow [46]) refers to the analogous problem to vanishing gradients in the forward direction. The features of the input instance, or those computed by earlier layers, are "washed out" through repeated multiplication or convolution with weight matrices, making it hard for later layers to identify and learn "meaningful" information. To be specific, the over-smoothing problem occurred in multi-layer GCN models is caused by diminishing feature reuse.

As mentioned in Section2.2, some existing approaches involve deep CNN architecture techniques to improve information preservation in GCN. To provide a microscopic view of these models, we investigate the information theoretical approach and focus on Mutual Information (MI), a metric quantifying the amount of information shared between two random variables. The MI is a theoretic measurement independent of any classifiers which reflects the relationship between the variables even if the relationship is highly nonlinear and hidden in high-dimensional data [47]. An important advantage of introducing MI as an indicator is that: when two networks achieve similar accuracy, the one with a higher MI is more suitable for the classification tasks [47]. For example, in [48], video sequences are required to classify into high-level human actions by considering multi-frames. A DNN with high MI will be able to perform better classification comparing to another DNN with low MI due to the well distributed feature representations of individuals in the model. In other words, the network architecture with higher MI could preserve more information behind the data and could be more helpful to the bottom classifier. Therefore, MI can provide an important metric to better understand multi-layer GCNs.

As shown in Figure 2, we estimate MI between the output of hidden layers $\boldsymbol{H}^{(l)}$ with the input features $\boldsymbol{X}$ after the models converged. The visualization of the MI for vanilla GCN illustrates that different hidden layers carry different levels of information and the low MI of the final layer further proves the over-smoothing problem [27]. ResGCN can reduce the loss of information across the layers for shallow layers. But for deeper layers, the over-smoothed information is directly added to the next layer and might result in bad performance. The combination operation in JK-Net significantly increases the MI value of the last two layers. And the dense concatenation in DenseGCN improves all hidden layers of information retention. The higher MI of the last layer the model has, the better performance the model may achieve in the final classification task. These observations motivate us to better utilize the multi-layer information in our following designs.

4 Methodology

In this section, we present the multi-layer framework, Lasagne, based on GCN. Note that, Lasagne is also applicable to other models (e.g., GAT, GraphSAGE) that apply the multi-layer neighborhood aggregation operations. As shown in Figure 3, the dense connection makes each layer collects information from different hops of neighbors and becomes a unique convolutional filter based on the previous layer, which is similar to the scheme in CNN of computer vision area [26]. It not only helps to reduce the gradients vanishment but also preserves the node representations at different levels of abstraction and prevents the information loss [47]. Considering the different node localities as we proposed in the Introduction, we propose a

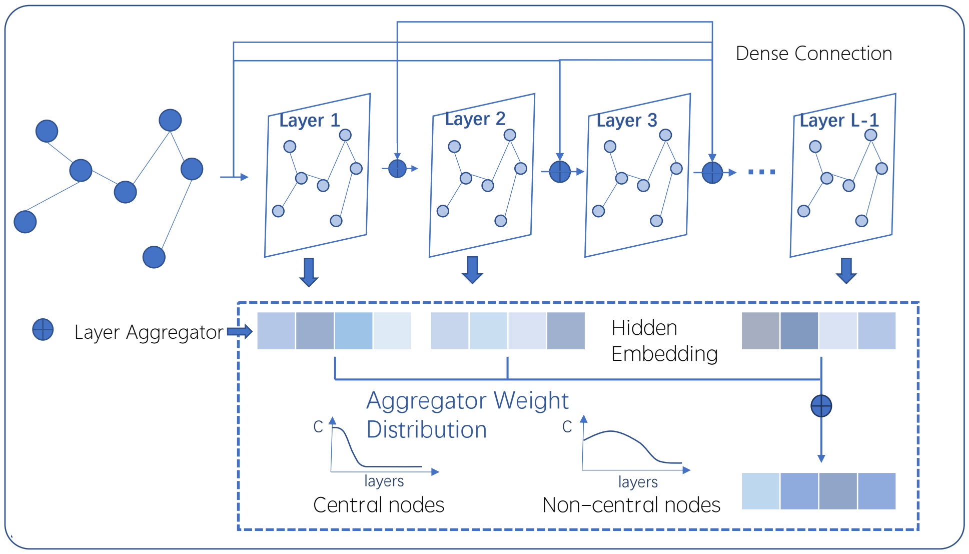


Fig. 3. The main architecture of Lasagne.

layer aggregator structure and design three different node-aware aggregators. They can automatically learn the hidden representations from the needed layers respectively for each node, which helps to alleviate the over-smoothing problem to some extent. Besides, the vertex-wise addition operation in ResGCN and DenseGCN leads to the restriction which requires the same dimension for all layers. It may hurts model performance and has been removed from Lasagne because our layer aggregators support flexible hidden dimensions among these layers. After the main architecture of Lasagne, we further design a GC-FM layer (introduced in Section 4.2) as the last graph convolutional layer to automatically capture the interactions between the node representation from different layers and pass its output to the downstream classifier.

4.1 Layer Aggregator

As shown in Figure 3, there is a layer aggregator after each layer's output. The layer aggregator aggregates all previous layers' hidden representations and makes up of a dense connection structure. DenseGCN [25] uses a straightforward vertex-wise concatenation operation to densely fuses the input graph with all the intermediate GCN layer outputs. However, as the concatenation just treats the node hidden representations from different layers in the same way, it cannot capture the node locality. To better utilize different levels of node abstraction, we propose the node-aware layer aggregators in Lasagne. The forward model is formulated as:

$$\boldsymbol{H}^{(l)} = \text{Aggregator}(\boldsymbol{C}^{(l)}, \boldsymbol{H}^{(1)}, \boldsymbol{H}^{(2)}, ..., \boldsymbol{H}^{(l)}), \quad (4)$$

where $1 < l < L$, $\boldsymbol{C}^{(l)} \in \mathbb{R}^{N \times l}$ is the weight matrix, *enabling different nodes using a different weighted aggregation for the previous layers*. For the l-th layer's output of a specific node v_i, the trainable parameter $\mathbf{C}_{i,j}^{(l)}$ represents for the contribution from the j-th layer's output.

During the training procedure, Lasagne could adaptively help each node learn how to aggregate information from different layers. As verified in our experiments (Depth analysis), the node locality could leads to different aggregator weight distribution. More concretely, for "central" node, deep GCN leads to a rapid expansion, which may expand too broadly and beyond the range of the cluster. These nodes involve an excess of neighborhoods and thereby the embeddings are over-smoothed and lose information. While non-central nodes rely on the deep architecture to obtain a sufficient neighborhood for stabilizing predictions. Previous studies ignore the above node locality, *apply the same number of GC layers on the nodes and finally lead to the over-smoothing problem*. Our model permits general layer-aggregation mechanisms, and we explore three special approaches below. Others custom aggregation operations (e.g., mean, LSTM) are also possible.

