Applied Intelligence (2020) 50:3441–3451
https://doi.org/10.1007/s10489-020-01729-w

Hierarchical graph attention networks for semi-supervised node classification

Kangjie Li[1] · Yixiong Feng[1] · Yicong Gao[1] · Jian Qiu[2]

Published online: 10 June 2020

Abstract

Recently, there has been a promising tendency to generalize convolutional neural networks (CNNs) to graph domain. However, most of the methods cannot obtain adequate global information due to their shallow structures. In this paper, we address this challenge by proposing a hierarchical graph attention network (HGAT) for semi-supervised node classification. This network employs a hierarchical mechanism for the learning of node features. Thus, more information can be effectively obtained of the node features by iteratively using coarsening and refining operations on different hierarchical levels. Moreover, HGAT combines with the attention mechanism in the input and prediction layer. It can assign different weights to different nodes in a neighborhood, which helps to improve accuracy. Experiment results demonstrate that state-of-the-art performance was achieved by our method, not only on Cora, Citeseer, and Pubmed citation datasets, but also on the simplified NELL knowledge graph dataset. The sensitive analysis further verifies that HGAT can capture global structure information by increasing the receptive field, as well as the effective transfer of node features.

Keywords Graph convolutional networks · Hierarchical representation · Semi-supervisded

1 Introduction

Graphs can encode complex geometric structures that lie in the non-Euclidian domain. They can be studied with strong mathematical tools [1], and nowadays have become ubiquitous. For example, in e-commerce, to make accurate recommendations, it is necessary to exploit the interactions between users and products [2, 3]. In chemistry, a new drug can be discovered by using a graph-based learning method that models the molecules as a graph [4]. In citation networks, papers can be categorized into different groups through their citation graphs [5, 6]. Moreover, there are many unlabeled data in the real world and labeling data is sometimes unrealistic and time-consuming. The semi-supervised manner means that only a small amount of labeled data is used to train the model. Consequently, it is often crucial to analyze graphs in that situation, and the key issue is to maximize the effective utilization of the feature information of the unlabeled data [7].

As an approach to graph analysis, graph neural networks (GNNs) are closely related to graph embedding. Graph embedding [8] is a method that learns to represent graph nodes in low-dimensional vectors. Approaches such as word embedding [9], DeepWalk [10], node2vec [11] have achieved a breakthrough. However, because they are unsupervised algorithms, they cannot perform in an end-to-end fashion. The early GNNs employ a recursive mechanism. Gori [12] and Scarselli et al. [13] first introduced the operation of neural networks to the graph. It consists of a repeated application of propagation function until the node states equilibrium. As a result, it suffers a high cost of calculation. This problem is alleviated by Li et al. [14], who propose to use gated recurrent units in the propagation step. As deep learning achieves great success for Euclidean data [15, 16], there is a more promising way to generalize

✉ Yixiong Feng
fyxtv@zju.edu.cn

Kangjie Li
21825116@zju.edu.cn

Yicong Gao
gaoyicong@zju.edu.cn

Jian Qiu
qiujian@hdu.edu.cn

[1] State Key Laboratory of Fluid Power and Mechatronic Systems, Zhejiang University, Hangzhou, 310027, China

[2] School of Cyberspace, Hangzhou Dianzi University, Hangzhou, 310027, China

convolution to graph domain. However, the existing deep learning algorithm cannot be directly implemented to the irregular graph data. Specifically, each graph has a different size of node, and each node has a different size of neighbor. It means that some important operations, like convolution, cannot be applied directly. Furthermore, the core assumption that instances are independent of each other is not established in non-Euclidean space [17, 18].

Nevertheless, there are generally two categories of graph convolutional networks (GCNs) [19]: spatial-based and spectral-based. Spatial-based methods construct a new feature vector for each node using its neighborhood information. The convolution of the spectral-based methods is defined by decomposing the graph signal on the spectral domain and applying a spectral filter to the spectral components [20]. However the learned filter depends on the graph structure; the model trained cannot be directly applied to another graph. Consequently, the spatial-based methods begin to increase due to the ability to work with a different graph and weight sharing. More recently, graph attention networks (GATs) [5] have been developed by employing attention mechanisms in GCNs. They are more advanced by making the operator focus on the most relevant parts of the inputs.

In the present work, the unique architecture of a hierarchical graph attention network (HGAT) was proposed for semi-supervised node classification on the graph. Our work was based on GAT, which was introduced by Petar et al. [5]. We applied a hierarchical mechanism in our model, which could increase the receptive field of nodes and more effectively transfer node features. The HGAT consists of an input layer, a hierarchical layer, and a prediction layer. Specifically, every level in the hierarchical layer concludes two symmetrical calculations: the coarsening operation and the refining operation. After the coarsening operation, we got a smaller graph with hyper-nodes. It could reflect the local structure and help to exploit global information through overlying the operation. Before the refining operation, we concatenated the node representations of the coarsened graph with the next-level refined node representations. The refining operation was used to refine the graph structure of the previous level. Such a model can capture node information from most relevant neighbors, leading to a better node representation. To our best knowledge, the experiments show that our method had the best overall performance among different citation and knowledge graph datasets.

There are three main contributions of our work. First, for the first time, the attention mechanism was combined with the hierarchical representation learning on the graph, which could help capture global structure information. Second, the coarsening and refining operation based on the contraction sets were defined. Third, our method achieved a better performance than previous work in the semi-supervised node classification task. Notably, parameter sensitivity analysis showed that the proposed HGAT could get a larger receptive field, and more effectively transfer node features. All the experiment codes will be available at https://github.com/LeeKangjie after the review process.

The remaining sections are organized as follows. A review of the related work is given in Section 2. The methodology adopted in this paper is described in Section 3. Section 4 provides a comparative experiment and analysis. Finally, we summarize the paper in Section 5.

2 Related work

In this section, some previous works about the hierarchical representation learning on the graph and the graph convolutional networks are reviewed.

