Frontiers of Information Technology & Electronic Engineering
www.jzus.zju.edu.cn; engineering.cae.cn; www.springerlink.com
ISSN 2095-9184 (print); ISSN 2095-9230 (online)
E-mail: jzus@zju.edu.cn

NGAT: attention in breadth and depth exploration for semi-supervised graph representation learning*

Jianke HU, Yin ZHANG†‡

College of Computer Science and Technology, Zhejiang University, Hangzhou 310027, China

†E-mail: yinzh@zju.edu.cn

Received Nov. 22, 2020; Revision accepted Jan. 10, 2021; Crosschecked Oct. 21, 2021; Published online Jan. 24, 2022

Abstract: Recently, graph neural networks (GNNs) have achieved remarkable performance in representation learning on graph-structured data. However, as the number of network layers increases, GNNs based on the neighborhood aggregation strategy deteriorate due to the problem of oversmoothing, which is the major bottleneck for applying GNNs to real-world graphs. Many efforts have been made to improve the process of feature information aggregation from directly connected nodes, i.e., breadth exploration. However, these models perform the best only in the case of three or fewer layers, and the performance drops rapidly for deep layers. To alleviate oversmoothing, we propose a nested graph attention network (NGAT), which can work in a semi-supervised manner. In addition to breadth exploration, a k-layer NGAT uses a layer-wise aggregation strategy guided by the attention mechanism to selectively leverage feature information from the k^{th}-order neighborhood, i.e., depth exploration. Even with a 10-layer or deeper architecture, NGAT can balance the need for preserving the locality (including root node features and the local structure) and aggregating the information from a large neighborhood. In a number of experiments on standard node classification tasks, NGAT outperforms other novel models and achieves state-of-the-art performance.

Key words: Graph learning; Semi-supervised learning; Node classification; Attention
https://doi.org/10.1631/FITEE.2000657 **CLC number:** TP391

1 Introduction

Recently, deep learning architectures such as convolutional neural networks (CNNs) have been applied with great improvement in computer vision tasks (Krizhevsky et al., 2012; Simonyan and Zisserman, 2014; He et al., 2016). However, unlike speech, image, and video data that lie on the Euclidean domains, many machine learning tasks involve data that cannot be represented in a grid structure, e.g., social networks and biological networks, where the entities are interdependent through multiple relationships. Such data are normally represented in the form of graphs, in which entities and relations correspond to nodes and edges, respectively.

In recent years, researchers have become more interested in applying deep learning architectures to graph domains, such as Random Walks algorithms (Perozzi et al., 2014; Grover and Leskovec, 2016; Ribeiro et al., 2017), which use the topology of the graph to directly train the individual node embedding, but discard node features and label information. Many other approaches extend the convolutional operator to graphs. For example, spectral graph convolutional networks (Bruna et al., 2014; Defferrard et al., 2016) design the convolutional layer based on graph spectrum analysis, while non-spectral networks (Niepert et al., 2016; Atwood and Towsley, 2016; Hamilton et al., 2017; Kipf and Welling, 2017) follow a feature passing scheme (or neighborhood aggregation) and achieve state-of-the-

‡ Corresponding author

* Project supported by China Knowledge Centre for Engineering Sciences and Technology (CKCEST)

ORCID: Jianke HU, https://orcid.org/0000-0001-9902-8748; Yin ZHANG, https://orcid.org/0000-0001-6986-4227

art performance in many graph-structured tasks such as node classification and link prediction.