4.1.1 Weighted aggregator

In our architecture, each node hidden representation relies on its all previous layers' information. However, the nodes have a different preference in these layers. For example, nodes with few neighbors may need more layers to involve enough information. While centric nodes with large degrees are facing over-smoothing problem after many graph convolutional layers, hence a shallow GCN is are quite enough. Considering the different node localities, we propose the weighted aggregator which uses an additional matrix $\boldsymbol{C}^{(l)} \in \mathbb{R}^{N \times l}$ for the i-th layer to capture the different contribution of previous layers for each node. The operation is a kind of weighted-sum on the hidden representations:

$$\boldsymbol{H}^{(l)} = \sum_{i=1}^{l-1} \hat{\boldsymbol{A}}\boldsymbol{C}^{(l)}[:, i] \otimes \boldsymbol{H}^{(i)}\boldsymbol{W}^{(il)} + \boldsymbol{C}^{(l)}[:, l] \otimes \boldsymbol{H}^{(l)}, \quad (5)$$

where $\boldsymbol{C}^{(l)}[:, i]$ is the ith column of the contribution matrix $\boldsymbol{C}^{(l)}$ and represents for the importance of the ith layer's hidden representation for all nodes, $\otimes$ broadcasts $\boldsymbol{C}^{(l)}[:, i]$ to an $N \times L$ matrix and performs element-wise multiplies with $\boldsymbol{H}^{(i)}$. Notice that we not only add weighted-sum but also apply an additional GC transformation i.e.,

$\boldsymbol{W}^{(il)} \in \mathbb{R}^{D^{(i)} \times D^{(l)}}$ is the parameters. Recently, the area of automated machine learning (AutoML) and neural network architecture search (NAS) has noticed the graph neural networks and proposes to improve the graph learning performance through exploring the search space. Many of them [49], [50], [51] have clarified that the hidden dimension is a crucial component of the search space, which significantly affects the model performance. Therefore, removing the same-dimension limitation could provide more chances of exploring more hidden dimension combination choices and improving the GCN model performance. We remove the limitation in ResGCN of keeping all hidden dimensions the same and replace the linear transformation in JK-Net to a GC layer, which accelerates the information propagation.

4.1.2 *Max Pooling aggregator*

This operation is proposed in some recent advancements in applying neural network architectures to learn over general point sets in the computer vision area [52]. Intuitively, the multi-layer GCN can be thought of as a set of functions that compute features for each node representations. By applying the max-pooling operator to each of the computed features, the model effectively captures the most informative layer for each feature coordinate without any additional parameters to learn. The max pooling aggregator can be viewed as a special case of weighted aggregator but scale $\boldsymbol{C}^{(l)}$ from $\mathbb{R}^{N \times l}$ to $\mathbb{R}^{N \times l \times \boldsymbol{D}^{(l)}}$. For each node v_i, the weight matrix is $\boldsymbol{C}_i^{(l)} \in \mathbb{R}^{l \times \boldsymbol{D}^{(l)}}$ with the constraint that each column of $\boldsymbol{C}_i^{(l)}$ only has one 1 item and the rest of them are 0. Max pooling is adaptive and has the advantage that it does not introduce any additional parameters to learn.

4.1.3 *Stochastic aggregator.*

Inspired by the Stochastic Depth ResNet method [39], during training we shorten the network significantly by randomly removing a substantial fraction of layers independently for each iteration. Shortening the depth during training reduces the chain of forward propagation steps and gradient computations, which strengthens the information propagation. Another advantage is that networks trained with stochastic depth can be interpreted as an implicit ensemble of networks of different depths, mimicking the record breaking ensemble of depth varying GCNs.

Different from the fixed survival probabilities in Stochastic Depth ResNet, we propose a learnable activation function for each layer of each node. The stochastic aggregator adopts a layer-wise-dropout manner that using the Bernoulli sampling procedure to aggregate embeddings. The form of the operation is identical to Eq (5), but the difference is that each item of $\boldsymbol{C}$ is an independent Bernoulli random variable. To be specific, $\boldsymbol{C}_{ij}$ represents for the activated probability for ith node's jth layer, formulated as:

$$\boldsymbol{C}_{ij} \sim Bernoulli(\frac{e^{\boldsymbol{P}_{il}}}{\max\limits_{1 \le j \le L-1} \{e^{\boldsymbol{P}_{ij}}\}}), \tag{6}$$

where $\boldsymbol{P} \in \mathbb{R}^{N \times L}$ is the trainable parameters that determine the probability distributions.

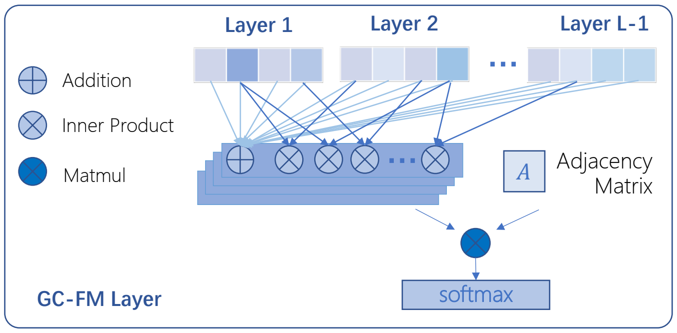


Fig. 4. Illustration of GC-FM layer.

4.2 Layer Interaction

Factorization Machines (FM) in their general form models the interactions among individual features and have been used for both sparse and dense representations in deep neural networks [53], [54]. By assuming the embedding dimensions are independent with each other, Convolutional Factorization Machine [55] proposes a multi-layer CNN to learn the feature interaction patterns between embedding dimensions. Inspired by these approaches, we propose a novel GC-FM layer to preserve the localized spectral filter and automatically capture the interactions between different layers' embedding at the same time. We formulate it as $\boldsymbol{H}^{(L)} = \text{ReLU}(\hat{\boldsymbol{A}}\boldsymbol{O})$,

$$\begin{aligned} \boldsymbol{O}_{ij} &= \langle \boldsymbol{W}^{(L)}[:, j], \boldsymbol{h}_i \rangle + \\ &\sum_{p=1}^{L-2} \sum_{m=1}^{\boldsymbol{D}^{(L-1)}} \sum_{q=p+1}^{L-1} \sum_{n=1}^{\boldsymbol{D}^{(L-1)}} \langle \boldsymbol{V}_{jpm}, \boldsymbol{V}_{jqn} \rangle \boldsymbol{h}_{ipm} \cdot \boldsymbol{h}_{iqn}, \end{aligned} \tag{7}$$

where $\boldsymbol{O} \in \mathbb{R}^{N \times F}$, $\boldsymbol{V}_{jpm} \in \mathbb{R}^k$ and k is the latent constant parameter for FM. As shown in Figure 4, we illustrate the calculation procedure of Eq (7). For each node, we concatenate all its hidden representations as the input of GC-FM layer, which equals $[\boldsymbol{h}_i^{(1)}, \boldsymbol{h}_i^{(2)}, ..., \boldsymbol{h}_i^{(L-1)}]$.

The first part of Eq (7) is the addition for all embedding dimensions and the second part is the pair-wise inner products. Please note that we only interact between different layers' embeddings. After the factorization operation, we perform the convolutional filter $\hat{\boldsymbol{A}}$ to propagate the interacted information among graphs. It can be regarded as a convolution in the depth direction for the layer interactions in a rather explicit manner.

