ScaleGCN: Efficient and Effective Graph Convolution via Channel-Wise Scale Transformation

Article *in* IEEE Transactions on Neural Networks and Learning Systems · August 2022

DOI: 10.1109/TNNLS.2022.3199390

5 authors, including:

Tianqi Zhang
Shanghai Jiao Tong University
4 PUBLICATIONS **9** CITATIONS

SEE PROFILE

Qitian Wu
Shanghai Jiao Tong University
30 PUBLICATIONS **210** CITATIONS

SEE PROFILE

Junchi Yan
Shanghai Jiao Tong University
270 PUBLICATIONS **5,961** CITATIONS

SEE PROFILE

ScaleGCN: Efficient and Effective Graph Convolution via Channel-Wise Scale Transformation

Tianqi Zhang, Qitian Wu, Junchi Yan, *Senior Member, IEEE*, Yunan Zhao, and Bing Han

***Abstract*— Graph convolutional networks (GCNs) have shown success in many graph-based applications as they can combine node features and graph topology to obtain expressive embeddings. While there exist numerous GCN variants, a typical graph convolution layer uses neighborhood aggregation and fully-connected (FC) layers to extract topological and node-wise features, respectively. However, when the receptive field of GCNs becomes larger, the tight coupling between the number of neighborhood aggregation and FC layers can increase the risk of over-fitting. Also, the FC layer between two successive aggregation operations will mix and pollute features in different channels, bringing noise and making node features hard to converge at each channel. In this article, we explore graph convolution without FC layers. We propose scale graph convolution, a new graph convolution using channel-wise scale transformation to extract node features. We provide empirical evidence that our new method has lower over-fitting risk and needs fewer layers to converge. We show from both theoretical and empirical perspectives that models with scale graph convolution have lower computational and memory costs than traditional GCN models. Experimental results on various datasets show that our method can achieve state-of-the-art results, in a cost-effective fashion.**

***Index Terms*— Graph convolutional networks (GCNs), graph neural networks (GNNs), graph representation learning, semi-supervised learning.**

I. Introduction

RECENTLY, graph convolutional networks (GCNs) [1], [2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [12] have become increasingly popular in graph-based applications. Different from traditional graph representation learning algorithms [13], [14], [15] where only the graph topology is considered, GCNs can combine node features and graph topology to obtain embeddings layer by layer. Thus, GCNs outperform traditional methods in various tasks.

Manuscript received 30 June 2021; revised 3 March 2022; accepted 14 August 2022. This work was supported in part by the National Key Research and Development Program of China under Grant 2020AAA0107600, in part by the Shanghai Municipal Science and Technology Major Project under Grant 2021SHZDZX0102, in part by the National Natural Science Foundation of China (NSFC) under Grant 61972250 and Grant 72061127003, and in part by MYbank, Ant Group. *(Corresponding author: Junchi Yan.)*

Tianqi Zhang, Qitian Wu, and Junchi Yan are with the Department of Computer Science and Engineering, and the MoE Key Laboratory of Artificial Intelligence, AI Institute, Shanghai Jiao Tong University, Shanghai 200240, China (e-mail: lygztq@sjtu.edu.cn; echo740@sjtu.edu.cn; yanjunchi@sjtu.edu.cn).

Yunan Zhao and Bing Han are with MYbank, Ant Group, Hangzhou 310063, China (e-mail: yunan.zya@mybank.cn; hanbing.hanbing@mybank.cn).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TNNLS.2022.3199390.

Digital Object Identifier 10.1109/TNNLS.2022.3199390

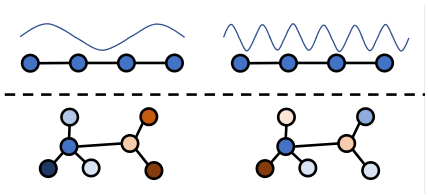


Fig. 1. Left: node features with low frequency. Models can easily utilize these features to extract common properties shared in node clusters. Right: node features containing high-frequency noises. Models are hard to extract common properties since these features change drastically in node clusters.

Despite its tremendous success, almost all GCNs are shallow networks which only exploit the information of a very limited neighborhood for each node. For example, the vanilla GCN [3] achieves the best performance with two layers in semi-supervised node classification task. This gives GCNs a small and noninformative receptive field. A larger receptive field would be desirable to provide the network with more information, especially for semi-supervised and graph-level task. However, the performance of GCNs will decay when more layers are stacked. This phenomenon is caused by both over-smoothing and over-fitting problems. Specifically, the neighborhood aggregation scheme used in GCNs is a special type of Laplacian smoothing [16], too many layers with aggregation will make node features of different node types indistinguishable, which leads to over-smoothing. On the other hand, a typical graph convolution layer uses a fully connected (FC) layer after aggregation to extract node features. When more layers are stacked to make the receptive field larger, the tight coupling between the aggregation and FC layer in one graph convolution layer will increase the risk of over-fitting [3] as the number of parameters increases with network depth. Experimental results in simplified graph convolution (SGC) [7] also show that on small graphs a network with only one linear layer can achieve competitive results compared with other GCNs.

Recently, many efforts have been put to address the over-smoothing issue. One promising direction is to find new aggregation scheme with lower over-smoothing risk. Xu *et al.* [6] [jumping knowledge network (JK-Net)] show a connection between the neighborhood aggregation in GCNs and the random walk method, which inspires a new aggregation scheme based on personalized PageRank [13] as proposed in approximate personalized propagation of neural prediction (APPNP) [11]. Klicpera *et al.* [17] (GDC) generalize APPNP by using arbitrary graph diffusion processes. GCN via

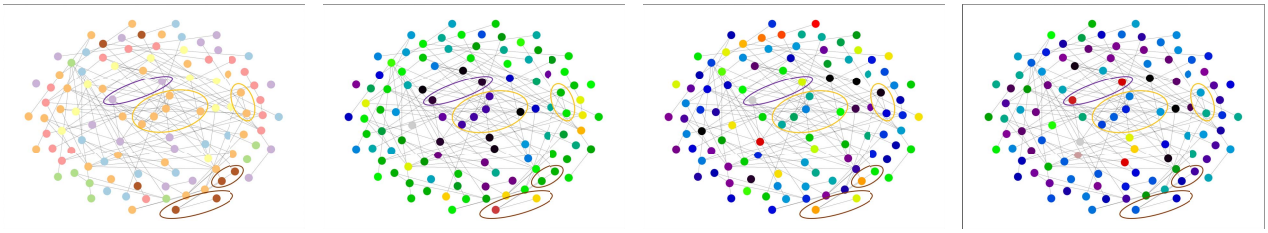


Fig. 2. Node labels and features (the first channel) of models at the last layer on a subset of the Cora dataset [18]. We partition the original graph into sub-graphs with mean size 100 and randomly sample a sub-graph for visualizing. We mark small graphs on this sub-graph where GCNII suffers severe noise. (a) Node Labels. (b) GCN. (c) GCNII. (d) ScaleGCN.

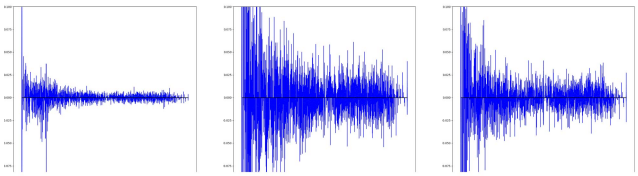


Fig. 3. Signal in the frequency domain obtained by performing Fourier transformation on node features (the first channel) of different models at the last layer on Cora dataset [18] where x-axis represents eigenvalues of the graph Laplacian as the frequency and y-axis represents features in the frequency domain as the magnitude. All models have 16 layers. (a) Vanilla GCN [3] suffers from severe over-smoothing problem that almost no useful feature remains. (b) GCNII [12] has more high-frequency noise than other models due to the channel mixing. (c) Our proposed ScaleGCN uses channel-wise scale transformation to avoid channel mixing. Compared to those of GCNII, features obtained by ScaleGCN contain less noise.

initial residual and identity mapping (GCNII) [12] extends the aggregation in APPNP to a parameterized one and builds deep nonlinear GCNs which prevents over-smoothing.

However, all these works above only try to solve the over-smoothing issue. For the over-fitting issue, all these works are either use a nonparameterized aggregation scheme and a shallow nonlinear network to extract final features, lacking expressive power, or still have tight coupling between aggregation and FC layer in their layers, which brings the risk of over-fitting. None of them tries to solve over-smoothing and over-fitting issues simultaneously and balances the reception field and the expressive power. Meanwhile, as shown in [11], node features in GCNs converge to a random walk's limit distribution as the number of layers increases. For GCNII, we further make an observation that the FC layer between two successive aggregation operations mixes features of different channels, bringing noise from other channels and making features hard to converge at each channel. As shown in Fig. 1, adjacent nodes in graphs often share some properties, which intuitively is why the message-passing mechanism works in graph representation learning models. Therefore, node features in local clusters are prone to change slightly, which corresponds to smooth signal with low frequency. On the other hand, if the graph signal contains noises, it will have drastic changes in local clusters, which corresponds to signals with high frequency. And these noises will make graph representation learning models hard to utilize shared properties in local clusters. Examples can be found in Figs. 2 and 3: first, we visualize node features of the first channel on a subgraph of the Cora dataset [18] in Fig. 2. Then we perform Fourier transformation and plot the signal in the frequency domain in Fig. 3, with eigenvalues of the graph Laplacian as frequency on the x-axis and features in the frequency domain as the magnitudes on the y-axis. From these two aspects, GCNII has more high-frequency noises since it outputs node features with intense changes. We believe these noises come from its FC layers in each graph convolution operation where features belong to different channels are mixed. We call such a phenomenon the **channel mixing** problem, which directly motivates this article. To the best of our knowledge, almost all theoretical analyses [3], [11], [12] are based on single-channel node features. The multichannel case with FC layers as the update scheme actually has no theoretical support. Since almost all GCNs use FC layer between successive aggregations to extract node features, the channel mixing problem can also bother other GCN models, as long as they adopt the same scheme as GCNII, especially for deep architectures. Although GCNII applies a decay trick where the last few FC layers have almost no effect on node features to avoid channel mixing, yet it incurs huge redundant computations and it also requires many additional layers to achieve its best performance.

