Multi-hop Hierarchical Graph Neural Networks

Hui Xue[1,2], Xin-Kai Sun[1,2], Wei-Xiang Sun[1,2]
[1]*School of Computer Science and Engineering, Southeast University*
[2]*MOE Key Laboratory of Computer Network and Information Integration (Southeast University)*
Nanjing, 210096, China
{hxue, xinkais, vex-soon}@seu.edu.cn

***Abstract*—Graph representation learning has been widely applied to graph tasks such as node classification, link prediction, graph-level classification and so on. Graph representation learning takes full advantage of spatial or spectral approaches to embed nodes into low-dimension space correctly and effectively. Especially, Graph Neural Networks (GNNs), which is one of representation learning, attract increasing interests due to their powerful ability to integrate local information and outstanding generalization in graph tasks. Currently, GNN models are also able to capture further nodes information with more stacked convolution layers. But when the number of layers reaches a certain level, these existing methods perform worse as the depth of networks increases continuously. Thus, most GNNs employ shallow architectures, leading to the failure of capturing further information. In this paper, we present Multi-hop Hierarchical Graph Neural Networks (MHGNNs), a new graph neural network framework, to address the shortcomings of lacking further node information and obtain broad receptive field. Distinct from prior works, one layer of MHGNNs can concatenate hop-level features in a hierarchical way, where features in the same hop are aggregated with each other. Another advantage is that MHGNNs have a flexible structure, where the number of used hops can be different in each layer. Besides, MHGNNs also use attention mechanism during the integrated step, which mines latent relationships among hops and adaptively selectes important hop-level features. Finally, our MHGNN model was evaluated on citation and protein-protein interaction graph benchmarks to conduct node classification, and has advanced or matched the outcomes of state-of-the-art methods in node classification tasks.**

***Index Terms*—graph representation learning, graph neural networks, attention mechanism**

I. Introduction

Convolutional Neural Networks (CNNs) have acquired great success in the field of image and video analysis. CNNs can embed image data into a low-dimension space to tackle relevant tasks [1] [2] [3]. But convolutional neural networks are only designed to data with Euclidean structures, where this kind of data has fixed arrangement, thus CNNs can easily extract neighbors in accordance with certain order. Because of this characteristic, CNNs can use fixed-size filters to learning local information and achieve parameter sharing.

However, most non-Euclidean structured data, like manifolds and graphs, can not be arranged into fixed order, which makes CNNs invalid. Unfortunately, non-Euclidean structured data do exist in telecommunication, society and biology widely. Therefore, it is necessary to design new methods to solve problems with non-Euclidean structured data. In the past years, graph representation learning has been empirically applied to graph tasks such as node classification [4], link prediction [5], graph classification [6] and so on. Especially, Graph Neural Networks (GNNs) [7] [8] [9], as an outstanding method among graph representation learning, have attracted increasing interests in node classification including transductive and inductive tasks (The graph in transductive tasks is definitive while inductive tasks require problems with unseen graphs to be solved.).

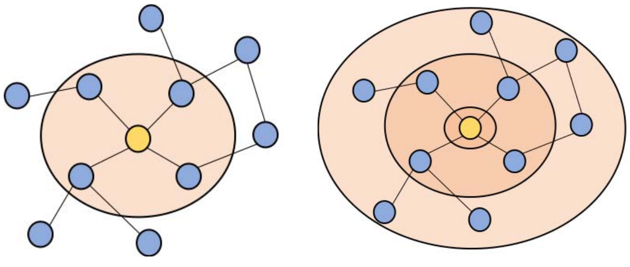
(a) prior models (b) our models

Fig. 1: Comparsion between prior convolution models (a) and ours (b). We show the current node (yellow node) captures further and more hop nodes (blue ones) with our model. The cover ellipses (orange ones) is represented as importance to current node with different color shades.

Most existing GNN methods are intended to get local information by aggregating one-hop neighbors and employing more layers (hop refers to the interval between two nodes, and one-hop nodes are also called neighbors in this paper). But the performance would become worse as the number of layers increases [10], thus only 3—4 layers are stacked in most GNN models, which results in few local nodes or narrow receptive field. This phenomenon can be interpreted as Laplacian smoothing—it explains that all nodes in a graph will have same features when layers are towards infinity [11]. The majority of current methods are in the dilemma where networks should be undoubtedly deeper for wider receptive field but should be shallow for better perfermance. To get rid of this dilemma, we extend existing methods to obtain broader

2375-9356/20/$31.00 ©2020 IEEE
DOI 10.1109/BigComp48618.2020.00-95

receptive field.