2.1 Hierarchical representation learning on graph

Some works have been raised to get global information using hierarchical models. Chen et al. [21] propose the hierarchical representation learning for networks (HARP). It works through finding a smaller graph that can approximate the global structure, then using the embedding method to learn the initial representation. Finally, it inductively embeds the hierarchy of graph from the coarsest one to the original graph. Homologous as Chen, Liang et al. [22] use a hybrid matching technique to maintain the backbone structure of the graph. Then they apply existing embedding methods on the coarsest graph and refines the embedding to the original graph. However, these two methods are both unsupervised, which doesn't use the node features. Besides they are both designed for large graph embedding, which introduces a huge computational overhead, not suitable to deal with the node classification task here. Hu et al. [23] propose hierarchical graph convolutional networks (HGCN) to increase the receptive field. It first repeatedly assigns nodes with similar structures into a hyper-node and then refines the representation for each node. However, each hierarchical level includes a GCN operation, which suffers from the Laplacian smoothing problem. Ying et al. [24] propose a differentiable graph pooling module that could generate hierarchical representations of graphs. The module can be combined with numerous graph neural network architectures in an end-to-end fashion. However they orient for link prediction and graph classification, therefore, cannot be directly applied in node classification tasks. Lv et al. [25] propose ant colony based multi-level network embedding (ACE) to preserve the features of hierarchical clustering structures. It coarsens the graph by an ant colony based algorithm, then

the embedding vectors are generated from multiple layers of the coarsened graph. The idea is very impressive and we can learn from it. But, the final vector can be very large and it cannot handle node classification tasks either. In conclusion, we can see that hierarchical is a useful trick that could help to get a better result if used appropriately.

2.2 Graph neural networks for semi-supervised learning

Advances in the graph convolutional networks are generally categorized as spectral approaches and spatial approaches. The spectral approaches have been successfully applied in node classification tasks. In the work of Bruna et al. [20], the eigendecomposition of the graph Laplacian is computed. Then, the convolution operation is defined, which opens a precedent for graph convolutional networks. Inspired by that, Henaff et al. [26] introduce a parameterization of the spectral filters. Then, the smooth coefficient is used to localize them in space. Because the computation of Laplacian eigenvector is computation inefficiently, Defferrard et al. [27] raise a method to avoid it. They build a K-localized ChebNet by approximating spectral filter with Chebyshev polynomials up to the K^{th} order. Later, Kipf et al. [6] simplify it by restricting the filters to the 1^{st} order, which means use the 1-step neighborhood information. Based on Kipf work, Zhuang et al. [28] raise a method to jointly consider global consistency and local consistency. However, all the spectral-based methods depend on the Laplacian eigenbasis of the given graph, which is not portable. In addition, the computation is non-parallel. These characteristics limit their development to some extent. As for spatial approaches, the main challenge is how to define the operator that can work with different-sized neighborhoods at the same time maintain the weight sharing properties. To achieve that, Duvenaud et al. [29] present a method through learning a specific weight matrix for each node degree in the molecular feature extraction task. Atwood et al. [30] present a model to do the node classification based on the diffusion-based representations. It defines the neighborhood using a transition matrix while learning weights for neighborhood degree and each input channel. Then Niepert et al. [31] present a general approach by extracting and normalizing locally connected regions to learn convolutional neural networks for an arbitrary graph. After that Hamilton [32] introduces GraphSAGE, a general inductive framework to generate node embedding. It operates by sampling the local neighborhood of nodes and perform aggregators over it. Besides that, to enable better structure-aware representation, Xu [33] explore a jumping knowledge network (JK-Net) to leverage different neighborhood ranges of each node. Based on GCNs, Petar et al. [5] propose GAT by introducing the attention mechanism. The architecture leverages a masked self-attentional layer to enable different weights to different nodes. By stacking layers, the node can also attend over their neighborhood features. It achieves state-of-art results among established benchmarks and allows for dealing with variable sized inputs. Due to these benefits, the attention mechanism is employed in our method.

3 Methodology

In this section, we define some notations used in the representation of HGAT architecture. Then we decompose the architecture into input layer, hierarchical layer, and prediction layer, and explain the implementation details.

3.1 Preliminaries

An undirected graph is represented by $G = (V, E)$, where V is the set of nodes (also known as vertices), E is the set of edges. The notation $||$ means the element number in a particular set. The notation v_i and e_i represent the i^{th} node and the i^{th} edge respectively. The adjacency matrix is represented by $A = [a_{ij}]$, which is nonnegative. Notation $D = diag(d_1, d_2, ..., d_n)$ is the degree matrix of A, where $d_i = \sum_j a_{ij}$ is the degree of v_i. The undirected graph is associated with node representation matrix $H^{n\times f}$ (also known as node features), where n is the total node number, f is the node features dimension.

3.2 HGAT architecture

As shown in Fig. 1, the workflow of our HGAT architecture is divided into three parts: input layer, hierarchical layer, and prediction layer. Specifically, the hierarchical layer includes l levels, each of which consists of two symmetrical calculations: the coarsening operation and the refining operation. We employed GAT as an input layer, combined with a multi-head attention mechanism [5] to stabilize the learning process. It takes the graph and the node representation H_0 as input, and outputs the node representation H_1. In the hierarchical layer, taking the i^{th} level as an example, the coarsening operation derives a coarsened graph G_{i+1} and node representation matrix H_{i+1}, which will be fed into the next level. Then, we concatenated H_{i+1} and next-level refined node representation matrix H^* resulting in H^*_{i+1}. Through putting H^*_{i+1} into the refining operation, we finally obtained the i^{th} level node representation matrix. After that, in order to classify each node, we employed a softmax classifier after the GAT. Outputting the classes of each node in the one-hot encoding. An example of the two-level hierarchical layer is shown in Fig. 2. After each coarsening operation, the graph becomes