5 Experiments

To validate the effectiveness of Lasagne, we test its performance with some popular semi-supervised node classification tasks. The source code of Lasagne is released at [56].

5.1 Experiment setup

5.1.1 *Datasets*

We evaluate Lasagne on 11 different graph datasets including both popular benchmark datasets and a real-world industrial dataset. The detailed statistics of these datasets are listed in Table 2.

TABLE 2
Overview of datasets

Dataset	#Nodes	#Features	#Edges	#Classes	#Train/Val/Test	Task	Description
Cora	2,708	1,433	5,429	7	140/500/1,000	Transductive	citation network
Citeseer	3,327	3,703	4,732	6	120/500/1,000	Transductive	citation network
Pubmed	19,717	500	44,338	3	60/500/1,000	Transductive	citation network
NELL	65,755	61,278	266,144	210	6,575/500/1,000	Transductive	knowledge graph
Amazon Computer	13,381	767	245,778	10	200/300/12,881	Transductive	co-purchase graph
Amazon Photo	7,487	745	119,043	8	160/240/7,087	Transductive	co-purchase graph
Coauthor CS	18,333	6,805	81,894	15	300/450/17,583	Transductive	citation network
Coauthor Physics	34,493	8,415	247,962	5	100/150/34,243	Transductive	citation network
Flickr	89,250	500	899,756	7	44,625/22,312/22,312	Inductive	image network
Reddit	232,965	602	11,606,919	41	155,310/23,297/54,358	Inductive	social network
Tencent	1,000,000	64	1,434,382	253	5,000/10,000/30,000	Transductive	user-video graph

Benchmark. We use three citation network datasets (e.g., Citeseer, Cora, and Pubmed), two social network datasets (e.g., Flickr and Reddit), one knowledge graph dataset (e.g., NELL) and four additional datasets (e.g., Amazon and Coauthor). For comparison, we use the released partitioned datasets for the three citation networks as in [16], the two social networks as in [57], the knowledge graph as in [58] and the four additional datasets in [59].

Production. Unlike the previous widely used benchmark datasets, the Tencent dataset is an user-video graph, collected from a real-world mobile application from our industry partner Tencent Inc.. We obtained 100,000 nodes and the correspond watching record from the history. The generated graph is a bipartite graph including 57,022 short-videos with labels and 42,978 users. The edge between each pair of short-video item and user represents that the user have watched this short-video. Each user has 64 features. Our target is to classify these short-videos into 253 different predefined classes. Some early approaches usually exploit the interactions between the user-item bipartite graph through collaborative filtering methods [60], [61]. However, these approaches lack an explicit encoding of the crucial collaborative signal which is latent in user-item interactions to reveal the behavioral similarity between users (or items). For example, NGCF [62] first proposes to exploit the high-order connectivity from user-item interactions, which is a natural way that encodes collaborative signal in the interaction graph structure. LightGCN [63] follows the same intuition and involves GCN to exploit the user-item graph structure for recommendation. Both NGCF and LightGCN are evaluated with 4 layers model architecture to aggregate multi-hop neighborhoods information and the experiment results have verified the performance improvements due to the high-order connectivity. These approaches motivate us to utilize deep GCN to improve the industrial user-video recommendation performance.

5.1.2 Baselines

To evaluate the performance of Lasagne, we compare our method with 20 representative methods including GCN, JK-Net, ResGCN, DenseGCN, GAT, GraphSAGE (as we introduced in Section 2) and the following state-of-the-art methods:

- GPNN [58] exploits a propagation schedule combining features of synchronous and sequential propagation schedules.
- NGCN [64] trains multiple instances of GCNs over node pairs discovered at different distances in random walks and learns a combination of the instance outputs which optimizes the classification objective.
- DGCN [65] considers the local consistency and global consistency in semi-supervised learning and it uses two convolutional neural networks to embed the local-consistency-based and global-consistency-based knowledge respectively.
- DropEdge [29] randomly removes a certain number of edges from the input graph to reduce the convergence speed of over-smoothing.
- STGCN [66] generalizes spectral graph convolution and deep GCN in block Krylov subspace forms and devise deeper architectures.
- DGI [67] leverages local mutual information maximization across the graph's patch representations.
- GMI [68] extends the idea of DGI and maximize the mutual information of both node features and edges between inputs and node representations.
- GIN [69] characterizes certain graph structures via multiset aggregation to improve the model representational power.
- SGC [70] proposes to simplify the multi-layer GCN architectures by removing the activation functions.
- LGCN [71] transforms graph data into grid-like structures and adopts a sub-graph training method.
- APPNP [40] utilizes the personalized PageRank scheme to improve the information propagation.
- FastGCN [72] proposes to sample a fixed number of nodes at each layer via importance sampling.
- ClusterGCN [28] limits the training inside graph partitions to alleviate the neighborhood expansion.
- GraphSAINT [57] proposes a variance reduction based graph sampling method and normalization technique to eliminate bias.
- Pairnorm [44] prevents over-smoothing by introducing a pairwise normalization scheme to make distant node pairs having less similar node representations.
- ADSF [42] proposes an adaptive structural fingerprint model to exploit graph topology in GAT. It incorporates both graph structures and node features similarities while computing attention scores.
- MixHop [41] mixes the powers of the adjacency

TABLE 3
The test accuracy (in %) on the citation dataset. * indicates that we ran our own implementation.

Models	Cora	Citeseer	Pubmed
GPNN	81.8	69.7	79.3
NGCN	83.0	72.2	79.5
DGCN	83.5	72.6	80
DropEdge	82.8	72.3	79.6
STGCN	83.6	72.6	79.5
DGI	82.3±0.6	71.8±0.7	76.8±0.6
GMI	82.7±0.2	73.0±0.3	80.1±0.2
GIN	77.6±1.1	66.1±0.9	77.0±1.2
SGC	81.0±0.0	71.9±0.1	78.9±0.0
LGCN	83.3±0.5	73.0±0.6	79.5±0.2
APPNP	83.3±0.5	71.8±0.5	80.1±0.2
GAT	83.0±0.7	72.5±0.7	79.0±0.3
Pairnorm*	81.4±0.6	68.5±0.9	79.1±0.5
ADSF*	83.8±0.5	72.8±0.7	80.1±0.8
MixHop*	82.1±0.4	71.4±0.8	80.0±1.1
MADReg*	82.3±0.8	71.6±0.9	79.5±0.6
GCN*	81.8±0.5	70.8±0.5	79.3±0.7
JK-Net*	81.8±0.5	70.7±0.7	78.8±0.7
ResGCN*	82.2±0.6	70.8±0.7	78.3±0.6
DenseGCN*	82.1±0.5	70.9±0.8	79.1±0.9
Lasagne (Weighted)*	84.1±0.2	73.2±0.5	79.5±0.4
Lasagne (Stochastic)*	**84.2±0.5**	73.1±0.6	**80.2±0.5**
Lasagne (Max pooling)*	84.1±0.8	**73.3±0.5**	79.6±0.6

matrix and combines the node representations of neighbors at various distances.