We propose a general framework called Multi-hop Hierarchical Graph Neural Networks (MHGNNs). To the best of our knowledge, it is a new framework which makes full use of multi-hop node information to broaden receptive field. Unlike previous methods that only aggregate one-hop neighbors in a layer, our methods can capture nodes within multiple hops in a layer. Fig. 1 shows that MHGNNs can preserve multi-hop node information with different importance in comparison to prior models. Meanwhile, the number of hops is variable in differnet layers, which brings flexibility to MHGNNs. If nodes within multiple hops are considered independently, huge complexity will make models hard to train. Through the observation of convolution, we reuse neighbor information and apply aggregation function to extract multi-hop node features in a hierarchical way, which reduces complexity effectively. Because further hop nodes are integrated into current node features from hop to hop, we call this method as multi-hop hierachical structure. Furthermore, attention mechanism is a powerful tool for mining data relationships in natural language processing [12]. And it has been applied to Graph Attention Networks (GATs) [7]. Inspired by this work, we distribute attention to represent the latent relationships among hops. Due to attention, MHGNNs can selectively extract significant hop-level features.

MHGNNs take a graph as input and obtain representations of every node. The representations are sent to fully connected layers to make node classification including both transductive and inductive tasks. To demonstrate MHGNNs' ability in graph representation learning, we conducted experiments on four graph benchmarks and reach the state-of-the-art level.

Concretely, our contributions are the following:

- We design a multi-hop hierachical structure to reach further hop nodes, acquiring wider receiptive field.
- Neighbor information is reused, reducing complexity effectively.
- In the fusion process of multi-hop nodes, we apply attention mechanism to control importance of hop-level features, which is beneficial to mine latent relationships.
- We evaluate MHGNNs on four graph benchmarks including citation network datasets and protein-protein interaction (PPI) dataset. Our MHGNN model has achieved the state-of-the-art results.

The rest of paper is as follows. In section II, we present related work about our models. Then, our proposal method is described in detail in section III. And we conduct our exiperiments and analyze results in section IV. Finally, conclusions are conducted in section V.

II. Related Work

Our models have a strong link with node embeddings and graph neural networks. Node embeddings generally apply unsupervised learning to get node representation, and graph neural networks adopt semi-supervised learning or supervised learning.

Node Embeddings Node embeddings aim at learning low-dimension latent representation of nodes in a graph. Thus, it is of great significance to thoroughly capture related information. DeepWalk model [13] uses a random walk method, specifically, a node searches for the next node according to the normalized edge weight. Then Skip-Gram model [14] is designed for representation of nodes. Based on DeepWalk, Tang et al. in [15] adopted first-order nearest neighbors and second-order nearest neighbors, which is so called one-hop and two-hop, fusing two-order nearest neighbors methods to generate node representation. Node2vec is another method to find broader nodes by controlling transition probability [16]. For global structural information, GraRep captures k-step relational information from graph directly, which is manipulating various transition matrices [17].

These models have done outstanding works in exploring nodes via edges in graphs, which makes great success because of further nodes captured by random walk or global information. Inspired by these work, our methods also try to integrate furher nodes and mine latent relationships.

Graph Neural Networks Graph neural networks and variants have emerged a lot in recent years [8] [7] [9] [18] [19] [20] [24]. Similar to convolutional networks, GNNs adopt learnable and parameterized filters to update node features with neighbors information. According to the type of convolution, GNNs can be divided into spectral approaches and spatial approaches. Spectral approaches make convolution operation in the Fourier domain like GCNs [8]. And spatial approaches directly define convolutions in original graph, utilizing neighbors as convoluted objects. Among these approaches, the famous representatives include GraphSAGE [9], GATs [7] and so on. The significant benefit of spatial approaches is that graphs can use neighbor information flexibly. But either spectral approaches or spatial approaches only work on one-hop neighbors in a layer. For the further nodes, these approaches have no choice to stack more layers, causing worse performance [10]. Thus, some work has been designed to git rid of shallow networks, absorbing experience from the advances in CNNs [21] [22] [23]. In [10], Li et al. distribute residual connections, dense connections and dilation to GCNs and applied deeper architectures. From the results of deeper GCNs, it is not difficult to find that further nodes play the essential role in representation learning. But how to concatenate further nodes is a vacant position in graph neural networks.

Inspired by above works, our model emerges as a general framework to integrate further nodes. Our models adopt spatial approaches and extend neighbor information.

III. Proposed Method

In this section, we will explain how to construct our model—MHGNNs. The key point is to collect the information of multi-hop nodes, and to integrate them into the current node. Fortunately, our model receives abundant node information at the expense of low complexity. What's more, it is also demonstrated the way of utilizing attention to learn hop-to-hop relationships. According to the method-building process, we divide our model into two part: multi-hop hierarchical structure and attention mechanism.

A. Multi-hop Hierarchical Structure

Multi-hop hierarchical structure effectively collects node information from hop to hop, and different hop nodes hierarchically stack together, which is shaped like a tree as shown in Fig. 2. The depth of the tree corresponds to the maximum hop. It is noteworthy that this tree is only an abstract model for understanding this structure, and general graph is used in calculation. Next, how one layer collects multi-hop nodes information will be introduced. We split the structure of one layer into three parts: linear transformation, aggregation and multi-hop concatenation. Linear transformation can transfer input features into a new feature space, aggregation provides some functions to extract hop-level features, and multi-hop concatenation is designed to concatenate hop-level features to every node.