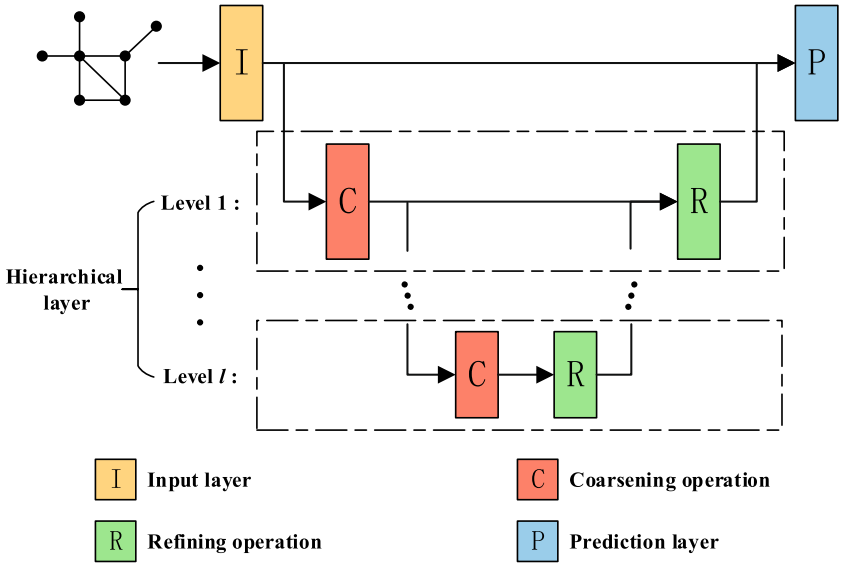

Fig. 1 The architecture of our method. From left to right, there are three layers: input, hierarchical, and prediction. The input layer employs multi-head GAT to learn the node representations. The hierarchical layer includes l levels, each of which has symmetric coarsening and refining operations. We employed a softmax classifier after the GAT in the last prediction layer

smaller. Before the concatenation, the smaller graph needs to be refined first.

We employed the hierarchical mechanism in the architecture of our model. Because it plays a key role in increasing the receptive field, it could help improve classification accuracy. At each hierarchical level, after the graph coarsening operation, the refining operation was used to help recover the graph structure. It would lead to an effective transfer of a node feature to the most relevant one. Through iteratively refining operations, a node can receive information from further places. More details can be found in the description of the hierarchical layer.

As for the running time of our algorithm, it was basically the same as GAT. GAT needs to compute attention coefficients between every two nodes, which demands numerous parameters to be learned. In the hierarchical layer

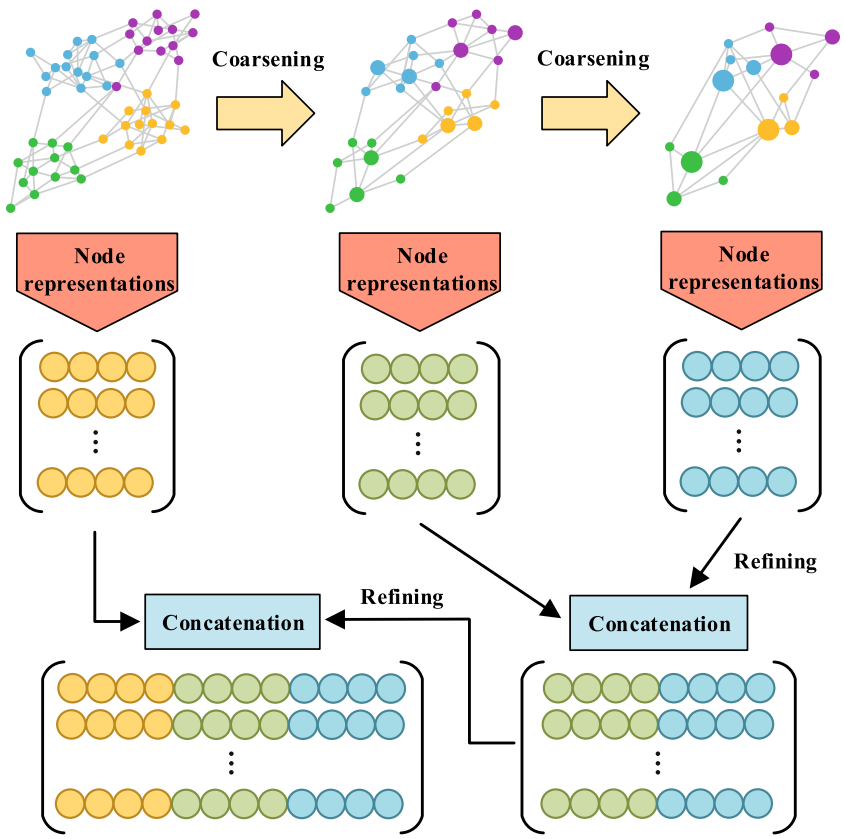

Fig. 2 Example of the two-level hierarchical layer. From left to right, the coarsened graphs appear with the corresponding node representations after the coarsening operation. Then through iteratively refining operation and concatenation, we get the final output

of our method, we only need to do a matrix multiplication, as shown in the subsequent section, in each level, which has no parameters to be learned. As a result, our method could improve accuracy without a big sacrifice of efficiency compared to GAT.

3.2.1 Input layer

Because of the promising performance of GAT, we employed attention mechanisms in the input layer. Moreover, we used a multi-head mechanism to stabilize the learning process. The layer took initial G_0 and H_0 as input, and the output node representation matrix H_1 was calculated as:

$$H_1 = \bigcup_{k=1}^{K} \sigma(\alpha^k W^k H_0) \tag{1}$$

where $\cup$ represents concatenation, K is the multi-head number, α^k is a normalized attention coefficient matrix, W^k is the corresponding transformation matrix [5], and σ () is the nonlinearity ELU function.

3.2.2 Hierarchical layer

The hierarchical layer involves l levels, as shown in Fig. 1, each level consisting of a coarsening operation and refining operation. The hierarchical layer is based on the assumption that nodes have similar connections are likely to share their features with each other.

Coarsening is a type of graph reduction, which can interpret the graph transformation using a set of constraints. We first expressed a surjective map from the node set V_i to V_{i+1} as φ_i, then defined the set of nodes V_i^r of V_i mapped onto the same node v_r of V_{i+1} as a contraction set, which is formulated in (2). We conducted the equivalent structure selection and similar structure selection one after another to construct all the contraction sets.

$$V_i^r = \{v \in V_i : \varphi_i(v) = v_r\} \tag{2}$$

During the equivalent structure selection, we selected nodes having the same neighbors to form contraction sets. Also, we can say nodes are equivalent if their corresponding rows in the adjacency matrix are identical. Each set corresponds to a hyper-node in the coarsened graph. For instance, as the example in Fig. 3 shows, nodes B and C were selected to form a contraction set.

Then, we performed a similar structure selection. Inspired by heavy edge matching [34], the connection strength between node v_j and v_k is defined as (3).

$$s_i(v_j, v_k) = \frac{A_i(v_j, v_k)}{\sqrt{D_i(v_j) D_i(v_k)}} \tag{3}$$

where A_i and D_i are adjacency matrix and degree matrix of graph i, respectively.