- MADReg [43] proposes the mean average distance (MAD) to reflects the node smoothness and adds a MADGap-based regularizer for better GNN training.

Note that, our techniques are orthogonal to these baselines and can be used to further improve their performance. Here we evaluate our method on vanilla GCN to demonstrate the effectiveness of Lasagne.

5.1.3 Experiment Settings

We use PyTorch to implement the models and we train them using Adam optimizer with a learning rate of 0.02 for the citation datasets and Tencent, 0.005 for Reddit, and 0.01 for other datasets. Besides, the l_2 regularization factor is set to 5e-4 for the citation datasets and 1e-5 for the other datasets. The dropout is applied to all feature vectors with rates of 0.8 to the citation dataset, 0.5 to Flickr and Tencent, 0.2 to Reddit, and 0.3 to other datasets. We train each model 400 epochs and terminate the training process if the validation accuracy does not improve for 20 consecutive steps. For the GC-FM layer, we set its latent parameter k to 5. Last, the dimension of hidden features 32 for citation datasets and 100 for other datasets. Note that JK-Net has three aggregators, and we choose the concatenation as the final aggregation layer since it performs best on the citation dataset. We run each method 10 times and and report the mean accuracy and the standard deviation to eliminate random factors.

To make a fair comparison with existing baselines, we use the standard released partitioned datasets for both transductive tasks (especially for the three widely-used citation networks) and inductive tasks. The data splitting difference fundamentally determines whether we could reuse the reported results of them. Therefore, for GraphSAGE, we use the results on Flickr and Reddit as reported in [23] and [57].

TABLE 4
Additional comparison on inductive tasks. * indicates that we ran our own implementation.

Models	Flickr	Reddit
GraphSAGE	50.1±1.3	95.4±0.0
FastGCN	50.4±0.1	93.7±0.0
ClusterGCN	48.1±0.5	96.6±0.0
GraphSAINT	51.1±0.1	96.6±0.1
Lasagne*	**52.9±0.2**	**96.7±0.1**

For ClusterGCN, we use the results on Reddit as reported in [28], and run our own implementation on Flickr. We have also empirically verified these results to guarantee that the other factors (e.g., random seeds and parameter settings) could not lead to significant differences.

5.2 Results Analysis

5.2.1 Comparison to state-of-the-art

We compare our Lasagne with the state-of-the-art methods on different datasets, including both benchmark datasets and the production dataset. We introduce and analyze these results in the following.

Transductive As shown in Table 3, Lasagne outperforms the baselines by a significant margin. Specifically, Lasagne achieves better performance over the current state-of-the-art methods by a margin of 0.4% (i.e., ADSF), 0.3% (i.e., GMI), and 0.1% (i.e., APPNP and ADSF) on the three citation datasets respectively. Compared to existing deep GCN techniques, Lasagne not only involves multi-hop neighbors while aggregating node representations, but also proposes an adaptive node-locality-aware mechanism to learn how to utilize these information, which leads to the superior performance. Besides, the performance of Lasagne using different aggregator is roughly the same in Cora and Citeseer. However, on Pubmed, the Stochastic aggregator can achieve the accuracy of 80.2%, while the second-best aggregator Max pooling can only get the accuracy of 79.6%, which means different aggregators may result in different performance. We further evaluate Lasagne on a variety of other datasets such as Coauthor CS, Coauthor Physics, Amazon Computers and Amazon Photo and the results in Table 5 demonstrate that Lasagne could also outperforms the baselines.

Inductive Besides the transductive tasks, we also make an addition comparison on inductive tasks, where the Weighted aggregator and the Stochastic aggregator in Lasagne are not suitable. Only the nodes in the training set can be used during the training procedure, and the pre-trained parameters of these aggregators lose efficacy on the validation set and testing set. Even so, Table 4 shows the Lasagne (Max pooling) can still outperform GraphSAGE [23], FastGCN [72], ClusterGCN [28] and GraphSAINT [57] on Flickr and Redddit.

Production For the production dataset Tencent with one million nodes, the oversmoothing becomes more crucial. For each short-video node of the user-video graph, the edges represents for concurrent clicks on the short-video by the users. The "hot" short-videos could be watched by most of the users, which makes them nearly indistinguishable by

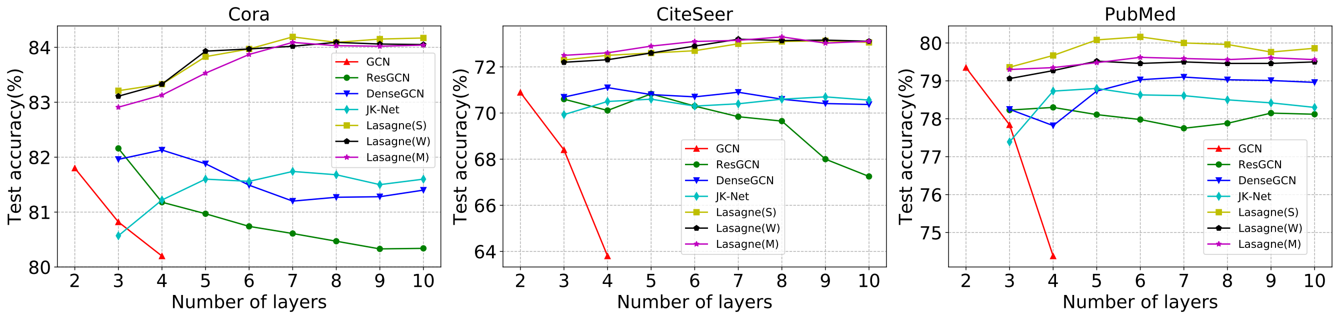


Fig. 5. Influence of model depth (number of layers) on classification performance.

TABLE 5
Test accuracy (in %) on other datasets. * indicates that we ran our own implementation.

Models	Amazon Computer	Amazon Photo	Coauthor CS	Coauthor Physics	Tencent
GAT*	80.1±0.6	85.7±1.0	87.4±0.2	90.2±1.4	46.8±0.7
GCN*	82.4±0.4	85.9±0.6	90.7±0.2	92.7±1.1	45.9±0.4
JK-Net*	82.0±0.6	85.9±0.7	89.5±0.6	92.5±0.4	47.2±0.3
ResGCN*	81.1±0.7	85.3±0.9	87.9±0.6	92.2±1.5	46.8±0.5
DenseGCN*	81.3±0.9	84.9±1.1	88.4±0.8	91.9±1.4	46.5±0.6
Lasagne (Weighted)*	83.9±0.7	87.4±0.4	92.4±0.6	93.8±0.5	47.6±0.3
Lasagne (Stochastic)*	**84.5**±0.7	88.2±0.4	**92.5±0.5**	**94.1±0.6**	**48.7±0.5**
Lasagne (Max poling)*	84.1±0.4	**88.7±0.8**	92.1±0.5	93.8±0.5	48.1±0.6

the aggregated node embeddings. In such cases, the node-aware characteristic of Lasagne is essential to preserve the personalized short-video features information. The results in Table 5 verify the significant improvement from Lasagne on large scale graph data.