Linear Transformation The input of every layer is a set of node features, $\boldsymbol{A} = \{\boldsymbol{a}_1, \boldsymbol{a}_2, ..., \boldsymbol{a}_N\}, \boldsymbol{a}_i \in \mathbb{R}^M$, where N represents the number of nodes in the graph, and M is the dimension of features in every node. In every layer, there is a weight matrix, $\boldsymbol{W} \in \mathbb{R}^{M' \times M}$, where M' is dimension of output features . This matrix provides a linear transformer for input features. Thus output features $\boldsymbol{h}_i^0$ is

$$\boldsymbol{h}_i^0 = \boldsymbol{W} \cdot \boldsymbol{a}_i \tag{1}$$

where $i \in \{1, 2, ..., N\}$, $\boldsymbol{h}_i^0$ represents hop-level features that are computed on zero-hop nodes, that is node iteself, and $\boldsymbol{h}_i^0 \in \mathbb{R}^{M'}$. Therefore, we can produce output features $\boldsymbol{H} = \{\boldsymbol{h}_1^0, \boldsymbol{h}_2^0, ..., \boldsymbol{h}_N^0\}$. Linear transfomation can project the original input space into a new feature space, and parameters of $\boldsymbol{W}$ are learnable and shared by all nodes.

Aggregation Aggregation can be viewed as a process that integrates one-hop nodes features by a class of functions with special properties [9]. Unlike Euclidean structured data, the neighbors of nodes in graph have no fixed order and number, which means that neighbors may have many different permutations. Thus, we must apply symmetric aggregation function that is invariant to permutations of neighbors, in order to ensure that our model is able to tackle graph tasks. Our aggregation functions follow a general framework in [9]. Besides the framework in [9], we find out that ADD aggregation function is also a good function in [20]. So three symmetric aggregation functions are examined.

- We firstly introduce $MEAN$ aggregation function which is only intended to take column-element-wise mean of features in $\{\boldsymbol{h}_u^0, u \in \mathcal{N}(i)\}$, where $\boldsymbol{h}_u^0$ represents the neighbors' features of node i and $\mathcal{N}(i)$ means the neighbors' set of node i. The mean aggregation function amounts to convolution operator, where all weights are the inverse of $|\mathcal{N}(i)|$, where $|\mathcal{N}(i)|$ refers to the number of set. Moreover, we can use another weight matrix $\boldsymbol{W}_{mean}$ to reweight neighbors. Thus we design an $AGGREGATION_{mean}$ function:

$$\begin{aligned} &AGGREGATION_{mean}(\boldsymbol{h}_i^0) = \\ &\qquad \boldsymbol{W}_{mean} \cdot MEAN(\{\boldsymbol{h}_u^0, u \in \mathcal{N}(i)\}). \end{aligned} \tag{2}$$

  In our experiments, this weight matrix is set as a square matrix to make dimension consistent. And a shadow network could replace W_{mean}. Of course, if we replace the weight matrix with a shadow network, it would make some changes in formula but it is still a variant of mean aggregation. Therefore, we do not distinguish these variants.
- In addition to $MEAN$ aggregation function, ADD function is also suggested to take column-element-wise addition in features of neighbors. It can be regarded as a special case of mean function. When we set row-level parameters of $\boldsymbol{W}_{mean}$ as $|\mathcal{N}(i)|$, they will become consistent. But we discover that this function has efficient calculation and reaches better performance in some datasets. An $AGGREGATION_{add}$ function is designed:

$$AGGREGATION_{add}(\boldsymbol{h}_i^0) = ADD(\{\boldsymbol{h}_u^0, u \in \mathcal{N}(i)\}). \tag{3}$$

  It is noted that the distinction between (3) and (2) is weight matrix. As mentioned above, we avoid the duplication of definitions to omit the weight matrix.
- Finally we examine another symmetric aggregation function, max aggregation function, which is inspired by pooling operation in CNNs. In this work, neighbors' features can be mapped with a parameterized matrix independently. Following networks' mapping, a column-element-wise max pooling operation will be taken, which is similar to pooling layer in CNNs. Therefore, max aggregation function, $AGGREGATION_{max}$, is defined as follows;

$$\begin{aligned} &AGGREGATION_{max}(\boldsymbol{h}_i^0) = \\ &\qquad MAX(\{\boldsymbol{W}_{max} \cdot \boldsymbol{h}_u^0 + \boldsymbol{b}\}, u \in \mathcal{N}(i)). \end{aligned} \tag{4}$$

  where $\boldsymbol{W}_{max}$ represent a weight matrix and $\boldsymbol{b}$ is a bias. Intuitively, $\boldsymbol{W}_{max}$ can be also replaced by a fully connected network which will be trained as a set of functions capturing neighbors information.

We have discussed three symmetric aggregation functions. In principle, every function has its own scenarios. In previous work [20], it pointed out that $MEAN$ and ADD aggregation functions perform better and we got the same results in real experiments. So we focused on these two aggregation functions in our experiments.