Theoretically, a single graph convolutional layer serves as a localized node feature aggregator in the first-order neighborhood (Kipf and Welling, 2017). Similar to CNNs, stacking multiple graph convolutional layers can yield a larger receptive field. In semi-supervised settings, as depicted in Fig. 1, where the labeled nodes are very sparsely distributed, a larger receptive field allows the unlabeled nodes to benefit from the label information of further nodes. However, current work (Li et al., 2018; Klicpera et al., 2019) shows that neighborhood aggregation is equivalent to Laplacian smoothing, and that excessively increasing the range of neighborhood aggregation results in the problem of oversmoothing. Oversmoothing causes each node in a connected component to converge to a similar embedding, which means that the features learned by the graph neural networks (GNNs) become indistinguishable. In nature, the learned node embedding is not suitable to describe the node's local information. Hence, with deeper layers, graph convolutional networks (GCNs) might show worse performance. The issue occurs in many GNN models using such a neighborhood aggregation strategy, and becomes the major bottleneck for these models to be scalable to large graphs.

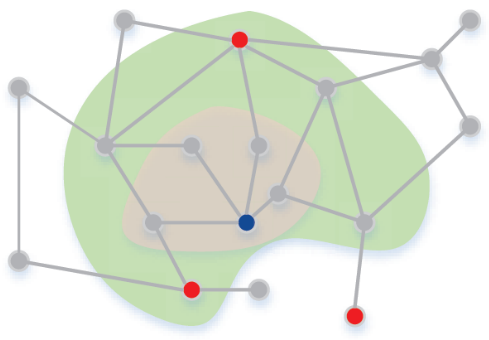

Fig. 1 The first- and second-order neighborhood of the root node (drawn in blue) in an undirected label-sparse graph. The unlabeled nodes are drawn in grey and the labeled nodes are drawn in red

Since the neighbors of each node might not be equally important, some interesting work (Veličković et al., 2018; Lee et al., 2019) uses attention mechanisms to construct a dynamic neighborhood in each layer to improve the breadth exploration (feature passing among neighboring nodes). Recently, Knyazev et al. (2019) performed analysis on the interpretability of GNN models that incorporate the attention mechanism, and described how the attention works.

However, attention-based GNNs still encounter the problem of oversmoothing. Therefore, we try to alleviate this model degradation, and propose an adaptive neighborhood aggregation algorithm inspired by previous achievements. In addition to using the attention mechanism in breadth exploration, we are more concerned about the information aggregation in depth exploration. A k-layer GNN can obtain the information in the k^{th}-order neighborhood. To efficiently leverage this information from different depths, we collect the hidden embedding that is learned by each intermediate layer and assign different weights to the corresponding layers. Thus, each node can enhance the attention bias against different neighborhood depths, and fuse the features to learn the depth-adaptive embedding. Evaluation of the node classification benchmarks shows the representation learning ability of our model. In summary, the main contributions of this paper are as follows:

1. We propose an end-to-end deep architecture for graph representation learning. It directly accepts graphs as inputs without any feature engineering, and provides high-quality node embedding as outputs.

2. We propose a novel neighborhood aggregation algorithm to alleviate the problem of oversmoothing. As far as we know, this is the first attempt to design a depth-wise feature aggregation scheme in self-attention fashion.

3. The experimental results on a number of semi-supervised node classification datasets show that our nested graph attention network (NGAT) is highly competitive compared to other novel models for node classification.

2 Preliminaries

We begin by introducing our notations. Formally, let $\mathcal{G}=\{\mathcal{V},\mathcal{E},\boldsymbol{A}\}$ denote an undirected graph, which consists of a finite set of nodes $\mathcal{V}$ with $|\mathcal{V}|=N$, a set of edges $\mathcal{E}$, and a symmetric adjacency matrix $\boldsymbol{A}$ (its entry $A_{i,j}$ represents the weight of an edge $e=(v_i,v_j)\in\mathcal{E}$ connecting the nodes $v_i,v_j\in\mathcal{V}$; when there is no edge between nodes i and j, $A_{i,j}=0$). We use $\boldsymbol{X}\in\mathbb{R}^{N\times d_i}$ to represent the node feature matrix, and $\boldsymbol{X}_v$ represents the d_i-dimensional feature vector of node v. $\mathcal{N}(v) = \{u \in \mathcal{V} | (v, u) \in \mathcal{E}\}$ denotes a neighborhood function that maps to the set of neighboring nodes of v, and $\tilde{\mathcal{N}}(v) = \mathcal{N}(v) \cup \{v\}$ maps to the neighbors and node v itself.