5.2.2 *Depth analysis*

Lasagne is a representation learning framework which can be used to train a very deep GCN, thus we investigate the influence of model depth (number of layers) on classification performance on the three citation datasets. We compare Lasagne under different aggregators with ResGCN, DenseGCN, and JK-Net since all of them aim at training a deeper model. For each dataset, we calculate its Average Path Length (APL) [73],

$$L = \frac{2}{N(N-1)} \sum_{\forall v_i, v_j \in \mathcal{G},} d(v_i, v_j), \quad (8)$$

where $d(v_i, v_j)$ is the length of the shortest path between nodes v_i and v_j, N is the number of nodes. It is a measure of the efficiency of information transport on the graph $\mathcal{G}$, which means each node in the graph theoretically should capture the max L-hop neighborhood information. The APL for Cora, Citeseer, Pubmed and NELL are 7.3, 10.3, 6.3 and 5.4 respectively, so we just analyze the depth influence up to 10 layers.

As shown in Figure 5, for the original GCN, it gets the best results with a 2-layer model and its performance decreases rapidly with the addition of layers. For ResGCN, DenseGCN, and JK-Net, they can keep more information of the original features compared with GCN and get a relatively good performance. However, their performance is much lower than Lasagne, that's because they ignore the difference between each node and thus introduce the over-smoothing problem for the high degree nodes in the deep layers. Our proposed method Lasagne outperforms other methods a lot using different layers. Specifically, we find that there is no significant superior aggregator for Lasagne on Cora and CiteSeer. But on Pubmed, the Stochastic aggregator is significantly better than the other two aggregators. We infer that the stochastic aggregation mechanism might provide better generalization ability for deep GCN. Besides, even with very high depth, the performance of Lasagne does not decrease as the other baselines do. More importantly, we see that Lasagne gets the best result with more than 5 layers in each dataset, which proves the necessity of making a deep model.

Besides, we collect the trainable Stochastic aggregator parameters $\boldsymbol{P}$ from a 5-layer GCN on Cora and then use the page rank (PR) score to measure the node localities. We find that different nodes indeed show different probability distribution. For example, the node with the largest PR value (central) has a $\boldsymbol{P}$ distribution as [1.00, 0.95, 0.89] and the node with the lowest PR (non-central) has a distribution of [0.67, 0.86, 1.00]. The distribution vector represents the probability for the first three layers to be aggregated for the current node. The results show that the central node prefers to aggregate nearby neighbors while the non-central node prefers more distinct neighbors. The statistical results on $\boldsymbol{P}$ is helpful for the model interpretation and the hub hypothesis, and we are glad to study this in-depth in our future work.

5.2.3 *Model Interpretation*

To interpret the effectiveness of our method, we further run 10-layer Lasagne and other baselines on Cora. As shown in Figure 6, we evaluate their MI between the last layer's

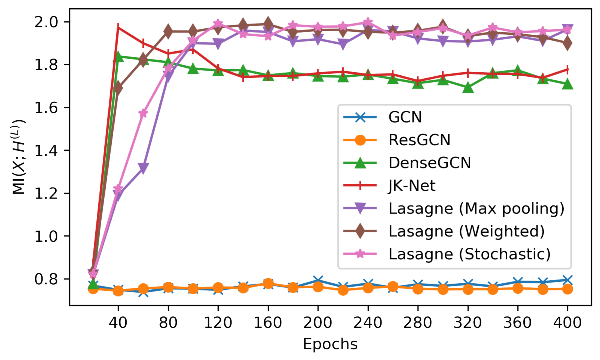


Fig. 6. The MI for the last layer during training on Cora

TABLE 6
The test accuracy (in %) with or without GC-FM on three datasets

Aggregators	Cora		Citeseer		PubMed	
	baseline	+GC-FM	baseline	+GC-FM	baseline	+GC-FM
Weighted	83.8±0.4	84.1±0.2	72.9±0.3	73.2±0.3	79.4±0.4	79.5±0.4
Stochastic	84.0±0.3	84.2±0.5	72.5±0.5	73.1±0.4	79.8±0.6	80.2±0.5
Max Pooling	83.7±0.8	84.1±0.8	72.7±0.7	73.3±0.6	79.3±0.5	79.6±0.6

TABLE 7
The test accuracy with and without Lasagne (stochastic aggregator).

Models	Cora		Citeseer		PubMed	
	baseline	+Lasagne(S)	baseline	+Lasagne(S)	baseline	+Lasagne(S)
GCN	81.8±0.5	84.2±0.5	70.8±0.5	73.1±0.6	79.3±0.7	80.2±0.5
SGC	81.0±0.3	83.9±0.5	71.9±0.3	72.6±0.4	78.9±0.1	80.1±0.3
GAT	83.0±0.7	84.1±0.7	72.5±0.7	73.1±0.8	79.0±0.3	79.7±0.5

hidden representation and the input feature. DenseGCN and JK-Net aggregate former layers' outputs, achieve high MI at the beginning and drop down quickly because of the over-smoothing. The curves illustrate that our method achieves the highest MI than other baselines. This means that Lasagne indeed helps multi-layer GCN to preserve more information than other approaches, thus improving the effectiveness of the proposed deep architecture.

5.2.4 *Ablation study*

To verify the effectiveness of the proposed GC-FM layers in Lasagne, we conduct ablation study on three citation networks, and the results are shown in Table 6. More concretely, we remove the GC-FM layer and replace it with the graph convolution layer of Lasagne and compare its performance with the original Lasagne. As shown in Table 6, it is evident that on all three citation datasets, Lasagne has better performance compared to Lasagne without GC-FM layer using different aggregators. For example, the performance gains on Citeseer are 0.3%, 0.6% and 0.6% respectively for the three different aggregators. These results verify that the proposed GC-FM layer could bring some performance improvements over Lasagne on different datasets because it can learn the cross features between each embedding layer.

5.2.5 *Effects on other GNNs*

Lasagne is a general node-aware deep GCN framework and can be applicable to different GNN models easily. To verify this, we conduct experiments to testify the effectiveness of the framework on other GNN models, such as SGC and GAT. We remain the node aggregation mechanism in each single layer of SGC and GAT (i.e., powers of adjacency matrix in SGC and self-attention among neighbors in GAT), but replace the deep architecture with Lasagne (Stochastic). Even with different base models, Table 7 shows that our proposed stochastic aggregator can boost the model performance in three citation networks. With Lasagne, the performance of base models can be boosted by a large margin of 2.9%, 2.4% and 1.1% in Cora, Citeseer and PubMed respectively, verifying its high generalization ability.

5.2.6 *Analysis on graph sparsity*

We evaluate with various sparsity level of the graph by taking the different label rates per class. In Table 8, we report node classification accuracy along with 5, 10, 15 and 20 labeled nodes per class (corresponding to the label rate of 1.3%, 2.6%,3.9%, and 5.2%) on Cora. To measure the influence of the sparsity level on the big graph, we also report the test accuracy along with the increased label rate per class on NELL. When the label rate decreases, our proposed method Lasagne still outperforms the baselines. As shown in Table 8, the test accuracy of Lasagne on Cora exceeds the next best model by 1.6%, 1.8%, 1.6% and 2.0% when the label rate is 1.3%, 2.6%, 3.9%, and 5.2% respectively. For the result on NELL, our method Lasagne still gets the highest performance compared with other baselines under different label rate, which means Lasagne is capable of training a large graph with limited labeled nodes.