Multi-hop Concatenation So far we only integrate neighbors' features by using aggregation functions, but the goal of our model is to extract multi-hop node features. An intuitive and simple idea is to explore arbitrary hop nodes of node i and store them into neighbors directly, thus multi-hop nodes that we need will be obtained independently. Though this idea

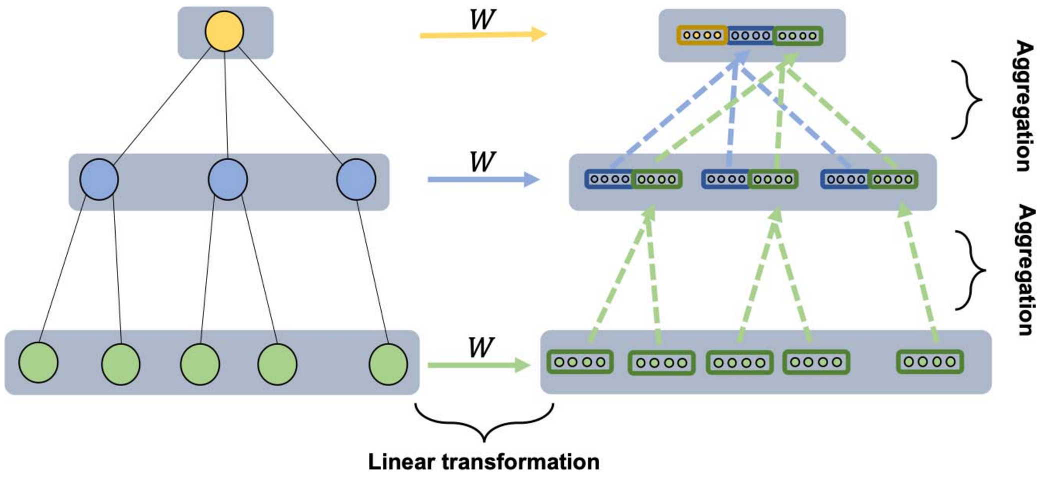

Fig. 2: A toy example of multi-hop hierarchical structure, where all nodes within 2 hop are used. Thus, the subgraph constructed with hop nodes can be seen as a tree, like the left part of this figure. In the left, every node has inherent features as inputs, then obtains new features via linear transformation $\boldsymbol{W}$ which represented as a solid line with arrow in figure. Iteratively, further hop information is delivered to upper levels via $AGGREGATION$ function represented by dotted lines with arrow in figure.

seems to be feasible, there exist obvious drawbacks that huge storage will be in need for neighbors when a graph owns millions of nodes and edges, and the procedure of exploring takes expensive complexity. So considering nodes independently is not a practical strategy. Actually, it is unnecessary to acquire every node, we just focus on neighbors features, and multi-hop node features will be gathered automatically.

As mentioned in Aggregation part, one-hop node features of every node have been integrated by aggregation functions. It is apparently different from previous works that one-hop node features are stored separately from zero-hop features. Corresponding to every node, its two-hop node features are entirely contained within its neighbors' one-hop features, three-hop features are in neighbors' two-hop features, and so on. Nodes get multi-hop features from hop to hop as hierarchical models. We define $\boldsymbol{h}_i^k$ as k hop features. Following above observation, $\boldsymbol{h}_i^k$ can be obtained as:

$$\boldsymbol{h}_i^k = AGGREGATION(\boldsymbol{h}_i^{k-1}) \tag{5}$$

where $AGGREGATION$ is any aggregation functions. For preserving features independently, all hop-level features apply concatenation operation to connect features in series. Thus, multi-hop features $\boldsymbol{h}_i$

$$\begin{aligned} \boldsymbol{h}_i &= [\boldsymbol{h}_i^0 \mid \boldsymbol{h}_i^1 \mid ... \mid \boldsymbol{h}_i^K] \\ &= \boldsymbol{h}_i^0 \overset{K}{\underset{k=1}{\|}} AGGREGATION(\boldsymbol{h}_i^{k-1}) \end{aligned} \tag{6}$$

where K is the maximum hop set manually and $\|$ represents concatenation operation.

Multi-hop hierarchical structure makes full use of neighbors features and concatenates all hop-level features in one layer successfully and effectively. Compared with other methods, our method has more receptive fields and it only needs to occupy $O(KNM')$ storage in one layer, which is growing linearly. As for time complexity, it is assumed that the time complexity of function is $O(\gamma)$ which is proportional to the total number of neighbors, and our model just runs aggregation function iteratively, thus consumes $O(K\gamma)$. It is also linear growth.

B. Attention Mechanism

In this part, we will further extract useful and adequate features from multi-hop features $\boldsymbol{h}_i = [\boldsymbol{h}_i^0 \mid \boldsymbol{h}_i^1 \mid ... \mid \boldsymbol{h}_i^K]$. Obviously, different hop-level feature $\boldsymbol{h}_i^k$ has different importance for current node, thus it is of great significance to give various and learnable weights to $\boldsymbol{h}_i^k$. Attention mechanism is a tool that is frequently used to measure importance, which is an indispensable factor to GATs' great success [7]. So our model utilizes attention mechanism, following GATs, but we focus on different hop-level features.