As nodes with fewer neighbors have fewer chances of being selected in the contraction set, we sorted all the nodes besides the contraction sets into ascending order of degrees to give these nodes higher priority in terms of selection. If nodes had the same degree, we sorted them into

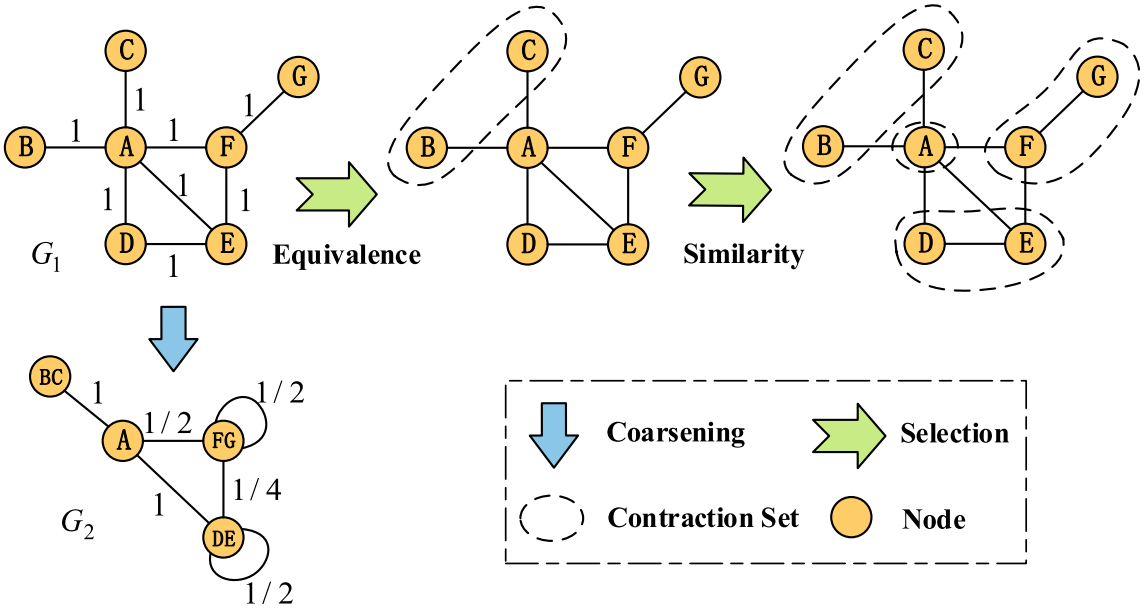

Fig. 3 Example of graph coarsening from G_1 to G_2. From left to right, we performed the equivalent structure selection and similar structure selection one after another to construct the contraction sets. From top to bottom, we used the contraction sets to derive the coarsened graph G_2. The adjacent matrix value of G_2 after being directly calculated using (6) is reflected on the edges

ascending order of their row number in the adjacent matrix. We iteratively picked up the node v_j besides the contraction sets and calculated the connection strength between all the neighbors besides the contraction sets. Then, we selected the pairs having the largest connection strength as a new contraction set. In particular, if a node didn't have neighbors besides the contraction sets, we selected it alone as a contraction set. Finally, every node would be selected in the contraction sets. As the example in Fig. 3 shows, after sorting the degree of node A, D, E, F, and G, we first picked up node G. Because it only had one neighbor F, we selected node F and G to form a contraction set. Then, we picked up the rest node which had a minimum degree, namely node D. After calculating the connection strength, we selected a pair (D, E) as a contraction set, for they had the largest value. Finally, because the single node A didnot have neighbors besides the contraction sets, it was selected as a contraction set itself.

To obtain the coarsened graph G_{i+1} from G_i including its node representation matrix H_{i+1} and adjacency matrix A_{i+1}, we defined our contraction matrix M_i based on all the contraction sets.

$$M_i(r,h) = \begin{cases} \dfrac{1}{|V_i^r|}, \text{if } v_h \in V_i^r, \\ 0, \text{otherwise.} \end{cases} \tag{4}$$

where r is the contraction set number, h is the node number in V_i, and V_i^r is a contraction set in G_i.

The node representation matrix H_{i+1} and adjacency matrix A_{i+1} of G_{i+1} are determined by (5) and (6) respectively.

$$H_{i+1} = M_i H_i \tag{5}$$

$$A_{i+1} = M_i A_i M_i^T \tag{6}$$

The graph coarsening operation can capture the global structure, and neglect some details. As the hierarchical level increased, the number of nodes decreased. At each hierarchical level, the coarsening techniques reduced the size of the graph by an approximate factor of two, which offers control over the size. We then introduced the refining operation to help recover its structure. The refinement meant calculating the node representation of the current graph from the node representation of its coarsened graph as illustrated in (7). It would average the node features once in the hyper-node, leading to an effective transfer of node features to the most relevant one. Through iteratively refining operations, a node could receive information from further places, that is, obtain a large receptive field.

$$H_{i+1} = M_{2l+1-i}^T H_i \tag{7}$$

where H_{i+1} is the refined graph node representation, and H_i is the original graph node representation.

Algorithm 1 summarizes the steps of hierarchical layer calculation. The hierarchical layer input H_1 and finally output H_{2l+1}.

Algorithm 1 The hierarchical layer calculation.

Input: Graph G_1, node representation H_1, hierarchical level l
Output: Node representation H_{2l+1}
for $i = 1...l$ **do**
  $C = \emptyset$ ▷ the contraction set collector
  Select all nodes having identical rows in adjacent matrix to form contraction sets V^S, $C = C \cup V^S$
  Sort nodes out of C in ascending order of degrees
  for each node v_j out of C **do**
    if v_j doesn't have adjacent nodes out of C **then**
      Select v_j as a contraction set V, $C = C \cup V$
    else
      for each adjacent v_k out of C **do**
        Calculate $s(v_j, v_k)$ according to (3)
      Select pair having the largest connection strength forming a contraction set V, $C = C \cup V$
  Use contraction sets in C to construct M_i via (4)
  Compute node representation H_{i+1} according to (5)
  Construct graph G_{i+1} using A_{i+1} according to (6)
for $i = l + 1...2l$ **do**
  Calculate node representation H_{i+1} using (7)
return H_{2l+1}

3.2.3 Prediction layer

Finally, we applied a softmax classifier after GAT to make the prediction.

$$H^{out} = softmax(\frac{1}{K}\sum_{k=1}^{K}\alpha^k W^k H_1^*) \tag{8}$$

where $H^{out} \in \mathbb{R}^{|V|\times|Y|}$ is the prediction of nodes belonging to the class $y_i \in |Y|$, and H_1^* is the concatenated node representation of H_1 and H_{2l+1}.