To tackle these fundamental problems, we propose scale graph convolution, based on channel-wise scale transformation on node features, and models using this convolution scheme, ScaleGCN. As illustrated in Fig. 4, by replacing FC layers in graph convolution with channel-wise scale transformation, scale graph convolution prevents interchannel feature mixing while keeping the model capacity for complex tasks. The empirical study shows that ScaleGCN can achieve state-of-the-art results with fewer layers compared to GCNII on various datasets. Compared with FC layers, experiments show that models with channel-wise scale transformation have lower over-fitting risk. Moreover, ScaleGCN has lower computational and space complexity, resulting in more efficient models compared with traditional GCNs. Also, scale graph convolution itself as a parameterized aggregation can help graph neural network (GNN) models achieve better performance. Finally, by using scale graph convolution we can perform channel selection and pruning on models to achieve higher inference speed. The highlights of this article are as follows.

1) We analyze the above fundamental channel mixing problem which has not been formally studied and resolved

before, and propose our scale graph convolution based on channel-wise scale transformation to solve this problem.
2) We conduct experiments to show that scale graph convolution can prevent the channel mixing problem. Experimental results also show that ScaleGCN has less risk of over-fitting than traditional GCN models. Experimental results also show scale graph convolution can be combined into traditional GNNs as a parameterized aggregation to achieve better performance.
3) We further compare training speed and memory usage between ScaleGCN and traditional GCN models. Experimental results show that ScaleGCN is more cost-effective than traditional models in most cases. We also perform channel selection and pruning on GNNs using scale graph convolution and experimental results show this can greatly improve the inference speed.

II. Related Works

We briefly discuss the related works on addressing over-smoothing in GCNs and learning on large graphs.

A. Graph Convolutional Networks

The first work of GCNs is [1]. In their work, Bruna *et al.* show how to generalize Convolutional Neural Networks (CNNs) to graph-structured data from a spectral perspective. However, the network proposed in [1] needs to perform Fourier transformation on graphs explicitly and use a global filter, which is inefficient on large graphs. To solve these problems, Defferrard *et al.* [2] (ChebNet) propose a new network based on the K-order Chebyshev polynomials without expensive explicit Fourier transformation to make the spectral filters spatially localized and more efficient. The vanilla GCN [3] uses the first-order Chebyshev polynomials and a redefined adjacency matrix to perform graph convolution and thus is a special simplified case of ChebNet. By using the simplified first-order Chebyshev polynomials form, GCN can alleviate the problem of over-fitting on local neighborhood structures while still recovering a rich class of convolutional filter functions. However, the tight coupling between the neighborhood aggregation and node feature updating in one GCN layer still has a high risk of overfitting. Recently, many new variants of GCN have been proposed. The SGC [7] separates the aggregation and update operations of one graph convolutional layer and shrink all aggregation operations into one high-order aggregation at the beginning of the model as a preprocess. It also removes all the nonlinearities expect the last one to combine update operations into a single linear layer. Velickovic *et al.* [5] introduce attention mechanism which estimates importance of different neighbors, and uses the attention scores as weights for neighborhood aggregation.

B. Solving Over-Smoothing Problem

Different methods have been devised to address the over-smoothing problem. Residual/dense connections are used in [3] and [9] to facilitate training of deeper models by enabling the model to carry over information from the previous layer. The DropEdge [19] randomly removing out edges from the input graph to relieve the impact of over-smoothing. Node-wise attention mechanism [5] is also proposed to perform a weighted neighbor aggregation at each graph convolution layer. Graph isomorphism network (GIN) [20] also derives its update scheme from the Weisfeiler–Lehman (WL) algorithm [21] to obtain networks with more expressive power. High-order aggregation schemes are also proposed to alleviate the over-smoothing problem. Abu-El-Haija *et al.* [22] (N-GCN) try to train multiple networks which use different order adjacency matrices as the propagation matrix and concatenates results from different networks at the end of each layer. Similarly, IncepGCN [8], [19] uses multiple convolution kernels with different spatial sizes in one layer and combines these convolution results at the end. MixHop [23] uses a combination scheme to combine results from aggregation in different layers. However, the mechanisms used to combine representations from different distances in these high-order models are fixed and not learnable. Different from these methods, a recent work deep adaptive GNN (DAGNN) [24] devises a trainable parameter vector to compute scores for representations generating from various distances. However, all these methods lack theoretical support and can only alleviate the impact of over-smoothing. Different from these methods, methods using new aggregation scheme with less over-smoothing risk have shown that they can largely get rid of over-smoothing problem and can achieve the state-of-the-art result on various tasks. JK-Net [6] connects the neighborhood aggregation in GCNs and the random walk method. This connection inspires a new aggregation scheme based on personalized PageRank [13] as proposed in APPNP [11]. Klicpera *et al.* [17] (GDC) generalize APPNP by using arbitrary graph diffusion processes. GCNII [12] extends the aggregation in APPNP to a parameterized one and builds deep nonlinear GCNs which prevents over-smoothing. However, all these works above only try to solve the over-smoothing issue. For the over-fitting issue, all these works either use a nonparameterized aggregation scheme and a shallow nonlinear network to extract final features, lacking expressive power, or still have tight coupling between aggregation and FC layer in their layers, which brings the risk of over-fitting.

C. Training on Large Graphs

Training GNNs directly on large-scale graphs can be difficult due to the memory limitation. Specifically, memory in computation devices like GPU cannot keep the entire graph and the embedding of each node in it. Thus, a stochastic training scheme is necessary in such scenario. To solve this problem, two strategies can be applied: sampling and clustering. GraphSage [4] uses a neighborhood sampling method which samples neighbors of current nodes to build computation flow on graphs layer by layer. PinSage [25] extends the neighborhood sampling method in GraphSage to a random-walk-based method. FastGCN [26] adopts importance sampling which directly samples subgraphs for computation at each layer according to the importance score of nodes.

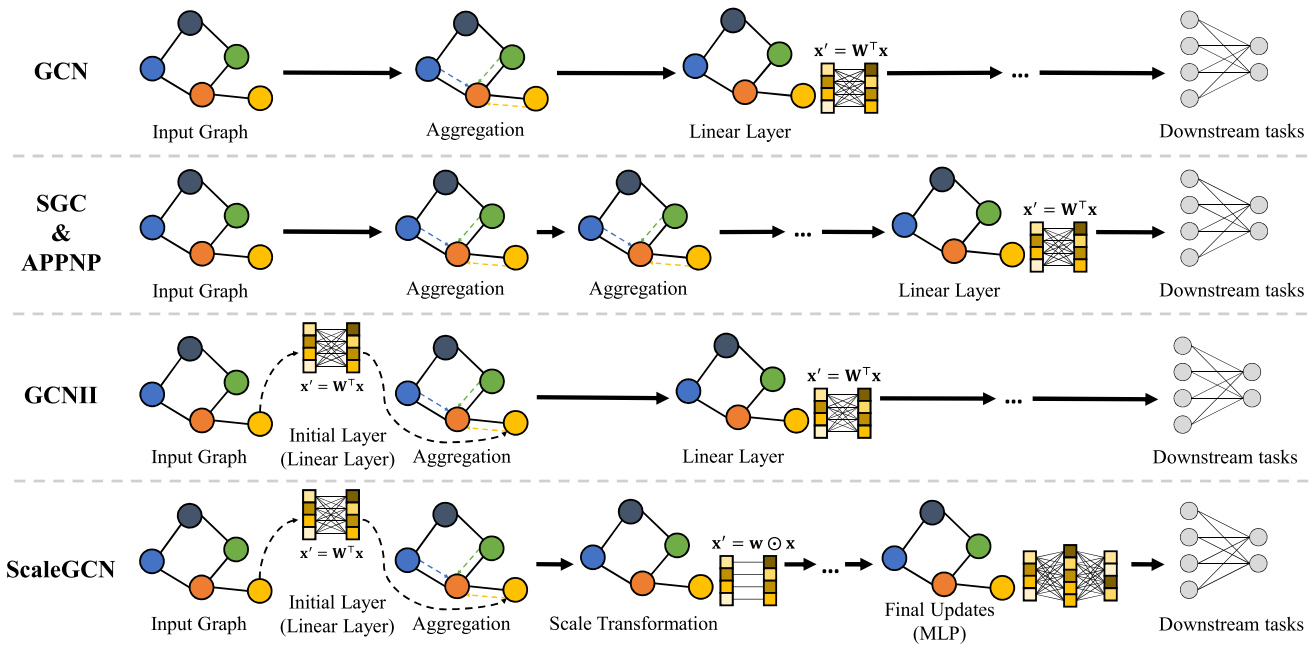


Fig. 4. Architecture comparison of our ScaleGCN and other GCN models. We omit the initial residual connection [11] and identity mapping [12] here. First row: traditional GCN uses FC layers to update node features at each layer. Second row: SGC [7] and APPNP [11] separate the aggregation and update operations of one graph convolutional layer and shrink all aggregation operations into one high-order aggregation at the beginning of the model as a preprocess. Third row: GCNII [12] extends the aggregation in APPNP to a parameterized one and builds deep nonlinear GCNs which prevents over-smoothing. Final row: ScaleGCN uses channel-wise scale transformation to update node features, being a cost-effective way of avoiding interchannel feature mixing. To change the number of feature channels, we use an FC layer at the beginning as the initial layer and a multilayer perceptron at the end as final update layers.