5.3 Efficiency comparison

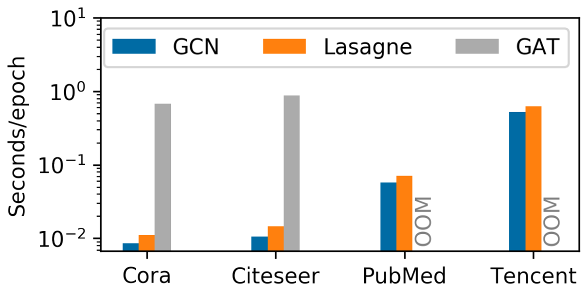
Although our proposed Lasagne significantly outperforms other state-of-the-art methods, we also concern the execution efficiency. In the following, we demonstrate that the proposed method achieves comparable training speed with the original GCN, while maintaining superior prediction performance.

In Figure 7(a), we compare the per epoch time (in seconds) of GCN, Lasagne (Weighted) and GAT with the same model depth (4 layers) on citation datasets and Tencent dataset. The hardware environment is the NVIDIA TITAN RTX GPU with 24 GB RAM. As we can see, Lasagne always performs similar as the original GCN. Because the computation complexity of Lasagne is the same as GCN asymptotically. The extra overheads in Lasagne mainly come from the layer aggregators including the element-wise multiplications and additional GC transformations, which only incur linear time cost. These operations help Lasagne capture the node locality with slight efficiency loss.

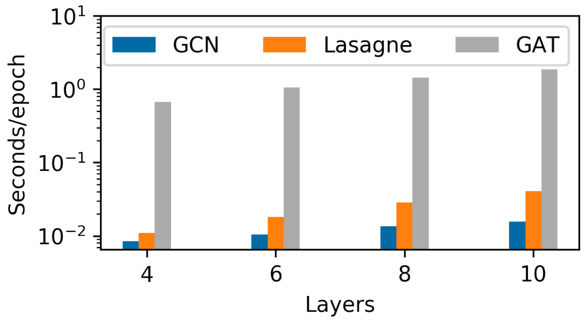
GAT manages to learn the individual node aggregation pattern through adding an additional matrix multiplication operation for each edge. The self-attention module [74] automatically learns the node aggregation weights and could have chance to capture the node localities potentially. However, the over-elaborate model design makes the learning more difficult due to over-parameterization and leads to serious efficiency problem at the same time. For example, running a 4-layer GAT could be 100× slower than GCN and exceed the 24 GB GPU memory on large graphs, e.g., Pubmed and Tencent. Besides, it has been observed that the over-smoothing problem is still unresolved in GAT [75].

TABLE 8
The test accuracy (in %) along with the increased label rate per class on Cora and NELL.

Models	Cora				NELL		
Label rate	**1.3%**	**2.6%**	**3.9%**	**5.2%**	**0.1%**	**1%**	**10%**
GCN	74.8	76.7	79.3	81.8	54.2	67.0	83.0
ResGCN	75.6	77.3	79.7	82.2	64.3	73.1	82.1
DenseGCN	75.1	77.1	80.1	82.1	65.2	72.8	83.4
JK-Net	73.9	76.6	79.4	81.8	58.4	73.4	84.1
Lasagne (Weighted)	76.8	78.8	**81.7**	84.1	**66.8**	**75.8**	84.8
Lasagne (Stochastic)	77.0	78.9	81.6	**84.2**	66.3	74.9	**85.2**
Lasagne (Max pooling)	**77.2**	**79.1**	81.4	84.1	66.1	75.2	84.5



(a) Per epoch time (s) comparison on different datasets (depth=4)



(b) Per epoch time (s) comparison with different depth on Cora

Fig. 7. Efficiency comparison among GCN, Lasagne (Weighted) and GAT.

In fact, our method could be viewed as a simplified attention mechanism among different layers for each node, but provide more powerful model performance than GAT. We also evaluate the efficiency as the network architecture goes deeper. As shown in Figure 7(b), Lasagne still performs comparable speed with the original GCN with even 10 layers, while preserving superior classification performance as introduced in Figure 5.

6 Conclusions

In this paper, we study some recent approaches involving CNN techniques potentially useful for deep GCNs and interpret these methods with Mutual Information. Then we propose a novel GCN framework, Lasagne, which consists of multiple graph convolutional layers and the dense connection across the layers. Based on this architecture, we propose three node-aware layer aggregators including max pooling, weighted and stochastic aggregators. By applying GC-FM layer, Lasagne can further capture the layer interaction information. Compared with other existing approaches, our proposed Lasagne can better utilize different layers' information in deep GCN. We conduct comprehensive experiments and the results confirm that our method consistently outperformed other state-of-the-art methods. In particular, our method achieves the largest MI preservation over them for deep GCNs. According to our experiments, different aggregators may result in very different performance on the same dataset. Therefore, how to make them interpretable and then instruct us to design and select the appropriate aggregator is an open question we would like to solve in our future work.

Acknowledgments

This work is supported by the National Key Research and Development Program of China (No. 2018YFB1004403), the National Natural Science Foundation of China under Grant (No. 61832001), Beijing Academy of Artificial Intelligence (BAAI), PKU-Baidu Fund 2019BD006, and PKU-Tencent Joint research Lab. Yingxia Shao's work is supported by the National Natural Science Foundation of China (No. U1936104), CCF-Baidu Open Fund, and The Fundamental Research Funds for the Central Universities 2020RC25. Yingxia Shao is the corresponding author, and the first two authors contributed equally.

Lei Chen's work is partially supported by National Key Research and Development Program of China Grant No. 2018AAA0101100, the Hong Kong RGC GRF Project 16202218, CRF Project C6030-18G, C1031-18G, C5026-18G, AOE Project AoE/E-603/18, Theme-based project TRS T41-603/20R, China NSFC No. 61729201, Guangdong Basic and Applied Basic Research Foundation 2019B151530001, Hong Kong ITC ITF grants ITS/044/18FX and ITS/470/18FX, Microsoft Research Asia Collaborative Research Grant, HKUST-NAVER/LINE AI Lab, Didi-HKUST joint research lab, HKUST-Webank joint research lab grants.

References

[1] W. Zhang, X. Miao, Y. Shao, J. Jiang, L. Chen, O. Ruas, and B. Cui, “Reliable data distillation on graph convolutional network,” in *SIGMOD*. ACM, 2020, pp. 1399–1414.

[2] H. Chen, H. Yin, T. Chen, Q. V. H. Nguyen, W. Peng, and X. Li, “Exploiting centrality information with graph convolutions for network representation learning,” in *ICDE*. IEEE, 2019, pp. 590–601.

[3] J. Hu, B. Yang, C. Guo, C. S. Jensen, and H. Xiong, “Stochastic origin-destination matrix forecasting using dual-stage graph convolutional, recurrent neural networks,” in *ICDE*. IEEE, 2020, pp. 1417–1428.