On the basis of the work of GATs, input features are transformed into new feasure space as we do in last part—Multi-hop Hierarchical Structure. Meanwhile, we also need to design a new mapping function to compute attention socres of hop-level features $\boldsymbol{h}_i^k$, where the function should take two inputs in $\mathbb{R}^{M'}$ into a real number in $\mathbb{R}$. So, we then perform an attention-score function $F(\cdot,\cdot)$:

$$F(\boldsymbol{h}_i^0, \boldsymbol{h}_i^k) = LeakyRelu(\boldsymbol{a}^T \cdot [\boldsymbol{h}_i^0 \| \boldsymbol{h}_i^k]) \tag{7}$$

where $\boldsymbol{a}$ is a parametrized vector, $\cdot^T$ is transposition, and $LeakyRelu(\cdot)$ represents leakyrelu nonlinearity activation function. $LeakyRelu(\cdot)$ is defined as follows:

$$LeakyRelu(x) = \begin{cases} x & \text{if } x \geq 0 \\ \beta x, \beta \in (0,1) & \text{if } x < 0 \end{cases} \tag{8}$$

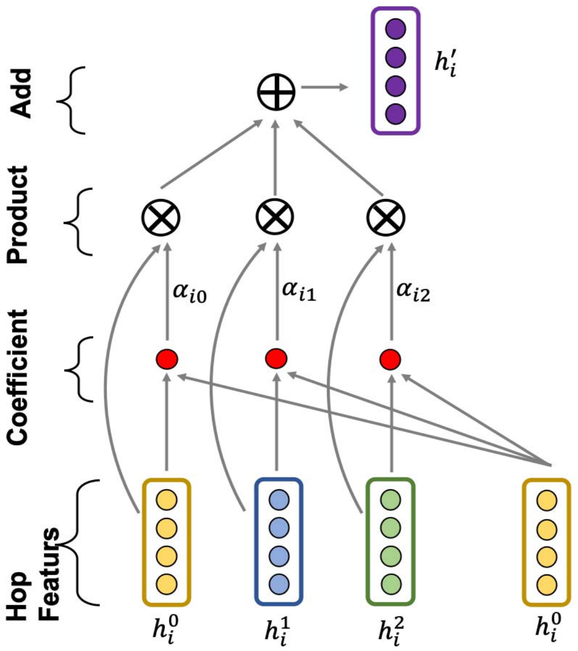

Fig. 3: The diagram of attention mechanism which is a two-hop case. The features $\boldsymbol{h}_i^0$ of current node (yellow features) is computed with other hop features $\boldsymbol{h}_i^k, k \in \{0,1,2\}$ to get attention scores. Meanwhile, normalized function is applied to get attention coefficients α_{ik}, which is represented in red node. Then eventual features $\boldsymbol{h}_i^{'}$ are obtained via product and addition.

In all of our experiments, β is set as 0.1.

The attention-score function computes a set of values to indicate the importance of multi-hop nodes to current node. But scores may be over large, which will limit model training. So scores should be normalized across multi-hop node features of current nodes. Intuitively, scores are larger and features are more important. Following the intuitive idea, a monotonic increasing function is expected. Mathematically, softmax function is an appropriate choice. Thus, normalization function may be then expressed as:

$$\alpha_{ik} = \text{softmax}(F(\boldsymbol{h}_i^0, \cdot)) = \frac{\exp(F(\boldsymbol{h}_i^0, \boldsymbol{h}_i^k))}{\sum_{j=0}^{K} \exp(F(\boldsymbol{h}_i^0, \boldsymbol{h}_i^j))} \tag{9}$$

The normalized scores are called attention coefficients. The coefficients are utilized to combine multi-hop features. To simplify model complexity, normalized coefficients is multiplied with the correponding features, then all multi-hop features apply a linear combination and an activation unit σ :

$$\boldsymbol{h}_i^{'} = \sigma\Big(\ \sum_{k=0}^{K} \alpha_{ik} \boldsymbol{h}_i^k\ \Big) \tag{10}$$

Finally, we output features $\boldsymbol{h}_i^{'}$ of every node in one layer. In detail, the specific architecture of applied attention is constructed with four steps, as shown in Fig. 3. Through stacking more layers, our model obtains representations of every node. Then these representations can be used to tackle various graph tasks.

In order to make our model clearer, we summarize our model in Algorithm 1 in detail.

C. Classification Networks

Our MHGNNs adopt multi-hop hierarchical structure and attention mechanism to capture more and further hop nodes information. Hence, the eventual node-level representations contain abundant information. To evaluate our models, we develop a fully connected network as our classifier. The network consists of one hidden layer, which is so called a three-layer network (including input and output layer). We apply $Relu(\cdot)$ activation function to input and hidden layer. And output layer is connected to softmax function or sigmoid function for single-label or multi-label classification. And cross-entropy is utilized to compute the loss.