However, sampling methods may suffer from the neighborhood expansion problem, i.e., the number of nodes involved in computation increases exponentially as the number of layers rises. Different from these methods, ClusterGCN [10] divides graphs into several disjoint sub-graphs. At each step, it samples a block of nodes that associate with a subgraph and restricts the neighborhood search within this subgraph. Thus the clustering strategy usually has low memory requirement than the sampling strategy. Pytorch-Direct [27] exploits the cuda-managed memory which can use the large host memory for heterogeneous-device training without CPU intervention.

III. Methodology

In this section, we provide the theoretical motivation and formal definition of scale graph convolution. We first derive scale graph convolution from the spectral graph convolution. Then we combine skip-connection tricks [11], [12], [17] to propose our ScaleGCN model. Next, we propose a bipartite computation graph version of scale graph convolution layer for large graph. We also combine scale graph convolution with other traditional GNN models as a parameterized aggregation to get other variants. Finally, we show a new distributed training scheme using ScaleGCN.

A. Scale Graph Convolution

Suppose there is a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with N nodes $v_i \in \mathcal{V}$ and $|\mathcal{E}|$ edges (v_i, v_j). Denote the adjacency matrix and degree matrix of $\mathcal{G}$ as $\mathbf{A} \in \mathbb{R}^{N\times N}$ and $\mathbf{D}_{i,i} = \sum_j \mathbf{A}_{i,j}$. Without loss of generality, assume the feature of each node in $\mathcal{G}$ is scalar x_i and use $\mathbf{x} \in \mathbb{R}^N$ to denote node features of all nodes in $\mathcal{G}$. Similar to the vanilla GCN [3], we consider spectral convolutions on graphs defined as the multiplication of node features with a filter $g_{\boldsymbol{\theta}} = \mathrm{diag}(\boldsymbol{\theta})$ parameterized by $\boldsymbol{\theta} \in \mathbb{R}^N$ in the frequency domain

$$g_{\boldsymbol{\theta}} * \mathbf{x} = \mathbf{U} g_{\boldsymbol{\theta}} \mathbf{U}^\top \mathbf{x} \tag{1}$$

where $\mathbf{U}$ is the matrix of eigenvectors of the normalized graph Laplacian $\mathbf{L} = \mathbf{D}^{-1/2}(\mathbf{I}_N - \mathbf{A})\mathbf{D}^{-1/2} = \mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^\top$, $\boldsymbol{\Lambda}$ is a diagonal matrix of eigenvalues of $\mathbf{L}$. A common practice is treating $g_{\boldsymbol{\theta}}$ as a function of $\boldsymbol{\Lambda}$, i.e., $g_{\boldsymbol{\theta}}(\boldsymbol{\Lambda})$, and using a polynomial approximation to reduce computational complexity. Kipf and Welling [3] use a special first-order Chebyshev polynomials with renormalization trick to simplify the spectral graph convolution as

$$g_\theta * \mathbf{x} \approx \theta\big(\tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1/2}\big)\mathbf{x} \tag{2}$$

where the parameter of filter g is simplified to a scalar θ, $\tilde{\mathbf{A}} = \mathbf{I}_N + \mathbf{A}$ and $\tilde{\mathbf{D}}_{i,i} = \sum_j \tilde{\mathbf{A}}_{i,j}$.

We now generalize this process to multidimensional cases. Suppose we are given node features $\mathbf{X} \in \mathbb{R}^{N\times C}$ with C channels, $\mathbf{x}_i \in \mathbb{R}^C$ is the feature vector of node v_i. Different from the vanilla GCN [3] that

$$\mathbf{X}' = \big(\tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1/2}\big)\mathbf{X}\boldsymbol{\Theta} \tag{3}$$

where a parameter matrix $\boldsymbol{\Theta} \in \mathbb{R}^{C\times C_{\text{out}}}$ is used to obtain the convolution result via matrix multiplication, we use a parameter vector $\boldsymbol{\theta} \in \mathbb{R}^C$ and apply (2) on each channel of node features

$$\mathbf{X}' = \big(\tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1/2}\big)\mathbf{X} \cdot \mathrm{diag}(\boldsymbol{\theta}) \tag{4}$$

TABLE I

COMPLEXITY COMPARISON. ASSUME ALL MODELS HAVE L LAYERS AND THE AVERAGE NUMBER OF NODE FEATURE CHANNELS IS C. h DENOTES THE NUMBER OF ATTENTION HEADS IN GRAPH ATTENTION NETWORKS (GAT) [5]. N AND $|\mathcal{E}|$ DENOTES THE NUMBER OF NODES AND EDGES OF THE INPUT GRAPH, RESPECTIVELY

	Space complexity	Time complexity
GCN [3]	$O(LC^2 + LNC)$	$O(L\|\mathcal{E}\|C + LNC^2)$
GAT [5]	$O(LC^2 + hLNC)$	$O(hL\|\mathcal{E}\|C + LNC^2)$
GCNII [12]	$O(LC^2 + LNC)$	$O(L\|\mathcal{E}\|C + LNC^2)$
ScaleGCN (ours)	$O(LNC)$	$O(L\|\mathcal{E}\|C + LNC)$

where $\mathbf{X}' \in \mathbb{R}^{N \times C_{\text{out}}}$ is the result matrix. The time complexity of this operation is $O(C(|\mathcal{E}| + N))$ while the complexity of vanilla GCN [3] is $O(C(|\mathcal{E}| + NC_{\text{out}}))$. Table I lists the complexity of typical models.

B. ScaleGCN

As shown in [6], there is a relationship between the aggregation scheme used in the vanilla GCN [3] and a random walk. Vanilla GCN [3] converges to a random walk's limit distribution as the number of layers increases. As [11] points out, such a limit distribution is a property of the graph as a whole and does not take the root node into account. This is the main reason for the over-smoothing problem and such an aggregation scheme is unsuited to describe the root node's neighborhood especially on small graphs where the node feature converges to its limit in few layers. To tackle this problem, we adopt two techniques used in GCNII [12], *Initial residual connection* and *Identity mapping*, in our basic building blocks. We formally define the ScaleGCN layer as

$$\begin{aligned}\mathbf{H}' &= (1-\alpha_l)\tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1/2}\mathbf{H}^{(l)} + \alpha_l\mathbf{H}^{(0)} \\ \mathbf{H}^{(l+1)} &= \sigma\left(\mathbf{H}'\left((1-\beta_l)\mathbf{I}_N + \beta_l \text{diag}\left(\boldsymbol{\theta}^{(l)}\right)\right)\right) \end{aligned} \tag{5}$$

where $\boldsymbol{\theta}^{(l)}$ is the parameter vector at layer l, $\mathbf{H}^{(l)}$ is the hidden node feature matrix at layer l and $\mathbf{H}^{(0)}$ is obtained by an initial layer parameterized by $\boldsymbol{\theta}$, i.e., $\mathbf{H}^{(0)} = f_{\boldsymbol{\theta}}(\mathbf{X})$. α_l and β_l are two hyper-parameters to weigh the sum. We initialize all $\boldsymbol{\theta}^{(l)}$ as $\mathbf{1}$ vector, which corresponds to the aggregation scheme of APPNP with an activation function.

We illustrate the architecture of our ScaleGCN model in Fig. 4. Different from the traditional GCN architecture where each convolution layer has a linear layer, ScaleGCN use channel-wise scale transformation in each convolution layer. Thus, we cannot change the number of feature channels. To solve this problem, we add a linear layer at the beginning of our architecture as the initial layer. We also add a multilayer perceptron at the end of our architecture to change node feature channels and increase the capacity of our model.

C. Bipartite Computation Graph Version

A severe issue of the graph convolution used in GCNII (and also in APPNP) is that it requires node feature matrices of all layers having the same size, otherwise the initial residual connection technique will be inapplicable. This restriction is not a concern on small homogeneous graphs but it is unrealistic on large or heterogeneous graphs. Specifically, when training neural networks on large graphs, one cannot feed the whole graph into computation devices like GPU due to the memory limitation. Thus, mini-batch training methods that randomly sample nodes on large graphs [4], [10], [26] are more suitable in this scenario. However, in mini-batch training, nodes involved in computation at each layer may be different. To get features of nodes in current layer, we need features of their neighbors in the previous layer. The situation for heterogeneous graph is similar, nodes at different layers may have different types. In general, we can use a bipartite computation graph to describe this scenario where nodes in the previous layer are called source nodes and nodes in the current layer are called destination nodes. We use $\mathcal{V}_{\text{src}}^{(l)}$ and $\mathcal{V}_{\text{dst}}^{(l)}$ to describe these two disjoint node sets at layer l.

To apply scale graph convolution on large graphs, we devise a variant of ScaleGCN layer, called Bipartite ScaleGCN layer (BiScaleGCN layer), for the bipartite computation graph scenario. We change the initial residual connection part of ScaleGCN layer to a parameterized residual connection from features of destination nodes to results

$$\mathbf{H}_{\text{src}}^{(l+1)} = \sigma\left(\mathbf{P}^{(l)}\mathbf{H}_{\text{src}}^{(l)}\text{diag}\left(\boldsymbol{\theta}_{\text{src}}^{(l)}\right) + \mathbf{H}_{\text{dst}}^{(l)}\text{diag}\left(\boldsymbol{\theta}_{\text{dst}}^{(l)}\right)\right) \tag{6}$$

where $\mathbf{P}^{(l)} \in \mathbb{R}^{|\mathcal{V}_{\text{dst}}^{(l)}| \times |\mathcal{V}_{\text{src}}^{(l)}|}$ is the propagation matrix at layer l which satisfies that $\mathbf{P}_{i,j}^{(l)} = 1$ if and only if there is a link from node $v_i \in \mathcal{V}_{\text{src}}^{(l)}$ to $v_j \in \mathcal{V}_{\text{dst}}^{(l)}$, $\mathbf{H}_{\text{src}}^{(l)} \in \mathbb{R}^{|\mathcal{V}_{\text{src}}^{(l)}| \times C}$ and $\mathbf{H}_{\text{dst}}^{(l)} \in \mathbb{R}^{|\mathcal{V}_{\text{dst}}^{(l)}| \times C}$ are feature matrices of source nodes and destination nodes at layer l, respectively. Finally, the result features at layer l will be used as features of source nodes at layer $l+1$.