[4] H. Shi, Q. Yao, Q. Guo, Y. Li, L. Zhang, J. Ye, Y. Li, and Y. Liu, “Predicting origin-destination flow via multi-perspective graph convolutional network,” in *ICDE*. IEEE, 2020, pp. 1818–1821.

[5] Y. Jin, W. Zhang, X. He, X. Wang, and X. Wang, “Syndrome-aware herb recommendation with multi-graph convolution network,” in *ICDE*. IEEE, 2020, pp. 145–156.

[6] Z. Li, X. Shen, Y. Jiao, X. Pan, P. Zou, X. Meng, C. Yao, and J. Bu, “Hierarchical bipartite graph neural networks: Towards large-scale e-commerce applications,” in *ICDE*. IEEE, 2020, pp. 1677–1688.

[7] Y. Zheng, C. Gao, X. He, Y. Li, and D. Jin, “Price-aware recommendation with graph convolutional networks,” in *ICDE*. IEEE, 2020, pp. 133–144.

[8] J. Hu, C. Guo, B. Yang, and C. S. Jensen, “Stochastic weight completion for road networks using graph convolutional networks,” in *ICDE*. IEEE, 2019, pp. 1274–1285.

[9] H. Yuan and G. Li, "A survey of traffic prediction: from spatio-temporal data to intelligent transportation," *Data Sci. Eng.*, vol. 6, no. 1, pp. 63–85, 2021.

[10] S. Wu, Y. Zhang, C. Gao, K. Bian, and B. Cui, "GARG: anonymous recommendation of point-of-interest in mobile networks by graph convolution network," *Data Sci. Eng.*, vol. 5, no. 4, pp. 433–447, 2020.

[11] H. Zhong and H. Mei, "Learning a graph-based classifier for fault localization," *Sci. China Inf. Sci.*, vol. 63, no. 6, 2020.

[12] Q. Zhang, R. Li, and T. Chu, "Kernel semi-supervised graph embedding model for multimodal and mixmodal data," *Sci. China Inf. Sci.*, vol. 63, no. 1, p. 119204, 2020.

[13] H. Liu, S. Lu, X. Chen, and B. He, "G3: when graph neural networks meet parallel graph processing systems on gpus," *Proc. VLDB Endow.*, vol. 13, no. 12, pp. 2813–2816, 2020.

[14] D. Zhang, X. Huang, Z. Liu, J. Zhou, Z. Hu, X. Song, Z. Ge, L. Wang, Z. Zhang, and Y. Qi, "AGL: A scalable system for industrial-purpose graph machine learning," *Proc. VLDB Endow.*, vol. 13, no. 12, pp. 3125–3137, 2020.

[15] R. Zhu, K. Zhao, H. Yang, W. Lin, C. Zhou, B. Ai, Y. Li, and J. Zhou, "Aligraph: A comprehensive graph neural network platform," *Proc. VLDB Endow.*, vol. 12, no. 12, pp. 2094–2105, 2019.

[16] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *ICLR*, 2017.

[17] R. v. d. Berg, T. N. Kipf, and M. Welling, "Graph convolutional matrix completion," *arXiv preprint arXiv:1706.02263*, 2017.

[18] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, "Graph convolutional neural networks for web-scale recommender systems," in *SIGKDD*. ACM, 2018, pp. 974–983.

[19] A. Krizhevsky and G. Hinton, "Convolutional deep belief networks on cifar-10," *Unpublished manuscript*, vol. 40, no. 7, pp. 1–9, 2010.

[20] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, "Imagenet large scale visual recognition challenge," *International journal of computer vision*, vol. 115, no. 3, pp. 211–252, 2015.

[21] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *ICLR*, 2015.

[22] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016, pp. 770–778.

[23] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *NIPS*, 2017, pp. 1024–1034.

[24] K. Xu, C. Li, Y. Tian, T. Sonobe, K. Kawarabayashi, and S. Jegelka, "Representation learning on graphs with jumping knowledge networks," in *ICML*, 2018, pp. 5449–5458.

[25] G. Li, M. Müller, A. Thabet, and B. Ghanem, "Can gcns go as deep as cnns?" in *ICCV*, 2019, pp. 29–38.

[26] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *CVPR*, 2017, pp. 4700–4708.

[27] Q. Li, Z. Han, and X.-M. Wu, "Deeper insights into graph convolutional networks for semi-supervised learning," in *AAAI*, 2018.

[28] W. Chiang, X. Liu, S. Si, Y. Li, S. Bengio, and C. Hsieh, "Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks," in *SIGKDD*, A. Teredesai, V. Kumar, Y. Li, R. Rosales, E. Terzi, and G. Karypis, Eds. ACM, 2019, pp. 257–266.

[29] Y. Rong, W. Huang, T. Xu, and J. Huang, "Dropedge: Towards deep graph convolutional networks on node classification," in *ICLR*, 2019.

[30] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *CVPR*, 2015, pp. 1–9.

[31] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Trans. Neural Networks*, vol. 20, no. 1, pp. 61–80, 2009.

[32] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and locally connected networks on graphs," in *ICLR*, 2014.

[33] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *NeurIPS*, 2016, pp. 3844–3852.

[34] M. Niepert, M. Ahmed, and K. Kutzkov, "Learning convolutional neural networks for graphs," in *ICML*, 2016, pp. 2014–2023.

[35] F. Monti, D. Boscaini, J. Masci, E. Rodola, J. Svoboda, and M. M. Bronstein, "Geometric deep learning on graphs and manifolds using mixture model cnns," in *CVPR*, 2017, pp. 5115–5124.

[36] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *ICLR*, 2018.

[37] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *NeurIPS*, 2012, pp. 1106–1114.

[38] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *ECCV*. Springer, 2014, pp. 818–833.

[39] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger, "Deep networks with stochastic depth," in *ECCV*. Springer, 2016, pp. 646–661.

[40] J. Klicpera, A. Bojchevski, and S. Günnemann, "Predict then propagate: Graph neural networks meet personalized pagerank," in *ICLR*, 2019.

[41] S. Abu-El-Haija, B. Perozzi, A. Kapoor, N. Alipourfard, K. Lerman, H. Harutyunyan, G. V. Steeg, and A. Galstyan, "Mixhop: Higher-order graph convolutional architectures via sparsified neighborhood mixing," in *ICML*, vol. 97. PMLR, 2019, pp. 21–29.

[42] K. Zhang, Y. Zhu, J. Wang, and J. Zhang, "Adaptive structural fingerprints for graph attention networks," in *ICLR*, 2020.

[43] D. Chen, Y. Lin, W. Li, P. Li, J. Zhou, and X. Sun, "Measuring and relieving the over-smoothing problem for graph neural networks from the topological view," in *AAAI*. AAAI Press, 2020, pp. 3438–3445.

[44] L. Zhao and L. Akoglu, "Pairnorm: Tackling oversmoothing in gnns," in *ICLR*. OpenReview.net, 2020.

[45] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *ICML*, 2015, pp. 448–456.