Algorithm 1 Algorithm for one-node representations in one layer of MHGNNs

```
Input:  $\boldsymbol{A} = \{\boldsymbol{a}_1, \boldsymbol{a}_2, ..., \boldsymbol{a}_N\}$, inputs features
        $i$, node number
        $K$, maximum hop
        $F(\cdot,\cdot)$, attentin-score function
Output: $\boldsymbol{h}_i^{'}$, node-level featurs
$\boldsymbol{h}_i^0 = \boldsymbol{W} \cdot \boldsymbol{a}_i$
for $k = 1$ to $K$ do
    $\boldsymbol{h}_i^k = AGGREGATION(\boldsymbol{h}_i^{k-1})$
end for
Obtain $\boldsymbol{h}_i = [\boldsymbol{h}_i^0 \mid \boldsymbol{h}_i^1 \mid ... \mid \boldsymbol{h}_i^K] = \|_{k=0}^{K} \boldsymbol{h}_i^k$
for $k = 0$ to $K$ do
    $\alpha_{ik} = \frac{\exp(F(\boldsymbol{h}_i^0, \boldsymbol{h}_i^k))}{\sum_{j=0}^{K} \exp(F(\boldsymbol{h}_i^0, \boldsymbol{h}_i^j))}$
end for
$\boldsymbol{h}_i^{'} = \sigma\big(\ \sum_{k=0}^{K} \alpha_{ik} \boldsymbol{h}_i^k\ \big)$
return $\boldsymbol{h}_i^{'}$
```

IV. Experiments

In this section, we demonstrate benchmark datasets, experiment setup and results. We have evaluated our MHGNN models compared with other existing state-of-the-art baseline methods on four widely used benchmark dataset for node classification including transductive and inductive tasks. In experiments, we acquire highly competitive results to prove the effectiveness of our methods.

A. Benchmark Datasets

Benchmark tasks are composed of transductive and inductive tasks, where different datasets are used for evaluation. We give an overview of these datasets in TABLE I .

Transductive Tasks Three graph benchmark datasets generated from standard citation networks which are Cora, Citerseer and Pubmed [25] are used for transductive tasks. Each dataset has only one graph. All of these graphs are undirected, where

TABLE I: Benchmark Datasets Statistics

	Cora	**Citeseer**	**Pubmed**	**PPI**
Nodes	2708	3327	19717	56944
Edges	5429	4732	44338	818716
Average degree	4.01	2.84	4.50	28.76
Features/Node	1433	3703	500	50
Labels	7	6	3	121 (multilabe)
Traing Nodes(label rate)	140 (0.052)	120 (0.036)	60 (0.003)	44906 (20 graphs)
Validation Nodes(label rate)	500 (0.185)	500 (0.150)	500 (0.025)	6514 (2 graphs)
Testing Nodes(label rate)	1000 (0.369)	1000 (0.301)	1000 (0.051)	5524 (2 graphs)

nodes represent papers and edges are citation links. And the goal is to predict paper categories correctly. Among these datasets, Cora dataset has 2708 nodes, average degree is 4.01, and the number of classes is 7. The Citeseer dataset contains 3327 nodes, average degree is 2.84, and the number of classes is 6. The Pubmed dataset consists of 19717 nodes, average degree is 4.50, and the number of classes is 3. Following previous experimental design [7] [8], we also apply a few nodes for training step, 500 nodes for validation, and 1000 nodes for testing.

Inductive Tasks Inductive tasks involve predicting nodes label in unseen graphs, so we adopt protein-protein interaction (PPI) dataset which has 24 graphs. Each graph in PPI corresponds to a different human tissue [26]. Given by positional gene sets, motif gene sets and immunological signatures collected from the Molecular Signatures Database [27], PPI dataset has 50 features and 121 labels (multilabel) per node. Each graph contains 2373 nodes on average, and average degree is 28.76. Like experimental setup of prior works [7] [9], we use 20 graphs to train models, and 2 graphs to validate models and 2 graphs to test models, where data have been processed by GraphSAGE.

B. Baseline Methods

We compared our approach with some baseline methods, including unsupervised learning in graph, GNNs and variants.

Transductive Task Baseline We compare against the same baselines as GATs [7], i.e. Multi-Layer Perception (MLP), label propagation (LP) [28], semi-supervised embedding (SemiEmb) [29], manifold regularization (ManiReg) [30], DeepWalk [13], the iterative classification algorithm (ICA) [31], Planetoid [32] and Chebyshev [33]. GCNs [8], MoNet [34] and GATs [7] are also used as comparisons.

Inductive Task Baseline For the inductive task, there are a few prior works. As far as we all know, GraphSAGE [9] and GATs [7] are fundamental achievements in this task. So we compare our model with GraphSAGE-GCN, GraphSAGE-mean, GraphSAGE-LSTM, and GraphSAGE-pool. A distinction among these approaches is that they employ a variety of aggregator functions, for example, GraphSAGE-mean takes the element-wise mean value of feature vectors, yet GraphSAGE-LSTM uses LSTM units to aggregate neighbors (LSTM is not invariant to permutations). Another particular point is that GraphSAGE samples parts of neighbors from all. Besides, we also use GATs as a baseline in this task. Finally, we present MLP as a node classifier in order to prove models' efficiency.