On homogeneous large graphs, to get $\mathbf{H}_{\text{dst}}^{(l)}$ we simply set $\mathcal{V}_{\text{src}}^{(l)} \leftarrow \mathcal{V}_{\text{src}}^{(l)} \cup \mathcal{V}_{\text{dst}}^{(l)}$. This is equivalent to adding a self-loop with learnable parameters to the original graph.

D. Scale Graph Convolution as Parameterized Aggregation

The proposed scale graph convolution can also be combined with other GNN models as a parameterized aggregation to help improve performance. To show this, in this section, we choose APPNP [11] and SGC [7] as base models and derive variants from these two models since these two models have similar structure to ScaleGCN where stacked aggregations are used to extract node features. Specifically, we replace normal aggregations in these models with scale graph convolutions and get two new models called *ScaleAPPNP* and *ScaleSGC*. Moreover, we find that when combined with scale graph convolution, models without skip-connections, e.g., SGC, can be used to perform channel selection and pruning. Specifically, we can pruning feature channels with small-scale graph convolution scales after training and preserve the accuracy. After pruning, models can also be further trained to alleviate the loss of accuracy due to pruning. By pruning unessential channels out, we can greatly improve the inference speed.

1) ScaleAPPNP: APPNP borrows ideas from personalized PageRank to get a new aggregation scheme which can make

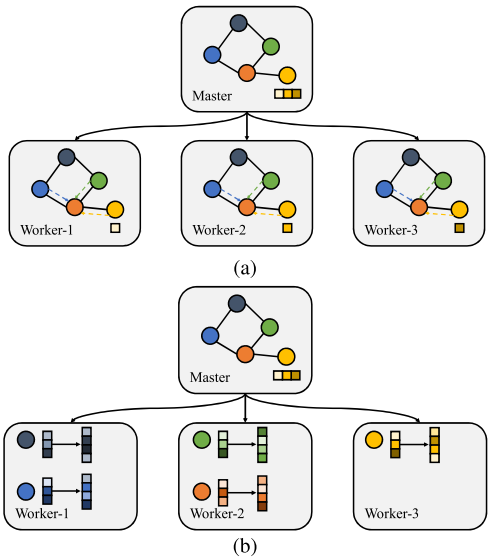


Fig. 5. Architecture of channel-wise distributed training scheme. (a) Aggregation. (b) Update features.

node features converge to a fix point

$$\begin{aligned}\mathbf{H}^{(0)} &= \mathbf{X}\\ \mathbf{H}^{(l+1)} &= (1-\alpha)\mathbf{A}\mathbf{H}^{(l)} + \alpha\mathbf{H}^{(0)}.\end{aligned} \tag{7}$$

We replace aggregation in APPNP with scale graph convolution as

$$\begin{aligned}\mathbf{H}^{(0)} &= \mathbf{X}\\ \mathbf{H}^{(l+1)} &= (1-\alpha)\mathbf{A}\mathbf{H}^{(l)} \cdot \mathrm{diag}\left(\boldsymbol{\theta}^{(l)}\right) + \alpha\mathbf{H}^{(0)}.\end{aligned} \tag{8}$$

2) ScaleSGC: SGC removes all update operations from graph convolution layers to form a high-order aggregation matrix $\mathbf{A}^K$, this can be viewed as stacked aggregations

$$\mathbf{H} = \mathbf{A}^K\mathbf{X} = \underbrace{\mathbf{A}, \ldots, \mathbf{A}}_{K}\mathbf{X}. \tag{9}$$

Therefore, we can replace all these aggregations with scale graph convolution, where $\mathbf{H} = \mathbf{H}^{(K)}$

$$\begin{aligned}\mathbf{H}^{(0)} &= \mathbf{X}\\ \mathbf{H}^{(l+1)} &= \mathbf{A}\mathbf{H}^{(l)}\mathrm{diag}\left(\boldsymbol{\theta}^{(l)}\right).\end{aligned} \tag{10}$$

Similar to ScaleGCN, FC layers are added at the end of these models to change node feature size and increase the capacity of these models.

E. Channel-Wise Distributed Training

When training GNNs on super-large graphs, a common practice is to exploit the power of distributed computation. In the context of this section, we will assume a cluster setting where a graph of machines form a cluster and we only discuss data parallelism methods here. Current methods usually divide a large graph into sub-graphs and each machine in a cluster is responsible for one sub-graph. Here, we discuss a new channel-wise distributed training method which divides node features by channel instead of dividing input graphs. And we show that ScaleGCN is much more suitable for such training scheme than traditional GCNs with linear layers as the update method. As shown in Fig. 5, suppose we have a cluster consists of three worker and one master. The computation of message-passing neural networks can be divided into two kinds of steps: aggregation and feature update. For aggregation, as shown in Fig. 5(a), we can partition training data by the node feature channel and each worker is responsible for the aggregation operation of channels assigned to it. And for node feature update, since the graph topology is not needed, we can directly divide nodes in graphs into disjoint graphs and assign each worker with one graph of nodes. For the gradient/activation aggregation[1] process, it depends on the task. For node classification, since there is no global pooling operation in the computation, losses of each node can be directly computed by each worker at the feature update step. No more additional steps are needed, except for some data transmission caused by the switch of aggregation and feature update steps. For graph-level tasks, the reduce operation of the global pooling process usually is commutative. Thus, an additional AllReduce operation is needed.

The design of traditional GCNs, where linear layers between successive aggregations are used to update node features, determines that it is not suitable for such distributed training methods, because the interleaved aggregation and linear layers will cause frequent switch between aggregation and feature update steps in the channel-wise distributed training method, resulting in high data transmission overhead. However, the same problem does not exist in ScaleGCN since there are only two switch between aggregation and feature update in ScaleGCN (one from initial layer to successive scale graph convolutions and another from scale graph convolutions to final updates). Since the scale graph convolution does not have interchannel computation, no switch is needed.

TABLE II
STATISTICS OF NODE CLASSIFICATION DATASETS USED IN THE ARTICLE

Dataset	Nodes #	Edges #	Train/Dev/Test Nodes #
CiteSeer	3,327	4,732	120/500/1,000
Cora	2,708	5,429	140/500/1,000
PubMed	19,717	44,338	60/500/1,000
Reddit	232,965	11,606,919	153,431/23,831/55,703
PPI	56,944	818,716	44,906/6,514/5,524
ogbn-products	2,449,029	61,859,140	196,615/39,323/2,213,091

IV. ANALYSIS

A. Connection to APPNP

In this section, we show that ScaleGCN can be viewed as a parameterized version of APPNP under certain conditions.

Denote $\mathbf{h}_i^{(l)} \in \mathbb{R}^N$ the ith channel of hidden node features at layer l, we can write the scale graph convolution as

$$\mathbf{h}_i^{(l+1)} = \sigma\left(\left(1-\beta_l+\beta_l\theta_i^{(l)}\right)\left((1-\alpha_l)\mathbf{P}\mathbf{h}_i^{(l)} + \alpha_l\mathbf{h}_i^{(0)}\right)\right) \tag{11}$$

where $\theta_i^{(l)}$ is the ith element of $\boldsymbol{\theta}^{(l)}$, $\mathbf{P} = \tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1/2}$ is the propagation matrix. Without loss of generality, we assume all

[1]Not the neighborhood aggregation on graphs.

TABLE III

COMPARISON OF CLASSIFICATION ACCURACY (%) ON CITATION NETWORKS WITH VARIOUS DEPTHS. THE RESULTS OF GCN, JKNET, AND INCEPGCN ARE DIRECTLY FROM [12]. FOR OTHER METHODS, WE CONDUCT EACH EXPERIMENT TEN TIMES AND REPORT THE MEAN ACCURACY. "ND" MEANS WE DO NOT APPLY DECAY ON β USED BY IDENTITY MAPPING TECHNIQUE DESCRIBED IN SECTION III-B. NOTE THAT * DENOTES THE VARIANT MODEL AS DESCRIBED IN [12]

Dataset	Method	Number of layers					
		2	4	8	16	32	64
Cora	GCN	81.1	80.4	69.5	69.4	60.3	28.7
	JKNet	-	80.2	80.7	80.2	81.1	71.5
	Incep	-	77.6	76.5	81.7	81.7	80.0
	GCNII	82.4	82.5	84.1	84.6	**85.2**	85.0
	GCNII_{ND}	83.4	84.4	84.8	84.3	84.1	84.3
	GCNII*	80.8	82.5	83.3	84.5	84.9	85.1
	$\text{GCNII*}_{\text{ND}}$	82.6	82.8	82.2	81.5	80.9	81.0
	ScaleGCN	**83.8**	84.4	84.4	84.5	**85.2**	**85.4**
	$\text{ScaleGCN}_{\text{ND}}$	**83.8**	**84.6**	**84.9**	84.6	84.4	84.5
	ScaleGCN*	82.7	84.0	84.5	**84.9**	**85.2**	85.1
	$\text{ScaleGCN*}_{\text{ND}}$	83.6	**84.6**	84.8	84.7	84.1	84.2
CiteSeer	GCN	70.8	67.6	30.2	18.3	25.0	20.0
	JKNet	-	68.7	67.7	69.8	68.2	63.4
	Incep	-	69.3	68.4	70.2	68.0	67.5
	GCNII	67.4	68.3	70.9	72.4	73.2	72.8
	GCNII_{ND}	71.6	71.2	70.7	71.3	71.3	71.3
	GCNII*	67.3	67.6	69.5	71.9	72.5	72.8
	$\text{GCNII*}_{\text{ND}}$	70.8	70.1	69.6	68.6	69.0	69.0
	ScaleGCN	**72.3**	**73.1**	**73.2**	73.1	72.7	72.6
	$\text{ScaleGCN}_{\text{ND}}$	**72.3**	72.7	**73.2**	**73.3**	**73.3**	**73.3**
	ScaleGCN*	72.1	72.5	72.8	73.2	72.9	72.6
	$\text{ScaleGCN*}_{\text{ND}}$	72.2	72.5	72.8	73.1	73.2	**73.3**
PubMed	GCN	79.0	76.5	61.2	40.9	22.4	35.3
	JKNet	-	78.0	78.1	72.6	72.4	74.5
	Incep	-	77.7	77.9	74.9	OOM	OOM
	GCNII	79.0	78.9	79.6	**79.7**	79.7	79.6
	GCNII_{ND}	79.0	79.1	79.4	79.5	79.7	79.6
	GCNII*	78.2	79.1	79.3	79.6	**79.8**	79.7
	$\text{GCNII*}_{\text{ND}}$	78.4	79.1	79.4	78.9	78.8	78.9
	ScaleGCN	79.2	**79.3**	79.4	79.6	79.6	79.7
	$\text{ScaleGCN}_{\text{ND}}$	**79.3**	79.2	**79.7**	**79.7**	79.5	79.0
	ScaleGCN*	78.5	**79.3**	79.5	79.5	**79.8**	**79.8**
	$\text{ScaleGCN*}_{\text{ND}}$	79.1	79.2	**79.7**	79.5	79.4	79.0