To train the proposed model for classification, the cross-entropy error on the labeled nodes was defined:

$$L = -\sum_{i\in Y_L}\sum_{j=1}^{|Y|} Z_{ij} \log H_{ij}^{out} \tag{9}$$

where Y_L represents the node indices that have labels, $Z \in \mathbb{R}^{|V|\times|Y|}$ is the mask matrix. Z_{ij} will be 1 if node i belongs to class j; otherwise, it will be 0.

Table 1 Summary of datasets used in experiments

	Cora	Citeseer	Pubmed	Simplified NELL
Type	Citation network			Knowledge graph
#Nodes	2708	3327	19717	9891
#Edges	5429	4732	44338	13142
#Classes	7	6	3	210
#Features	1433	3703	500	5414
Labeling rate	0.052	0.036	0.003	0.01

4 Experiment and analysis

In this section, the validity of our methodology is verified by presenting the results from several experiments. The experiments were implemented on a Windows 10 system with 4 GPUs and 32GB RAM. They are based on a TensorFlow 1.14.0 package and programmed with Python 3.7. Firstly, we present the datasets used in the semi-supervised node classification task. Then, we list the experiment parameters and compare the results with the previous methods. Finally, a sensitivity analysis is conducted.

4.1 Experiment setup

4.1.1 Datasets

The citation network datasets closely followed the experiment setup in Yang et al. [35]. Specifically, there were three citation network datasets: Cora, Citeseer and Pubmed. The nodes were documents and edges were citation links. Each document had a sparse bag-of-words feature vector and a class label. We also included a knowledge graph dataset: never-ending language learning (NELL). This is because the preprocessing of it needed more than 64GB memory, which was infeasible for us. To further verify the algorithm's robustness, we employed the simplified NELL provided by Zhuang et al. [28]. The details of the datasets are summarized in Table 1. The labeling rate refers to the ratio of labeled nodes to total nodes.

4.1.2 Parameter setting

The hyper-parameters were set as follows. The datasets were all trained for a maximum of 100,000 epochs using Adam [36] with a learning rate of 0.005 and early stopping with a window size of 100, which meant the training would stop if the validation loss didnot decrease for 100 consecutive epochs. The hyperparameters were optimized for each dataset. For the Cora dataset, the hierarchical level was 1, and the input layer consisted of 8 attention heads computing 8 features each. The prediction layer consisted of 1 attention head, with L_2 regularization of 0.001, and dropout of 0.6 applied to the layer input. The hyperparameter in Citeseer dataset was the same as Cora except for a higher L_2 regularization value of 0.004, and the hierarchical level was 2. For the Pubmed dataset, we chose a lower dropout rate of 0.4 and set the L_2 regularization to 0.002. The prediction layer consisted of 8 attention heads. The hierarchical level was 1. For the simplified NELL dataset, we computed 210 features for each head and set the L_2 regularization to 0.0003, and the hierarchical level to 2. The remaining parameter settings were the same as Cora.

4.2 Node classification results

The performance of our method was compared with the baseline methods in the semi-supervised node classification. Table 2 summarizes the experimental results over four datasets. With random weight initializations, we reported the mean accuracy of our method in an inductive manner of 20 runs.

The comparison results in Table 2 are very encouraging. The HGAT outperforms all the baseline methods to the best of our knowledge, which verifies the effectiveness of the hierarchical layer. Specifically, HGAT exceeds GAT on Cora, Citeseer, and Pubmed by 0.7%, 0.8%, and 0.8% respectively. For the simplified NELL dataset, our method outperformed the GAT by 2.9%. The improvement was much bigger than the other two. It demonstrated the significance of being able to learn global structure information by the effective transfer of node features. DeepWalk is a random-walk based algorithm, which cannot model the attribute information. That leads to

Table 2 Results of node classification in terms of accuracies for Cora, Citeseer, Pubmed and simplified NELL

Method	Cora	Citeseer	Pubmed	Simplified NELL
DeepWalk[10]	67.2%	43.2%	65.3%	28.2%
Planetoid[35]	75.7%	64.7%	77.2%	37.7%
GCN[6]	81.5%	70.3%	79.0%	38.0%
GAT[5]	$83.0 \pm 0.7\%$	$72.5 \pm 0.7\%$	$79.0 \pm 0.3\%$	$38.2 \pm 0.6\%$
HGAT	$83.7 \pm 0.3\%$	$73.3 \pm 0.4\%$	$79.8 \pm 0.1\%$	$41.1 \pm 0.5\%$

poor performance. Another random-walk based algorithm Planetoid had a relatively poor performance too, for it could not fully utilize the graph structure knowledge due to its random sampling strategies. To avoid this problem, the neighborhood aggregation scheme was employed by GCN. It produced node embedding by using a linear function on the graph Laplacian spectrum. However, the shallow model restricted the scale of the receptive field. As for GAT, it can be seen removing the hierarchical layer in our algorithm to some extent. It is relatively improved by employing the trick to assign different weights to different neighbors. However, it is still inferior to our method. In conclusion, as there were fewer training samples in the semi-supervised node classification task, the baseline method with a fixed receptive field was unable to transfer the feature to more nodes. Although there are overlaps of the variance intervals compared to GAT, it can be seen that our algorithm had a relatively low variance and high mean value. Making the hypothesis that our algorithm accuracy is less than GAT, the significance level of all datasets is less than 0.001, which means we have 99.9% confidence in saying that our method is more advanced than GAT. Our HGAT method is more promising in terms of improving accuracy than others by learning global information. Because, in the hierarchical layer, more receptive fields can be obtained, most importantly, node features can be effectively transferred in the hierarchical layer.