C. Experimental Setup

All experiments run on the PC with an Nvidia TITAN RTX GPU with 24G RAM. And we implemented our models in Python with the excellent Pytorch-Geometric framework [35]. The experimental setup depends on types of tasks, so we distribute diverse architectures to four datasets. Dataset architectures are as follows:

- For Cora dataset, we design a four-layer MHGNN. The first layer consists of a five-hop hierarchical structure, that is $K = 5$. Then the second layer is a two-hop-structure layer, yet the third and fourth layer are both a one-hop layer. As for aggregation funtion and the dimension of features, the first two layers apply $MEAN$ function to compute $M' = 1024$ features, and the last two layers use ADD function to compute 512 features.
- For Citeseer dataset, we design a five-layer MHGNN. The first two layers match with Cora's. But in the third layer, we set $K = 2$. And one-hop hierarchical structure is applied to the rest of layers. We set all features as 1024-dimension vectors. And all layers apply $MEAN$ function as aggregation except that the last layer uses ADD function.
- For Pumbed dataset, we adopt a five-layer MHGNN that is the same with Citeseer.
- For PPI dataset, we distribute a five-layer MHGNN. The first layer consists of 256 features and a three-hop structure. The second and third layer have a two-hop structure, but the former use 512 features and the feature dimension of the latter is 1024. The rest of layers have 1024-dimension features and one-hop structure. Aggregation function of all layers is $MEAN$ function.

Furthermore, dropout [36] with p = 0.4 is applied to attention mechanism. For transductive task, L_2 regularization with 0.005 weight is utilized to control model complexity. During training, the Adam SGD optimizer [37] with a learning rate of 0.001 for transductive models is to optimize cross-entropy loss, yet inductive model is trained by using the same optimizer with a learning rate of 0.0005. To make it fair, all models are initialized using Glorot initialization [38] and we rerun GATs on transductive tasks with Pytorch-Geometric framework. Moreover, our models adopt early-stop technique to avoid over-fitting.

D. Results

TABLE II summarizes the comparative results on Cora, Citeseer and Pubmed. Our results are based on 10 runs in datasets and obtained as mean accuracy in test sets with standard deviation.We achieve the best performance in Cora 82.2% and Citeseer 71.1% compared with other methods, but in Pubmed we only match the state-of-the-art results. This is because Pubmed is a large graph with 19717 nodes, which needs much more nodes. And the experimental device limits more stacked layers to get further and more nodes. However, results in Cora and Citeseer indicate effectiveness of our model.

Furthermore, TABLE III shows results on PPI. For inductive task, we report the micro-averaged F1 score on test set, averaged after 10 runs. Our model also reaches the best performance 0.988, exceeding 0.015 beyond GATs. According to TABLE I, PPI dataset have more edges than citation graphs, where average degree indicates this point. More edges indicate that more nodes are captured by our model, thus every node has more rich information than prior methods like GATs.

From the results, we find our models have higher competitiveness in node classification because of broader receiptive field. Our ideas are verified and very competitive.

TABLE II: Transductive Task Results

	Cora	**Citeseer**	**Pubmed**
MLP	55.1%	46.5%	71.4%
ManiReg	59.5%	60.1%	70.7%
SemiEmb	59.0%	59.6%	71.7%
LP	68.0%	45.3%	63.0%
DeepWalk	67.2%	43.2%	65.3%
ICA	75.1%	69.1%	73.9%
Planetoid	75.7%	64.7%	77.2%
Chebyshev	81.2%	69.8%	74.4%
GCN	81.5%	70.3%	**79.0**%
MoNet	81.7%	—	**78.8**%
GAT(implemeted)	80.1±0.7%	70.5±0.7%	**79.0**±0.3%
MHGNN(ours)	**82.2**±0.7%	**71.1**±0.5%	**78.9**±0.3%

TABLE III: Inductive Task Results

	PPI
MLP	0.422
GraphSAGE-GCN	0.500
GraphSAGE-mean	0.598
GraphSAGE-LSTM	0.612
GraphSAGE-pool	0.600
GAT	0.973± 0.002
MHGNN(ours)	**0.988**± 0.001

V. Conclusion

In this paper, we have presented Multi-hop Hierarchical Graph Neural Networks (MHGNNs), a novel architecture of graph neural networks, for node representation and classification. The MHGNNs provide a method to tackle narrow receptive field problems for graph-structured data. The multi-hop hierarchical structure concatenates hop-level features and reuses neighbors information to capture further hop-level features effectively. Moreover, attention mechanism is applied to obtain the importance of hop node features to current node, which mines latent relationships among hops. Importantly, MHGNNs have a flexible structure, where we can design various aggregation functions (keeps symmetric), and set different hops in one layer. Eventually, MHGNNs reached or matched the state-of-the-art performance compared with existing baseline methods on benchmark datasets.