TABLE IV

MEMORY USAGE, TRAINING TIME PER EPOCH, AND CLASSIFICATION ACCURACY ON CITATION DATASETS WITH EIGHT-LAYER MODELS

		Cora	CiteSeer	PubMed
Memory (M)	GCNII	18.9	53.2	56.7
	ScaleGCN	**18.4**	**52.7**	**56.2**
Time/epoch (ms)	GCNII	19.8	21.7	24.8
	ScaleGCN	**18.7**	**21.6**	**24.6**
Accuracy (%)	GCNII	84.13±0.24	70.90±0.38	**79.57±0.14**
	ScaleGCN	**84.44±0.21**	**73.21±0.30**	79.42±0.17

layers share the same α and β and let $w_i^{(l)} = 1 - \beta + \beta\theta_i^{(l)}$. Also, since we use ReLU as the activation function of all scale graph convolution layers and the initial layer, all hidden node features in ScaleGCN are nonnegative vectors. We further assume $w_i^{(l)} \geq 0$. Thus, we can remove the ReLU function σ

$$\mathbf{h}_i^{(l+1)} = w_i^{(l)}\left((1-\alpha)\mathbf{P}\mathbf{h}_i^{(l)} + \alpha\mathbf{h}_i^{(0)}\right) \tag{12}$$

which is a generalized APPNP aggregation scheme with a weight parameter.

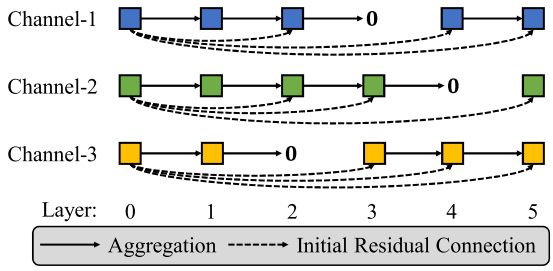


Fig. 6. If the weight $w_i^{(l)} = \beta_l + (1-\beta_l)\theta_i^{(l)}$ of channel i is nonpositive, ScaleGCN will set node features of this channel to zero in the next layer. Thus, the network will learn new node features from initial node features.

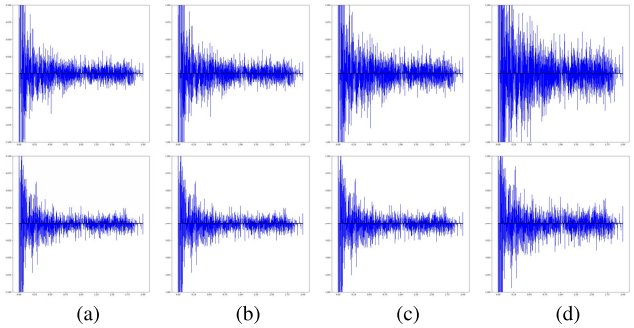


Fig. 7. Spectral coefficients of node features (first channel) at the last few layers on Cora. Top: spectral coefficients in GCNII. Bottom: spectral coefficients in ScaleGCN. Both models have 16 layers without decay on β. GCNII has more high-frequency noise at the last few layers and being more difficult to converge due to the *channel mixing* problem (see Section I). (a) Layer-10. (b) Layer-12. (c) Layer-14. (d) Layer-16.

B. Connection to NAS

Network architecture search (NAS) aims to automatically learn a network topology that can achieve best performance on a certain task. In this section, we show that ScaleGCN can automatically learn a global aggregation architecture at the node feature channel level in an end-to-end manner.

At layer l, if the weight $w_i^{(l)} = 1 - \beta_l + \beta_l\theta_i^{(l)}$ of channel i is nonpositive, then its node features will be set to zero

$$\mathbf{h}_i^{(l+1)} = \sigma\left(w_i^{(l)}\left((1-\alpha_l)\mathbf{P}\mathbf{h}_i^{(l)} + \alpha_l\mathbf{h}_i^{(0)}\right)\right) = \mathbf{0}. \tag{13}$$

Then at the next layer, since we have $\mathbf{h}_i^{(l+1)} = \mathbf{0}$, it will extract new node features directly from the initial node features

$$\mathbf{h}_i^{(l+2)} = \sigma\left(w_i^{(l+1)}\alpha_{l+1}\mathbf{h}_i^{(0)}\right). \tag{14}$$

From a global perspective, this gives ScaleGCN the ability to learn a proper number of aggregations for each node feature channel, as illustrated in Fig. 6.

V. EXPERIMENTS

In this section, we compare models using scale graph convolution with other state-of-the-art methods on various learning tasks. Source code will be made public available with the publication of this article.

We conduct our experiments on datasets corresponding to four graph learning scenarios: 1) semi-supervised node

TABLE V

COMPARISON OF MEMORY USAGE, TRAINING TIME PER EPOCH, AND CLASSIFICATION ACCURACY ON REDDIT. NUMBERS IN THE BRACKETS INDICATE THE NUMBER OF HIDDEN FEATURE CHANNELS USED IN THE MODEL. TWO VERSIONS OF MODELS WITH TWO LAYERS AND FOUR LAYERS ARE EVALUATED

Reddit		ClusterGCN(128)	GraphSage(128)	GraphSage(256)	ScaleGCN(128)
2-Layer	Memory(M)	1006	2246	2248	**745**
	Time/epoch(s)	5.36	5.11	5.22	**5.2**
	Accuracy(%)	95.73±0.07	95.51±0.13	95.40±0.15	**96.09±0.08**
4-Layer	Memory(M)	1007	5811	5814	**746**
	Time/epoch(s)	6.64	33.11	34.57	**5.25**
	Accuracy(%)	95.66±0.12	94.68±0.16	94.54±0.13	**96.06±0.11**

classification on small graphs; 2) semi-supervised node classification on large graphs; 3) inductive node classification across graphs; and 4) graph classification.

For node classification on small graphs, we use three standard citation network datasets Cora, CiteSeer, and PubMed [18]. In these datasets, each node corresponds to a paper and each edge corresponds to a citation between two papers. The node features in all these datasets are bag-of-words embeddings. For the node classification on large graphs, we use the Reddit post dataset [4], which is a large-scale social networks dataset, and ogbn-products dataset [28], which represents an Amazon product co-purchasing network. Each node in Reddit is a post and two nodes are linked by an edge if the same user comments both of them. Nodes in ogbn-products represent products sold in Amazon, and edges between two products indicate that the products are purchased together. To show our method can generalize across graphs, we also conduct experiments on inductive node classification tasks. We evaluate ScaleGCN on the protein–protein interactions (PPI) [4], which is a multigraph dataset with 24 graphs. We follow the setting of [4] which uses 20 graphs for training, two for validation, and two for testing. For graph classification, we select four datasets (D&D, PROTEINS, NCI1, and NCI109) with a large number of graphs from the TUDataset[2] benchmark datasets [29]. Table II summarizes the statistics of node classification datasets. All experiments are conducted on a machine with a GeForce RTX 2080 Ti GPU and a Intel i7-7820X CPU.

A. Node Classification on Small Graphs

1) Settings and Baselines: For semi-supervised node classification on small graphs, we apply the standard fixed dataset split [18] on citation networks with 20 nodes per class for training, 500 nodes for validation, and 1000 nodes for testing. We choose GCN [3], JKNet [6], IncepGCN [8], and GCNII (also its variant GCNII*) [12] as baselines. To show that GCNII suffers from the channel mixing problem, we also conduct experiments for GCNII and ScaleGCN without the decay of the weight matrix's parameter β. We use the Adam optimizer with a learning rate of 0.01 and early stopping with a patience of 100 epochs to train GCNII and ScaleGCN. The α for initial residual connection is set to be 0.1 and $\beta_l = \log(\lambda/l + 1)$ with $\lambda = 0.1$. We apply dropout on node features with dropout rate 0.5 at each layer.

We conduct experiments with model depth from 2 to 64 and all models have 64 hidden channels. We use a single linear layer as the final update layer for ScaleGCN. We apply L_2 regularization with a rate of 0.01 for linear layers at the beginning and end of networks and a rate of 0.0005 for all graph convolution layers. For traditional models like GCN, JKNet[3] and IncepGCN, we directly reuse the best-reported results in the GCNII paper [12]. For other experiments conducted in GCNII and ScaleGCN, we stick to the above settings and do not perform any fine-tuning or hyper-parameter searching to get a fair comparison, which is different from the fine-tuned experimental results reported in [12].