2.1 Semi-supervised graph learning

Collecting labeled data is very time-consuming. Semi-supervised learning methods address this problem by maximizing the use of unlabeled data, together with the labeled data, to build a robust model. In graph domains, we use $\boldsymbol{X}_L$ and Y_L to represent the feature matrix and labels, respectively, for labeled nodes. A commonly applied semi-supervised loss function with an explicit graph-based regularization is as follows:

$$\mathcal{L}(\boldsymbol{X}, Y_L) = \mathcal{L}_{\text{sup}}(\boldsymbol{X}_L, Y_L) + \lambda \mathcal{L}_{\text{reg}}(\boldsymbol{X}).$$

Here, $\mathcal{L}_{\text{sup}}$ denotes a supervised loss function, λ is a weighing factor, and $\mathcal{L}_{\text{reg}}$ is a regularization term. Label information is smoothed over the graph via some form of graph-based regularization, e.g., the graph Laplacian regularization: $\mathcal{L}_{\text{reg}}(\boldsymbol{X}) = f^{\text{T}}(\boldsymbol{X}) \boldsymbol{\Delta} f(\boldsymbol{X})$, where $f(\cdot)$ can be a differentiable embedding function, $\boldsymbol{\Delta} = \boldsymbol{D} - \boldsymbol{A}$ is the unnormalized graph Laplacian matrix, and $\boldsymbol{D}$ is a diagonal degree matrix with entry $D_{i,i} = \sum_j A_{i,j}$. There are many pioneer algorithms based on graph Laplacian regularization such as label propagation (Zhu et al., 2003), where the node's label propagates to neighboring nodes according to their proximity using the graph Laplacian matrix. Manifold regularization (Belkin et al., 2006) combines a support vector machine (SVM) classifier and Laplacian regularized least squares to compute the supervised loss. Other standard approaches that use graph regularization in semi-supervised graph learning have been summarized in Chapelle et al. (2009).

2.2 Graph neural networks

GNNs are deep architectures applied to graph domains, and have been widely used in many graph-data tasks. For the node classification task which aims to predict the node's label using the node features and graph structure, GNNs are designed to learn an embedding function $f : \mathcal{V} \to \mathbb{R}^{d_{\text{h}}}$ to generate a d_{h}-dimensional node embedding Z_v. Then, a classifier accepts Z_v as the input to predict the corresponding label y_v of node v $(\forall v \in \mathcal{V})$. Modern GNNs follow a neighborhood aggregation strategy, where they iteratively update the embedding of a node by aggregating the embeddings of its neighbors. After k iterations of aggregation, the final embedding of a node captures the features of neighboring nodes and structural information within the k^{th}-order neighborhood. For a k-layer GNN, we use $h_v^{(l)}$ to indicate the embedding of node v learned by the l^{th} hidden layer, and commonly, the embedding $h_v^{(k)}$ of the last layer is used as Z_v for downstream tasks.

3 Related work

We have presented several traditional approaches on semi-supervised graph representation learning in Section 2. In this section, we provide a brief overview of the recent advancements in using deep neural networks that are related to our work.