[46] R. K. Srivastava, K. Greff, and J. Schmidhuber, "Highway networks," *CoRR*, vol. abs/1505.00387, 2015.

[47] H. Fang, V. Wang, and M. Yamaguchi, "Dissecting deep learning networks - visualizing mutual information," *Entropy*, vol. 20, no. 11, p. 823, 2018.

[48] H. Fang, J. Thiyagalingam, N. Bessis, and E. A. Edirisinghe, "Fast and reliable human action recognition in video sequences by sequential analysis," in *ICIP*, 2017.

[49] K. Zhou, Q. Song, X. Huang, and X. Hu, "Auto-gnn: Neural architecture search of graph neural networks," *CoRR*, vol. abs/1909.03184, 2019.

[50] Z. Zhang, X. Wang, and W. Zhu, "Automated machine learning on graphs: A survey," *CoRR*, vol. abs/2103.00742, 2021.

[51] Y. Gao, H. Yang, P. Zhang, C. Zhou, and Y. Hu, "Graphnas: Graph neural architecture search with reinforcement learning," *CoRR*, vol. abs/1904.09981, 2019.

[52] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," in *CVPR*, 2017, pp. 652–660.

[53] X. He and T.-S. Chua, "Neural factorization machines for sparse predictive analytics," in *SIGIR*. ACM, 2017, pp. 355–364.

[54] H. Guo, R. Tang, Y. Ye, Z. Li, and X. He, "Deepfm: A factorization-machine based neural network for CTR prediction," in *IJCAI*, 2017, pp. 1725–1731.

[55] X. Xin, B. Chen, X. He, D. Wang, Y. Ding, and J. Jose, "CFM: convolutional factorization machines for context-aware recommendation," in *IJCAI*, 2019, pp. 3926–3932.

[56] "Lasagne source code," https://github.com/PKU-DAIR/Lasagne.

[57] H. Zeng, H. Zhou, A. Srivastava, R. Kannan, and V. K. Prasanna, "Graphsaint: Graph sampling based inductive learning method," in *ICLR*, 2020.

[58] R. Liao, M. Brockschmidt, D. Tarlow, A. L. Gaunt, R. Urtasun, and R. Zemel, "Graph partition neural networks for semi-supervised classification," *arXiv preprint arXiv:1803.06272*, 2018.

[59] H. Pei, B. Wei, K. C. Chang, Y. Lei, and B. Yang, "Geom-gcn: Geometric graph convolutional networks," in *ICLR*, 2020.

[60] Y. Koren, R. M. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, vol. 42, no. 8, pp. 30–37, 2009.

[61] H. Wang, N. Wang, and D. Yeung, "Collaborative deep learning for recommender systems," in *SIGKDD 2015*. ACM, 2015, pp. 1235–1244.

[62] X. Wang, X. He, M. Wang, F. Feng, and T. Chua, "Neural graph collaborative filtering," in *SIGIR*. ACM, 2019, pp. 165–174.

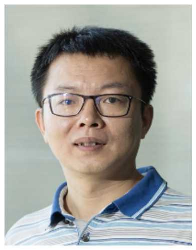
[63] X. He, K. Deng, X. Wang, Y. Li, Y. Zhang, and M. Wang, "Lightgcn: Simplifying and powering graph convolution network for recommendation," in *SIGIR 2020*. ACM, 2020, pp. 639–648.

[64] S. Abu-El-Haija, A. Kapoor, B. Perozzi, and J. Lee, "N-gcn: Multi-scale graph convolution for semi-supervised node classification," *arXiv preprint arXiv:1802.08888*, 2018.
[65] C. Zhuang and Q. Ma, "Dual graph convolutional networks for graph-based semi-supervised classification," in *WWW*, 2018, pp. 499–508.
[66] S. Luan, M. Zhao, X. Chang, and D. Precup, "Break the ceiling: Stronger multi-scale deep graph convolutional networks," in *NeurIPS*, 2019, pp. 10 943–10 953.
[67] P. Velickovic, W. Fedus, W. L. Hamilton, P. Liò, Y. Bengio, and R. D. Hjelm, "Deep graph infomax," in *ICLR*, 2019.
[68] Z. Peng, W. Huang, M. Luo, Q. Zheng, Y. Rong, T. Xu, and J. Huang, "Graph representation learning via graphical mutual information maximization," in *WWW*, 2020, pp. 259–270.
[69] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" in *ICLR*, 2019.
[70] F. Wu, A. H. S. Jr., T. Zhang, C. Fifty, T. Yu, and K. Q. Weinberger, "Simplifying graph convolutional networks," in *ICML*, 2019, pp. 6861–6871.
[71] H. Gao, Z. Wang, and S. Ji, "Large-scale learnable graph convolutional networks," in *SIGKDD*. ACM, 2018, pp. 1416–1424.
[72] J. Chen, T. Ma, and C. Xiao, "Fastgcn: Fast learning with graph convolutional networks via importance sampling," in *ICLR*, 2018.
[73] R. Albert and A.-L. Barabási, "Statistical mechanics of complex networks," *Reviews of modern physics*, vol. 74, no. 1, p. 47, 2002.
[74] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *NeurIPS 2017*, 2017, pp. 5998–6008.
[75] G. Wang, R. Ying, J. Huang, and J. Leskovec, "Direct multi-hop attention based graph neural network," *CoRR*, vol. abs/2009.14332, 2020.

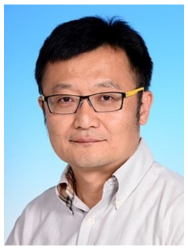
Xupeng Miao received his BSc in Computer Science and Technology from Northeastern University in 2017. Now he is a fourth year PhD candidate in the School of EECS, Peking University. His research interests include gpu acceleration, distributed deep learning system and graph analysis.

Wentao Zhang is a PhD student in Computer Science at Peking University. His research interests are Automated Machine Learning and Graph Computing.

Yingxia Shao is an associate researcher in the School of Computer Science, Beijing University of Posts and Telecommunications. His research interests include large-scale graph analysis, parallel computing framework, and knowledge graph analysis.



Bin Cui is a professor in the School of EECS and Director of Institute of Network Computing and Information Systems, at Peking University. His research interests include database systems, and data mining. Prof. Cui has published more than 100 research papers, and is the winner of Microsoft Young Professorship award (MSRA 2008), and CCF Young Scientist award (2009).



Lei Chen received the PhD degree in computer science from the University of Waterloo, Canada, in 2005. He is currently a professor with the Department of CSE, HKUST. His research inter- ests include crowdsourcing, uncertain databases, and data integration. He is an IEEE Fellow and ACM Distinguished Scientist

Jiawei Jiang is a PostDoc in Department of Computer Science at ETH Zurich. He received his Ph.D from Peking University in 2018. His interests include distributed ML, communication optimization and GBDT algorithms.

Ce Zhang is an Assistant Professor in Computer Science at ETH Zurich. His current research focuses on building data systems to support machine learning and help facilitate other sciences. He contributed to the research efforts that won the SIGMOD Best Paper Award and SIGMOD Research Highlight Award, and was featured in special issues including the Science magazine, the Communications of the ACM, "Best of VLDB", and the Nature magazine.