2) Results: We report the mean classification accuracy after ten runs in Table III. We reuse the results already reported in [12] for GCN, JKNet, and IncepGCN. From the result, one can see that ScaleGCN can achieve state-of-the-art performance with fewer layers than GCNII. Notably, removing the decay of β does not have a negative effect on the performance of ScaleGCN (actually removing decay has a positive effect on some datasets like CiteSeer). These phenomena show that ScaleGCN can converge with fewer layers than GCNII and adding more layers without decay does not cause a decline in accuracy. It also means that ScaleGCN can alleviate the severe channel mixing problem in GCNII. Moreover, we plot spectral coefficients of the first node feature obtained via GCNII and ScaleGCN at different layers on Cora in Fig. 7. The result shows that GCNII has more noise with high frequency at last few layers and thus is hard to converge due to the channel mixing problem. We also compare the training time per epoch and the maximum memory usage of GCNII and ScaleGCN. We use eight layers for both models and report the results in Table IV. Although the difference is not large in small graphs, the result shows that ScaleGCN is more efficient than GCNII with the same number of layers.

B. Node Classification on Large Graphs

1) Settings and Baselines: For node classification on large graphs, we follow the split of Reddit dataset in [4]. We choose ClusterGCN [10] and GraphSage [4] as baselines, which represent two strategies for training on large graphs: clustering

[2]Statistics of these graph classification datasets can be found in the website at Benchmark Datasets for Graph Kernels.

[3]Actually, the JKNet implementation in the GCNII paper [12] is from the DropEdge paper [19] and is different from the original implementation in [6].

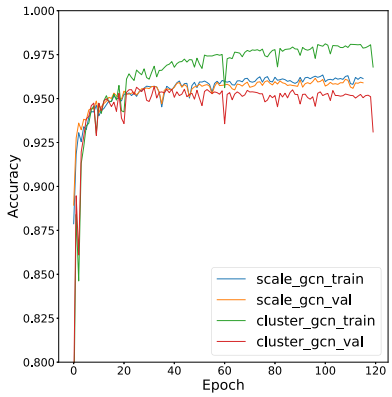


Fig. 8. Test accuracy over training epochs of ScaleGCN and ClusterGCN [10] on Reddit. Models have 128 hidden channels and four layers.

and sampling. For ScaleGCN we use the clustering strategy which divides the original graph into several subgraphs and uses mini-batch training on these sub-graphs. The layer use in ScaleGCN here is the BiScaleGCN layer. We also use Adam optimizer with a learning rate 0.01 and early stopping with patience of 20 epochs on Reddit dataset. The dropout rate for each layer is 0.2. For ScaleGCN we use a single linear layer as the final update layer. To enlarge the capacity of ScaleGCN on Reddit, we use leaky ReLU with a negative slope of 0.1 as the activation function and add a bias for each layers. We also do experiments on the ogbn-product dataset. We replace normal aggregations in SGC [7] and GraphSage [4] (marked as sample and aggregate (SAGE) in our experiments) with scale graph convolution to get ScaleSGC and ScaleSAGE models. Both ScaleSGC and ScaleSAGE used in our experiments have three scale graph convolutions as aggregations. The dropout rate is set to be 0.5 and the learning rate is 0.005.

2) Results: We report the maximum memory usage, training time per epoch, and classification accuracy on Reddit in Table V. We conduct each experiment five times and report the mean result. The result shows that ScaleGCN can beat other baselines on classification accuracy with lower memory usage and less training time. Also, since ScaleGCN has fewer parameters than normal GCN, it has less risk of over-fitting. To show this, we plot the accuracy curve in Fig. 8. The accuracy curve shows that ScaleGCN has smaller gap between accuracy on training and validation set compared with ClusterGCN. For the ogbn-products dataset, we report classification accuracy results in Table VI. Similarly, we conduct each experiment five times and report the mean result. The result shows that scale graph convolution can benefit performance of GNNs when used as a parameterized aggregation in these models.

C. Inductive Node Classification

For the inductive learning task, we use a four-layer ScaleGCN with three final update layers and the number of hidden channels is 2048. All graph convolution layers in ScaleGCN here are BiScaleGCN layer. We choose ClusterGCN and GCNII as baselines and use the suggested settings according to their papers. Specifically, we use a five-layer ClusterGCN and a nine-layer GCNIII, where the number of hidden channels is 2048. We train ScaleGCN for 8000 epochs with patience of 1000 epochs for early stop, learning rate $5e{-}4$, and dropout rate 0.2 for each layer. We conduct each experiment for five times and show the maximum memory usage, training time epoch, and classification accuracy in Table VII. The results show that ScaleGCN can achieve competitive results with the lowest memory usage and the least training time.

TABLE VI

SUMMARY OF NODE CLASSIFICATION ACCURACY RESULTS ON THE OGBN-PRODUCTS DATASET

Methods	Accuracy (%)
SGC	61.9 ± 0.14
ScaleSGC	68.0 ± 0.12
SAGE	77.9 ± 0.35
ScaleSAGE	79.0 ± 0.11

TABLE VII

SUMMARY OF MEMORY USAGE, TRAINING TIME PER EPOCH, AND NODE CLASSIFICATION ACCURACY RESULTS ON PPI

Methods	Memory(M)	Time/epoch(s)	Accuracy(%)
ClusterGCN	3822	2.43	98.64±0.12
GCNII	3993	4.36	**99.52±0.09**
ScaleGCN	**3508**	**1.36**	99.33±0.11

TABLE VIII

ACCURACY, EVALUATION TIME PER GRAPH, AND RELATIVE SPEEDUP OF SCALESGC ON CITATION NETWORKS AND OGBN-PRODUCTS BEFORE/AFTER PRUNING. NUMBER OF CHANNELS ARE IN PARENTHESES

Dataset	Model	Accuracy (%)	Time (ms)	Relative Speedup
Cora	Original (1433)	81.2	2.55	–
	Pruned (287)	81.2	2.22	1.15
CiteSeer	Original (3703)	70.4	5.66	–
	Pruned (741)	70.6	2.28	2.48
PubMed	Original (500)	80.1	4.86	–
	Pruned (160)	79.9	2.46	1.98
ogbn-products	Original (100)	68.0	7975.57	–
	Pruned (60)	64.3	4917.05	1.62

D. Channel Selection and Pruning

Since weights in the scale graph convolution are directly applied on certain channels of input features, the scales of these weights also represent the importance of certain channels in models with scale graph convolution. By training models with scale graph convolution, we are asking these models to perform a channel selecting operation. Based on this discussion, models without skip-connections like SGC, when combined with scale graph convolution, should be a good choice to test this channel selection process. After training, we can fix weights in scale graph convolutions and prune weights with small absolute values. By doing this, we are actually applying a structural weight pruning in these models.

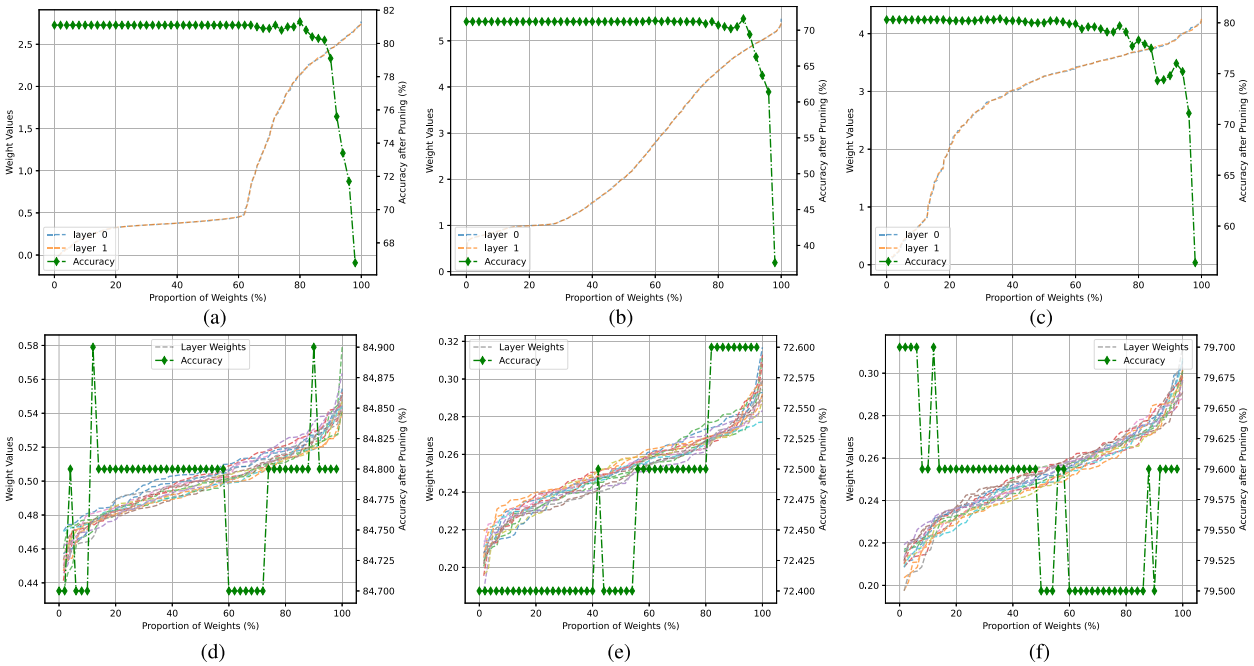


Fig. 9. Weight values and accuracy after weight pruning under different proportions of the total number of weights. We conduct this experiment on citation datasets [18] with ScaleGCN and SGC using scale graph convolution (i.e., ScaleSGC). We sort weights of each layer by their values to get the proportion of weights less than a certain threshold. We also sort and prune weights of each layer with different proportions, and after weight pruning we retrain models for ten epochs to get the accuracy after pruning. (a) ScaleSGC-Cora. (b) ScaleSGC-Citeseer. (c) ScaleSGC-Pubmed. (d) ScaleGCN-Cora. (e) ScaleGCN-Citeseer. (f) ScaleGCN-Pubmed.