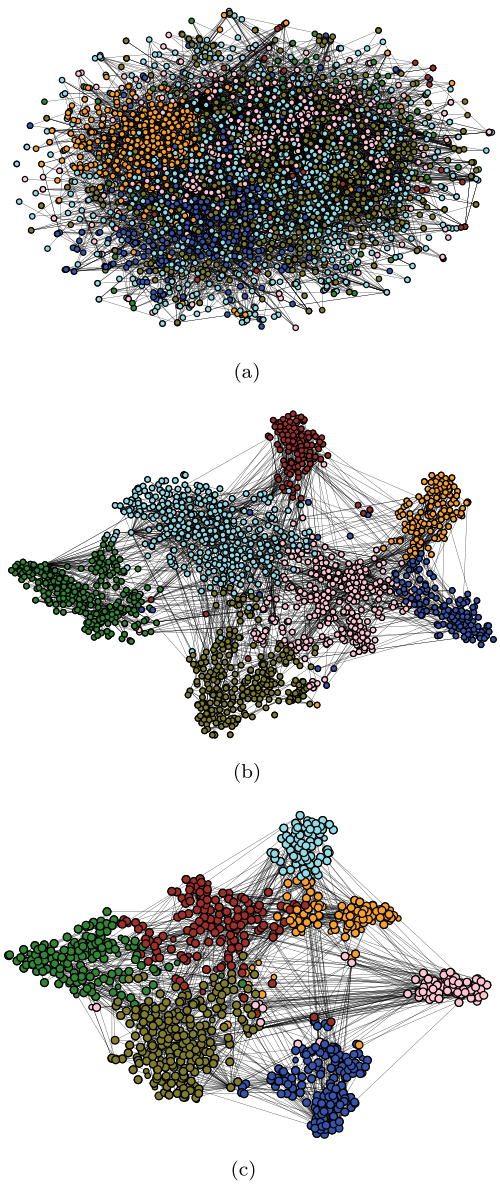

Fig. 4 Visualization of graphs on the Cora dataset. **a** t-SNE plot of the initial node representations. **b** t-SNE plot of node representations of the 1^{st} level hierarchical output. **c** t-SNE plot of node representations of the 2^{nd} level hierarchical output. The clusters are represented by different colors. The node size is proportional to the number of nodes it contains

For an intuitive understanding of the coarsening operation in the hierarchical layer, we made the t-SNE [37] plots of the node representations, as seen in Fig. 4. Different node colors correspond to different classes, each of which is clustered by applying K-means on the node representations. Also, there are seven different colors in the graph. The node size is proportional to the number of nodes it contains. Figure 4a is a visualization of the original Cora dataset features, different nodes being mixed together. Figure 4b is the visualization of the 1^{st} hierarchical level outputs. The total number of nodes is the same as Fig. 4a. However, we can see the cluster of different classes, which verifies the discriminative power across the seven topic classes of our algorithm. Figure 4c is the visualization of the 2^{nd} level output. As the hierarchical level increases, the graph size becomes smaller.

4.3 Parameter sensitivity

4.3.1 Lower labeling rate impact

As in reality, there are many situations that we cannot get more training data. It is important for the algorithm to work in this scenario. In this section, we decrease the labeling rate of two different class datasets: Citeseer and simplified NELL. And the performance of our method is compared to others. We decrease the labeled number of citation datasets per class from 20 to 15, 10 and 5, getting the labeling

Table 3 Results in terms of classification accuracies for Citeseer with different labeling rates

Method	0.036	0.027	0.018	0.009
GCN	70.3%	69.7%	68.6%	57.6%
GAT	72.5%	69.9%	68.8%	64.2%
HGAT	73.3%	70.6%	70.5%	67.4%

Table 4 Results in terms of classification accuracies for simplified NELL with different labeling rates

Method	0.01	0.006	0.003	0.001
GCN	38.0%	37.6%	22.0%	16.2%
GAT	38.2%	33.9%	23.3%	16.4%
HGAT	41.1%	38.3%	23.6%	21.8%

rate: 0.036, 0.027, 0.018 and 0.009 respectively. We use a simplified NELL dataset of labeling rate: 0.01, 0.006, 0.003 and 0.001 following the method of Zhuang et al. [28]. The corresponding average result of 20 runs is reported in Tables 3 and 4.

From the tables above, it can be seen that our method beats the baselines in different labeling rates. With the labeling percentage decreasing, the accuracy margin between HGAT and the best baseline method becomes bigger. Especially when the labeling rate is 0.009 on the Citeseer dataset, the accuracy of HGAT exceeds the GCN and GAT by 9.8% and 3.2% respectively. For the labeling rate of 0.001 in simplified NELL, the accuracy of HGAT exceeds the GCN and GAT by 5.6% and 5.4% respectively. With the labeling rate decreasing, the connections between the labeled and unlabeled nodes become fewer, that is, the number of edges used for propagation features are fewer. Only when the receptive field is large enough and the information from the unlabeled nodes is passed to the labeled one efficiently can we get a better result. Thus, the results prove that the proposed HGAT can get a larger receptive field and have a more effective node features transfer. In other words, by introducing the hierarchical layer, our method is more robust in much lower training data situations.

4.3.2 Effects of the hierarchical level

The hierarchical level is a key factor in Algorithm 1. A too high value will produce a "smoothing" effect, causing an inferior result. A too low value will not use larger receptive field information. For a better understanding of the hierarchical effect, we analyzed the accuracy of two different labeling rates with different hierarchical levels as shown in Fig. 5.

It can be seen from Fig. 5, before the peak point, that the accuracy grows as the hierarchical level increases. Then the