Acknowledgment

This work was supported by the National Key R&D Program of China (2017YFB1002801), the National Natural Science Foundations of China (Grant No. 61876091). It is also supported by Collaborative Innovation Center of Wireless Communications Technology.

References

[1] S. Jégou, M. Drozdzal, D. Vazquez, A. Romero, and Y. Bengio, "The one hundred layers tiramisu: Fully convolutional densenets for semantic segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2017, pp. 11–19.

[2] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in neural information processing systems*, 2015, pp. 91–99.

[3] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[4] S. Zhu, K. Yu, Y. Chi, and Y. Gong, "Combining content and link for classification using matrix factorization," in *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, 2007, pp. 487–494.

[5] S. Gao, L. Denoyer, and P. Gallinari, "Temporal link prediction by integrating content and structure information," in *Proceedings of the 20th ACM international conference on Information and knowledge management*. ACM, 2011, pp. 1169–1174.

[6] Z. Ying, J. You, C. Morris, X. Ren, W. Hamilton, and J. Leskovec, "Hierarchical graph representation learning with differentiable pooling," in *Advances in Neural Information Processing Systems*, 2018, pp. 4800–4810.

[7] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.

[8] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *International Conference on Learning Representations (ICLR)*, 2017.

[9] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in Neural Information Processing Systems*, 2017, pp. 1024–1034.

[10] G. Li, M. Müller, A. Thabet, and B. Ghanem, "Can gcns go as deep as cnns?" *arXiv preprint arXiv:1904.03751*, 2019.

[11] Q. Li, Z. Han, and X.-M. Wu, "Deeper insights into graph convolutional networks for semi-supervised learning," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[12] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[13] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2014, pp. 701–710.

[14] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[15] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "Line: Large-scale information network embedding," in *Proceedings of the 24th international conference on world wide web*. International World Wide Web Conferences Steering Committee, 2015, pp. 1067–1077.

[16] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2016, pp. 855–864.

[17] S. Cao, W. Lu, and Q. Xu, "Grarep: Learning graph representations with global structural information," in *Proceedings of the 24th ACM international on conference on information and knowledge management*. ACM, 2015, pp. 891–900.

[18] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel, "Gated graph sequence neural networks," *arXiv preprint arXiv:1511.05493*, 2015.

[19] R. Li, S. Wang, F. Zhu, and J. Huang, "Adaptive graph convolutional neural networks," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[20] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" *arXiv preprint arXiv:1810.00826*, 2018.

[21] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[22] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700–4708.

[23] F. Yu and V. Koltun, "Multi-scale context aggregation by dilated convolutions," *arXiv preprint arXiv:1511.07122*, 2015.

[24] 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 *Proceedings of the 36th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. Long Beach, California, USA: PMLR, 09–15 Jun 2019, pp. 21–29. [Online]. Available: http://proceedings.mlr.press/v97/abu-el-haija19a.html

[25] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad, "Collective classification in network data," *AI magazine*, vol. 29, no. 3, pp. 93–93, 2008.

[26] M. Zitnik and J. Leskovec, "Predicting multicellular function through multi-layer tissue networks," *Bioinformatics*, vol. 33, no. 14, pp. i190–i198, 2017.

[27] A. Subramanian, P. Tamayo, V. K. Mootha, S. Mukherjee, B. L. Ebert, M. A. Gillette, A. Paulovich, S. L. Pomeroy, T. R. Golub, E. S. Lander *et al.*, "Gene set enrichment analysis: a knowledge-based approach for interpreting genome-wide expression profiles," *Proceedings of the National Academy of Sciences*, vol. 102, no. 43, pp. 15 545–15 550, 2005.

[28] X. Zhu, Z. Ghahramani, and J. D. Lafferty, "Semi-supervised learning using gaussian fields and harmonic functions," in *Proceedings of the 20th International conference on Machine learning (ICML-03)*, 2003, pp. 912–919.

[29] J. Weston, F. Ratle, H. Mobahi, and R. Collobert, "Deep learning via semi-supervised embedding," in *Neural Networks: Tricks of the Trade*. Springer, 2012, pp. 639–655.

[30] M. Belkin, P. Niyogi, and V. Sindhwani, "Manifold regularization: A geometric framework for learning from labeled and unlabeled examples," *Journal of machine learning research*, vol. 7, no. Nov, pp. 2399–2434, 2006.

[31] Q. Lu and L. Getoor, "Link-based classification," in *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, 2003, pp. 496–503.

[32] Z. Yang, W. W. Cohen, and R. Salakhutdinov, "Revisiting semi-supervised learning with graph embeddings," *arXiv preprint arXiv:1603.08861*, 2016.

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

[34] 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 *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 5115–5124.

[35] M. Fey and J. E. Lenssen, "Fast graph representation learning with PyTorch Geometric," in *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[36] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[37] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[38] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, 2010, pp. 249–256.