To test whether the above technique is applicable on models with scale graph convolution, we replace normal aggregation in SGC with scale graph convolutions (i.e., ScaleSGC) and conduct experiments on the citation network datasets. To find out whether this method is also suitable for models with skip-connections, we conduct this experiment on ScaleGCN. Specifically, we want to see how the accuracy is affected as we pruning more and more weights out. We also want to visualize the distribution of weights' values. Therefore, we first sort weights for each layer according to their values and plot these sorted weights to as a curve to show the proportion of weights less than a certain threshold. After that, we prune these sorted weights of each layer and use the proportion of pruned weights as a measure of pruning strength.

We use a ScaleGCN with 16 layers and a ScaleSGC with two scale graph convolution layers and a final update layers. The dimension of hidden layers in ScaleGCN is set to 64. After pruning, we fix weights in the scale graph convolutions and retrain the final update layers for ten epochs with learning rate equals $1e{-}5$. The experimental results are shown in Fig. 9. We plot both the distribution of weight values and accuracy after pruning in a single figure for each method-dataset combination. We first discuss the results of ScaleSGC. On Cora and Citeseer, the accuracy after pruning starts to drop when about 80% weights are pruned. On Pubmed, the proportion is about 70%. The accuracy on Cora and Citeseer even increase when most of the weights are pruned in ScaleSGC (80% on Cora and 87% on Citeseer), we believe appropriate pruning on these two datasets can decrease the model complexity and reduce the overall structural risk. For the results of ScaleGCN, the accuracy after pruning does not change much. We think this phenomenon is caused by skip connections in ScaleGCN.

To show this pruning can help increasing inference speed of models, we also conduct experiments on citation networks and ogbn-products dataset with ScaleSGC to see the speedup before and after pruning. Since the ogbn-products is a large-scale dataset that cannot fit into our GPU device, we use its inference time on CPU instead. We summarize accuracy, inference time per graph, and relative speedup in Table VIII. We conduct experiment with a two-layer ScaleSGC on the citation networks and a three-layer ScaleSGC on the ogbn-products. We prune 80% weights on Cora and CiteSeer, 68% weights on PubMed, and 40% weights on ogbn-products. On small datasets like the citation networks, ScaleSGC can prune most of the weights without damaging the accuracy. On large datasets like ogbn-products, ScaleSGC can greatly reduce the inference time while maintaining accuracy in a certain level.

E. Scale Graph Convolution as Parameterized Aggregation

To show the proposed scale graph convolution can benefit graph representation learning models universally, we replace the normal aggregation in various GNNs with scale graph convolution and compare its performance with original networks. Notably, for the traditional GCN model with interleaving aggregations and updates, we only replace the normal aggregation with the parameterized version used in scale graph convolution and left the FC update layers unchanged. Apart from normal GCNs, we also conduct experiments on SGC, APPNP, and JKNet where stacked aggregations are used without FC update layers to see whether scale graph convolution can benefit these models. Since these models handle aggregation and update in a similar way to ours, they do

TABLE IX

CLASSIFICATION ACCURACY ON CORA WITH VARIOUS DEPTHS USING GNN MODELS WITH AND WITHOUT SCALE GRAPH CONVOLUTION

Methods	Number of Layers					
	2	4	8	16	32	64
GCN	81.1	80.4	69.5	69.4	60.3	28.7
GCN_{scale}	82.3	82.3	81.1	76.5	75.5	74.4
SGC	80.4	80.5	80.3	80.5	77.2	72.9
ScaleSGC	81.2	81.0	80.7	80.5	76.9	74.4
APPNP	82.3	83.3	83.6	83.2	83.3	83.2
ScaleAPPNP	82.3	83.4	83.6	83.4	83.3	83.3
JKNet	79.6	80.0	80.2	81.1	81.5	81.0
ScaleJKNet	81.1	81.4	81.1	81.2	81.4	81.1
ScaleGCN	83.8	84.4	84.4	84.5	85.2	85.4

not suffer from the channel-mixing problem. Therefore, we can show the performance of scale graph convolution on models without the channel-mixing problem through these experiments. Furthermore, since both over-smoothing and channel-mixing problem have strong relationship to the number of aggregations, we conduct experiments on models with various depths to understand how this factor affects the final results. We use the Cora dataset for our experiments and the results are summarized in Table IX. It can be shown from the experimental results that the scale graph convolution can still achieve performance gain without other tricks. Also, when we replace the normal aggregation with scale graph convolution as the parameterized version in models with stacked aggregations, we still achieve performance gain. Therefore, the scale graph convolution can also benefit models without channel-mixing problem. Actually, from Section IV-A we have already known that ScaleGCN can be viewed as a parameterized version of APPNP. Additionally, over-smoothing is a notorious problem in traditional GNNs like GCN and SGC, but results of GCN and SGC show that our method can also mitigate effects of over-smoothing.

F. Skip-Connection Weights and Weight Constrained Training

To show how weights of skip-connections, i.e., α and β, in ScaleGCN can affect the final performance, we conduct grid search on these two hyper-parameters and visualize the accuracy results corresponding to different α–β combinations. We also want to see the difference between distributions of accuracy obtained by models with and without decay on β, so we conduct experiments on both of these two kinds of models. Apart from these, we also find that sometimes the weights of scale graph convolutions all tend to be zero during the training process and this will damage the accuracy result. To avoid this, we want to add a constraint to the model to change the optimization task to

$$\begin{aligned} &\min_{\mathbf{w}} \ \mathcal{L}(\mathcal{D}, \mathbf{w}_{\text{scale}}, \mathbf{w}_{\text{other}}) \\ &\text{s.t. } \mathbf{w}_{\text{scale}} \geq \sigma \text{ and } (\sigma > 0) \end{aligned} \tag{15}$$

where $\mathcal{L}$ is the loss function, $\mathcal{D}$ is the training set, $\mathbf{w}_{\text{scale}}$ is the vector of weights in scale graph convolutions, $\mathbf{w}_{\text{other}}$ is the vector of weights for the rest of the model, $\mathbf{w} = \{\mathbf{w}_{\text{scale}}, \mathbf{w}_{\text{other}}\}$, and σ is a positive constant parameter. We can use the Lagrange multiplier method to get a new loss function that incorporates this constraint, where c is a hyper-parameter

$$\mathcal{L}' = \mathcal{L} + c \cdot \text{sum}(\text{ReLU}(\sigma - \mathbf{w}_{\text{scale}})). \tag{16}$$

We do experiments on the Cora dataset using ScaleGCN and plot the results as heat maps in Fig. 10. The weight constraint trick can help improve the accuracy result significantly on models without β decay. On models with β decay, this constraint trick can also improve the accuracy but this improvement is not as dramatic as on models without β decay.

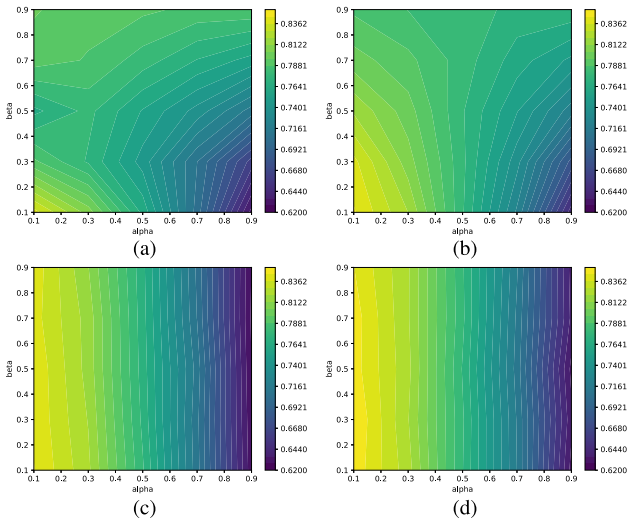


Fig. 10. Heat maps of accuracy on Cora obtained by ScaleGCN with and without β decay and weight constraint. The x-axis: α; y-axis: β. (a) Without decay and without constraint. (b) Without decay and with constraint. (c) With decay and without constraint. (d) With decay and with constraint.

TABLE X

CLASSIFICATION ACCURACY ON VARIOUS GRAPH CLASSIFICATION DATASETS. THE SUBSCRIPT "SCALE" MEANS WE CHANGE ALL GRAPH CONVOLUTION LAYERS TO SCALEGCN LAYER IN THE SAGPOOL MODEL

Methods	DD	PROTEINS	NCI1	NCI109
DiffPool	66.95	68.20	62.32	61.98
gPool	75.01	71.10	67.02	66.12
SAGPool	**76.45**	71.86	67.45	67.86
$SAGPool_{scale}$	74.71	**72.32**	**70.40**	**70.56**

G. Graph Classification

For the graph classification task, we replace all graph convolution layers in SAGPool [30] model with ScaleGCN layers, and compare this new model with the original SAGPool and other two hierarchical graph pooling methods: DiffPool [31] and gPool [32]. We reuse the results already reported in [30] and conduct experiments on our model 20 times. The mean classification accuracy results are reported in Table X. The experimental result shows that using scale graph convolution has a positive effect on the classification performance.

VI. CONCLUSION

In this article, we have presented scale graph convolution, an efficient graph convolution scheme using channel-wise

scale transformation to address the channel mixing problem, which is also first formally identified and analyzed by our work. Several GNN variants like ScaleGCN are proposed based on our new scale graph convolution, and these models, based on the purpose they were designed for, can alleviate the over-fitting phenomenon in previous works and improve the accuracy and inference speed. The theoretical analysis shows that ScaleGCN has lower time and space complexity compared with the vanilla GCN. Experimental results on small graphs show that ScaleGCN can converge with fewer layers than previous works. Experimental results on large-scale graph datasets show that ScaleGCN is more efficient than many previous works, and the risk of over-fitting is smaller. Experiments on channel selection and pruning also show that the scale graph convolution has the ability to distinguish channel importance and pruning inessential channels can improve inference speed for models.

Interesting directions for future work include more theoretical analysis on ScaleGCN and exploration of NAS using scale graph convolution.