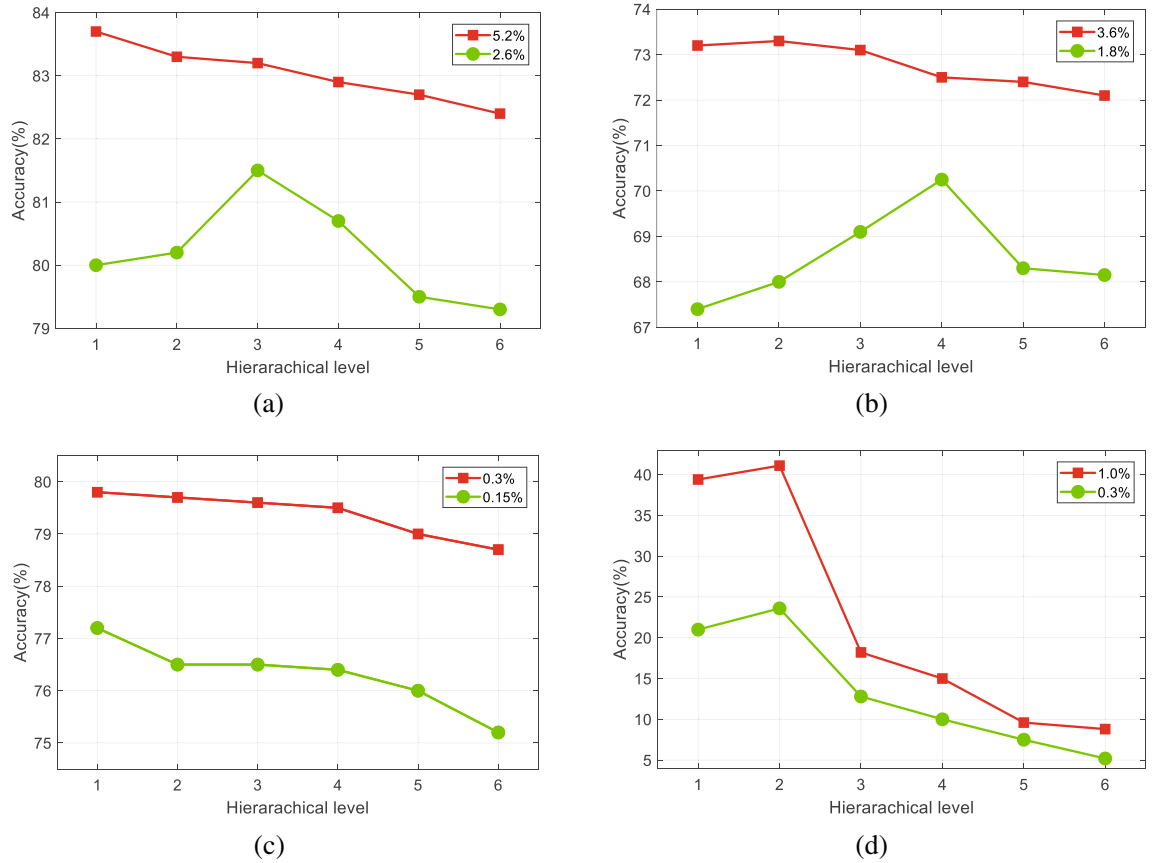

Fig. 5 Results of HGAT with varying hierarchical levels in terms of accuracies on (**a**) Cora dataset, **b** Citeseer dataset, **c** Pubmed dataset, and **d** simplified NELL dataset. Two labeling rates of different datasets are shown in the legend

accuracy falls with a higher hierarchical level. This could be explained by the fact that a higher hierarchical level helps capture the useful node features by increasing the receptive field. Nevertheless, too high levels lead to a bad feature because of the smoothing effect. The best hierarchical level for Cora and Citeseer becomes larger as the labeling rate decreases. It moves from 1 to 3 for the Cora dataset and moves from 2 to 4 for the Citeseer dataset. However, the best hierarchical levels for Pubmed and simplified NELL are the same in two different labeling rates. This could be explained that the labeling rate for Pubmed is extremely sparse, and the advantages of the hierarchical mechanism are not obvious in this case. Although the labeling rate of simplified NELL is the same order of magnitude as Citeseer, there are many more classes in their classification tasks. On average, for each class, the training number is still extremely sparse.

5 Conclusions and future work

In this work, we presented a novel hierarchical graph attention network for semi-supervised node classification. Through employing a hierarchical layer, the larger receptive field of nodes could be obtained, and node features could be effectively transferred. Besides, our method didnot need a costly matrix operation. It could be parallelized across all nodes. The results show that our method achieved state-of-the-art performance on four different datasets. There are several possible improvements that could be addressed in future work. Firstly, our method could be extended to other interesting tasks like graph classification, which is useful in practice. Secondly, many other datasets like text could also be treated as graphs. How to implement our method to these datasets is expected to be explored. Finally, our method cannot be directly applied to the directed graph, which is also due to be improved.

Acknowledgements This work is supported by the National Key R&D Program of China (No. 2018YFB1701702), the National Natural Science Foundation of China (Nos. 51675477, 51775489), Zhejiang Provincial Natural Science Foundation of China (No. LZ18E050001).