REFERENCES

[1] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and locally connected networks on graphs," in *Proc. 2nd Int. Conf. Learn. Represent., (ICLR)*, Banff, AB, Canada, Apr. 2014, pp. 1–4.

[2] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Proc. Adv. Neural Inf. Process. Syst., Annu. Conf. Neural Inf. Process. Syst.* Barcelona, Spain, Dec. 2016, pp. 3837–3845.

[3] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *Proc. 5th Int. Conf. Learn. Represent., (ICLR)*. Toulon, France, Apr. 2017, pp. 1–14.

[4] W. L. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proc. Adv. Neural Inf. Process. Syst., Annu. Conf. Neural Inf. Process. Syst.* Long Beach, CA, USA, Dec. 2017, pp. 1024–1034.

[5] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *Proc. 6th Int. Conf. Learn. Represent., (ICLR)*. Vancouver, BC, Canada, Apr. 2018, pp. 1–12.

[6] K. Xu, C. Li, Y. Tian, T. Sonobe, K. Kawarabayashi, and S. Jegelka, "Representation learning on graphs with jumping knowledge networks," in *Proc. 35th Int. Conf. Mach. Learn. (ICML)*, vol. 80. Stockholm, Sweden: Stockholmsmässan, Jul. 2018, pp. 5449–5458.

[7] F. Wu, A. Souza, T. Zhang, C. Fifty, T. Yu, and K. Weinberger, "Simplifying graph convolutional networks," in *Proc. 36th Int. Conf. Mach. Learn.*, vol. 97, Jun. 2019, pp. 6861–6871.

[8] A. Kazi *et al.*, "Inceptiongcn: Receptive field aware graph convolutional network for disease prediction," in *Proc. 26th Int. Conf. Inf. Process. Med. Imag. (IPMI)* in Lecture Notes in Computer Science, vol. 11492. Hong Kong: Springer, Jun. 2019, pp. 73–85.

[9] G. Li, M. Müller, A. Thabet, and B. Ghanem, "DeepGCNs: Can GCNs go as deep as CNNs?" in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 9266–9275.

[10] W.-L. Chiang, X. Liu, S. Si, Y. Li, S. Bengio, and C.-J. Hsieh, "Cluster-GCN: An efficient algorithm for training deep and large graph convolutional networks," in *Proc. 25th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Jul. 2019, pp. 257–266.

[11] J. Klicpera, A. Bojchevski, and S. Günnemann, "Predict then propagate: Graph neural networks meet personalized pagerank," in *Proc. 7th Int. Conf. Learn. Represent., (ICLR)*. New Orleans, LA, USA, May 2019, pp. 1–15.

[12] M. Chen, Z. Wei, Z. Huang, B. Ding, and Y. Li, "Simple and deep graph convolutional networks," in *Proc. 37th Int. Conf. Mach. Learn.*, vol. 119, Jul. 2020, pp. 1725–1735.

[13] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web," Stanford InfoLab, Stanford, CA, USA, Tech. Rep. 1999-66, 1999.

[14] B. Perozzi, R. Al-Rfou, and S. Skiena, "DeepWalk: Online learning of social representations," in *Proc. 20th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2014, pp. 701–710.

[15] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2016, pp. 855–864.

[16] Q. Li, Z. Han, and X. Wu, "Deeper insights into graph convolutional networks for semi-supervised learning," in *Proc. 32nd AAAI Conf. Artif. Intell., (AAAI), 30th Innov. Appl. Artif. Intell. (IAAI), 8th AAAI Symp. Educ. Adv. Artif. Intell. (EAAI)*. New Orleans, LA, USA: AAAI Press, Feb. 2018, pp. 3538–3545.

[17] J. Klicpera, S. Weißenberger, and S. Günnemann, "Diffusion improves graph learning," in *Proc. Adv. Neural Inf. Process. Syst., Annu. Conf. Neural Inf. Process. Syst.* Vancouver, BC, Canada, Dec. 2019, pp. 13333–13345.

[18] Z. Yang, W. W. Cohen, and R. Salakhutdinov, "Revisiting semi-supervised learning with graph embeddings," in *Proc. 33rd Int. Conf. Mach. Learn., (ICML)*, vol. 48. New York, NY, USA, Jun. 2016, pp. 40–48.

[19] Y. Rong, W. Huang, T. Xu, and J. Huang, "Dropedge: Towards deep graph convolutional networks on node classification," in *Proc. 8th Int. Conf. Learn. Represent., (ICLR)*. Addis Ababa, Ethiopia, Apr. 2020, pp. 1–17.

[20] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" in *Proc. 7th Int. Conf. Learn. Represent., (ICLR)*. New Orleans, LA, USA, May 2019, pp. 1–17.

[21] B. Y. Weisfeiler and A. A. Leman, "A reduction of a graph to a canonical form and an algebra arising during this reduction," *Nauchno-Technicheskaya Informatsia*, vol. 2, no. 9, pp. 2–16, 1968.

[22] S. Abu-El-Haija, A. Kapoor, B. Perozzi, and J. Lee, "N-GCN: Multi-scale graph convolution for semi-supervised node classification," in *Proc. 35th Uncertainty Artif. Intell. Conf.*, vol. 115. TelAviv, Israel, 2020, pp. 841–851.

[23] S. Abu-El-Haija *et al.*, "MixHop: Higher-order graph convolutional architectures via sparsified neighborhood mixing," *Proc. Mach. Learn. Res.*, vol. 97, pp. 21–29, May 2019.

[24] M. Liu, H. Gao, and S. Ji, "Towards deeper graph neural networks," in *Proc. 26th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2020, pp. 338–348.

[25] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, "Graph convolutional neural networks for web-scale recommender systems," in *Proc. 24th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Jul. 2018, pp. 974–983.

[26] J. Chen, T. Ma, and C. Xiao, "FastGCN: Fast learning with graph convolutional networks via importance sampling," in *Proc. 6th Int. Conf. Learn. Represent., (ICLR)*. Vancouver, BC, Canada, May 2018, pp. 1–15.

[27] S. Min *et al.*, "Pytorch-direct: Enabling GPU centric data access for very large graph neural network training with irregular accesses," 2021, *arXiv:2101.07956*.

[28] W. Hu *et al.*, "Open graph benchmark: Datasets for machine learning on graphs," in *Proc. Adv. Neural Inf. Process. Syst., Annu. Conf. Neural Inf. Process. Syst.*, Dec. 2020, pp. 22118–22133.

[29] C. Morris, N. M. Kriege, F. Bause, K. Kersting, P. Mutzel, and M. Neumann, "Tudataset: A collection of benchmark datasets for learning with graphs," 2020, *arXiv:2007.08663*.

[30] J. Lee, I. Lee, and J. Kang, "Self-attention graph pooling," in *Proc. 36th Int. Conf. Mach. Learn. (ICML)*, vol. 97. Long Beach, CA, USA, Jun. 2019, pp. 3734–3743.

[31] Z. Ying, J. You, C. Morris, X. Ren, W. L. Hamilton, and J. Leskovec, "Hierarchical graph representation learning with differentiable pooling," in *Proc. Adv. Neural Inf. Process. Syst., Annu. Conf. Neural Inf. Process. Syst. (NeurIPS)*. Montreal, QC, Canada, Dec. 2018, pp. 4805–4815.

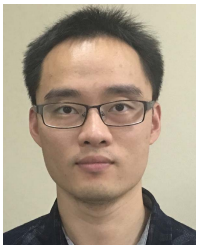
[32] H. Gao and S. Ji, "Graph U-Nets," in *Proc. 36th Int. Conf. Mach. Learn. (ICML)*, vol. 97. Long Beach, CA, USA, Jun. 2019, pp. 2083–2092.

Tianqi Zhang received the B.S. degree (Hons.) in computer science from Shanghai Jiao Tong University, Shanghai, China, in 2019, where he is currently pursuing the master's degree with the Department of Computer Science and Engineering.

He was a Research Intern with Microsoft Research Asia, Beijing, China, and the Amazon Shanghai AI Laboratory, Beijing. His current research interests include graph neural networks (GNNs) and Machine Learning (ML) system development.

Qitian Wu is currently pursuing the Ph.D. degree with the Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China.

He was a Research Intern with the Amazon Shanghai AI Laboratory. He has first-authored several papers in the top-tier conferences, such as International Conference on Machine Learning (ICML), Neural Information Processing System (NeurIPS), Conference on Knowledge Discovery and Data mining (KDD), The International Conference of World Wide Web (WWW), International Joint Conference on Artificial Intelligence (IJCAI), Conference on Information and Knowledge Management (CIKM), and International Conference on Data Mining (ICDM). His research interests include Machine Learning (ML) and data mining.



Junchi Yan (Senior Member, IEEE) received the Ph.D. degree from the Department of Electronic Engineering, Shanghai Jiao Tong University, Shanghai, China, in 2015.

He is currently an Associate Professor with Shanghai Jiao Tong University. Before that, he was with IBM Research, Beijing, China, where he started his career in April 2011. He has published over 100 peer-reviewed articles in top venues, in machine learning and computer vision.

Dr. Yan served as the Area Chair for Neural Information Processing System (NeurIPS), International Conference on Machine Learning (ICML), Computer Vision and Pattern Recognition Conference (CVPR), and Association for the Advancement of Artificial Intelligence (AAAI); and an Associate Editor for *Pattern Recognition*.

Yunan Zhao received the B.E. and M.E. degrees in automation from Shanghai Jiao Tong University, Shanghai, China, in 2013 and 2016, respectively.

He is currently a Staff Researcher with MYbank, Ant Group, Shanghai. He was an Engineer with the China Merchants Bank, Shanghai. His research interests are Machine Learning (ML) and computer vision.

Bing Han is currently a Senior Staff Engineer and the Head of the Intelligent Engine Department, MYbank, Ant Group, Shanghai. Her research interests are Machine Learning (ML) and data intelligence, especially in recommender systems and financial technology.