References

1. Shuman DI, Narang SK, Frossard P, Ortega A, Vandergheynst P (2013) The emerging field of signal processing on graphs: extending high-dimensional data analysis to networks and other irregular domains. IEEE Signal Proc Mag 30(3):83–98
2. Berg RVD, Kipf TN, Welling M (2017) Graph convolutional matrix completion. arXiv:1706.02263
3. Ying R, He R, Chen K, Eksombatchai P, Hamilton WL, Leskovec J (2018) Graph convolutional neural networks for web-scale recommender systems. In: Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery and data mining, pp 974–983
4. Gilmer J, Schoenholz SS, Riley PF, Vinyals O, Dahl GE (2017) Neural message passing for quantum chemistry. In: Proceedings of the 34th international conference on machine learning, pp 1263–1272
5. Veličković P, Cucurull G, Casanova A, Romero A, Lio P, Bengio Y (2018) Graph attention networks. In: 6th International conference on learning representations, pp 1–12
6. Kipf TN, Welling M (2018) Semi-supervised classification with graph convolutional networks. In: 5th International conference on learning representations, pp 1–14
7. Li Q, Han Z, Wu XM (2018) Deeper insights into graph convolutional networks for semi-supervised learning. In: Thirty-Second AAAI conference on artificial intelligence, pp 1–9
8. Zhou J, Cui G, Zhang Z, Yang C, Liu Z, Sun M (2019) Graph neural networks: a review of methods and applications. arXiv:1812.08434v3
9. Mikolov T, Chen K, Corrado G, Dean J (2013) Efficient estimation of word representations in vector space. arXiv:1301.3781
10. Perozzi B, Al-Rfou R, Skiena S (2014) DeepWalk: online learning of social representations. In: Proceedings of the 20th ACM SIGKDD international conference on knowledge discovery and data mining, pp 701–710
11. Grover A, Leskovec J (2016) node2vec: scalable feature learning for networks. In: Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining, pp 855–864
12. Gori M, Monfardini G, Scarselli F (2005) A new model for learning in graph domains. In: Proceedings of 2005 IEEE international joint conference on neural networks, pp 729–734
13. Scarselli F, Gori M, Tsoi AC, Hagenbuchner M, Monfardini G (2008) The graph neural network model. IEEE T Neural Networ 20(1):61–80
14. Li Y, Tarlow D, Brockschmidt M, Zemel R (2016) Gated graph sequence neural networks. In: 4th International conference on learning representations, pp 1–20
15. LeCun Y, Bengio Y (1995) Convolutional networks for images, speech, and time series. Handbook Brain Theory Neural Netw 3361:10
16. Hochreiter S, Schmidhuber J (1997) Long short-term memory. Neural Comput 9(8):1735–1780
17. Wu Z, Pan S, Chen F, Long G, Zhang C, Yu PS (2019) A comprehensive survey on graph neural networks. arXiv:1901.00596v3
18. Gao H, Wang Z, Ji S (2018) Large-scale learnable graph convolutional networks. In: Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery and data mining, pp 1416–1424
19. Bronstein MM, Bruna J, LeCun Y, Szlam A, Vandergheynst P (2017) Geometric deep learning: going beyond Euclidean data. IEEE Signal Proc Mag 34(4):18–42
20. Bruna J, Zaremba W, Szlam A, LeCun Y (2013) Spectral networks and locally connected networks on graphs. arXiv:1312.6203
21. Chen H, Perozzi B, Hu Y, Skiena S (2018) Harp: hierarchical representation learning for networks. In: Thirty-Second AAAI conference on artificial intelligence, pp 2127–2134
22. Liang J, Gurukar S, Parthasarathy S (2018) Mile: A multi-level framework for scalable graph embedding. In: The United States conference of Mayors' 86th Winter Meeting, pp 1–11
23. Hu F, Zhu Y, Wu S, Wang L, Tan T (2019) Hierarchical graph convolutional networks for semi-supervised node classification. In: Proceedings of the Twenty-Eighth international joint conference on artificial intelligence, pp 1–8
24. Ying Z, You J, Morris C, Ren X, Hamilton W, Leskovec J (2018) Hierarchical graph representation learning with differentiable pooling. In: Advances in neural information processing systems 31: annual conference on neural information processing systems, pp 4800–4810

25. Lv J, Zhong J, Liang J, Yang Z (2019) ACE: ant colony based multi-level network embedding for hierarchical graph representation learning. IEEE Access 7:73970–73982
26. Henaff M, Bruna J, LeCun Y (2015) Deep convolutional networks on graph-structured data. Comput Sci, 1–10
27. Defferrard M, Bresson X, Vandergheynst P (2016) Convolutional neural networks on graphs with fast localized spectral filtering. In: Advances in neural information processing systems 29: annual conference on neural information processing systems, pp 3844–3852
28. Zhuang C, Ma Q (2018) Dual graph convolutional networks for graph-based semi-supervised classification. In: Proceedings of the 2018 World Wide Web conference, pp 23–27
29. Duvenaud DK, Maclaurin D, Iparraguirre J, Bombarell R, Hirzel T, Aspuru-Guzik A, Adams RP (2015) Convolutional networks on graphs for learning molecular fingerprints. In: Advances in neural information processing systems 28: annual conference on neural information processing systems, pp 7–12
30. Atwood J, Towsley D (2016) Diffusion-convolutional neural networks. In: Advances in neural information processing systems 29: annual conference on neural information processing systems, pp 1993–2001
31. Niepert M, Ahmed M, Kutzkov K (2016) Learning convolutional neural networks for graphs. In: 33rd International conference on machine learning, pp 19–24
32. Hamilton W, Ying Z, Leskovec J (2017) Inductive representation learning on large graphs. In: Advances in neural information processing systems 30: annual conference on neural information processing systems, pp 1024–1034
33. Xu K, Li C, Tian Y, Sonobe T, Kawarabayashi KI, Jegelka S (2018) Representation learning on graphs with jumping knowledge networks. In: 35th International conference on machine learning, pp 1–14
34. Karypis G, Kumar V (1998) Multilevel k-way partitioning scheme for irregular graphs. J Parallel Distr Com 48(1):96–129
35. Yang Z, Cohen WW, Salakhutdinov R (2016) Revisiting semi-supervised learning with graph embeddings. In: Proceedings of the 33th international conference on machine learning, pp 1–9
36. Kingma DP, Ba J (2015) Adam: a method for stochastic optimization. In: 3rd International conference on learning representations, pp 1–15
37. Maaten LVD, Hinton G (2008) Visualizing data using t-SNE. J Mach Learn Res 9:2579–2605

Kangjie Li received the B.S. degree in mechanical engineering from China University of Mining and Technology, Xuzhou, China in 2018. He is currently a graduate student in Zhejiang University, Hangzhou, China. His research interest include design and optimization of mechanical structure, machine learning, and graph convolutional neural network and its application.

Yixiong Feng received the B.S. and M.S. degree in mechanical engineering from Yanshan University, Qinhuangdao, China, in 1997 and 2000, the Ph.D. degrees in mechanical engineering from Zhejiang University, Hangzhou, China, in 2004, respectively.

He is currently a Professor with the Department of Mechanical Engineering of Zhejiang University, China and the member of the State Key Lab of Fluid Power Transmission and Control of Zhejiang University, China. His research focuses on mechanical product design theory, intelligent automation and advance manufacture technology.

Yicong Gao received the B.S. degree in mechanical engineering from East China University of Science and Technology, Shanghai, China, in 2005, the Ph.D. degrees in mechanical engineering from Zhejiang University, Hangzhou, China, in 2011, respectively.

He is currently a Associate Professor with the Department of Mechanical Engineering of Zhejiang University, China and the member of the State Key Lab of Fluid Power Transmission and Control of Zhejiang University, China. His research focuses on mechanical product design theory, intelligent automation and advance manufacture technology.

Jian Qiu received BEng degree in Zhejiang University in 2005, MSc degree in the Department of Electronics, University of York in 2006 and PhD degree in the Department of Electronics, University of York in 2010.

He is now an Associate Professor in the School of Cyberspace, Hangzhou Dianzi University. His main research interests are in wireless sensor network, internet of things, and wireless network protocol design.