3.1 Graph convolutional network

GCN (Kipf and Welling, 2017) is a powerful neural network that achieves state-of-the-art performance in semi-supervised node classification on graph-structured data. A typical case of GCN stacks two graph convolutional layers as follows:

$$Z = \text{softmax}\left(\hat{\boldsymbol{A}}\, \sigma\left(\hat{\boldsymbol{A}} \boldsymbol{X} \boldsymbol{W}^{(0)}\right) \boldsymbol{W}^{(1)}\right),$$

where $\boldsymbol{W}^{(0)}$ and $\boldsymbol{W}^{(1)}$ are learnable weight matrices trained over all labeled nodes. $\sigma(\cdot)$ is a non-linear activation function such as $\text{ReLU}(x) = \max(0, x)$, applied in an entry-wise manner. The softmax function is defined as $\text{softmax}(x_i) = \frac{1}{\mathcal{Z}} \exp(x_i)$ with $\mathcal{Z} = \sum_i \exp(x_i)$. Furthermore, $\hat{\boldsymbol{A}}$ denotes the normalized symmetric adjacency matrix:

$$\hat{\boldsymbol{A}} = \tilde{\boldsymbol{D}}^{-1/2} \tilde{\boldsymbol{A}} \tilde{\boldsymbol{D}}^{-1/2},$$

where $\tilde{\boldsymbol{A}} = \boldsymbol{A} + \boldsymbol{I}_N$ and $\tilde{D}_{i,i} = \sum_j \tilde{A}_{i,j}$ with added self-loops. In detail, the neighborhood aggregation strategy for each node is motivated via a first-order approximation of the local spectral filters:

$$h_v^{(l+1)} = \sigma\left(\sum_{u \in \tilde{\mathcal{N}}(v)} \frac{1}{\sqrt{(d(v)+1)(d(u)+1)}} h_u^{(l)} \boldsymbol{W}^{(l)}\right),$$

where $d(v)$ measures the degree of node v $(\forall v \in \mathcal{V})$.

3.2 Graph attention network

GCN-like architectures use a fixed adjacency matrix $\hat{\boldsymbol{A}}$ in every graph convolutional layer, and ignore the difference between different neighboring nodes. Veličković et al. (2018) believed that designing an aggregation strategy that is aware of this imbalance can make GNNs more powerful, and that building a dynamic adjacency matrix also alleviates the problem of oversmoothing.

Graph attention network (GAT) designs a feedforward network to compute the attention value $e_{v,u}$ between nodes v and u:

$$e_{v,u} = \mathrm{LeakyReLU}\left(\left(h_v^{(l)}\boldsymbol{W}^{(l)} \| h_u^{(l)}\boldsymbol{W}^{(l)}\right)\boldsymbol{W}_\mathrm{a}\right),$$

where $\boldsymbol{W}_\mathrm{a}$ is a weight vector that is shared over all nodes. The attention value $e_{v,u}$ is regarded as the edge weight between nodes v and u, and it is recomputed in each layer; therefore, the adjacency matrix is dynamic.

Almost at the same time of the publication of GAT, there is a similar work, attention-based graph neural network (Thekumparampil et al., 2018), that uses a simpler attention mechanism but matches the state-of-the-art performance in some node classification benchmarks.

3.3 Other GNNs

Besides the two kinds of neural networks mentioned above, many other GNNs have been proposed recently. FastGCN (Chen et al., 2018) samples the nodes in each layer according to the importance distribution. GIN (Xu et al., 2019) uses a sum aggregator over neighboring nodes and a multi-layer perceptron after aggregating node features. SGC (Wu F et al., 2019) removes the non-linear activation between consecutive layers and collapses the weight matrices. LanczosNet (Liao et al., 2019) leverages the Lanczos algorithm to approximate the graph Laplacian to collect multi-scale information from graph convolutions. The thorough survey of graph neural networks can be found elsewhere (Zhou et al., 2018; Wu ZH et al., 2019).

4 Proposed method

In this section, we present the architecture of the NGAT, which deploys the node-wise and layer-wise attention mechanisms. The schematic layout of NGAT is shown in Fig. 2. Algorithm 1 presents the process for generating the node embedding using the NGAT that has already been trained, and the parameters are fixed.

4.1 Node-wise attention

We start with an embedding layer to map node features into a low-dimensional vector space, so that the inherent features of the nodes are visible in

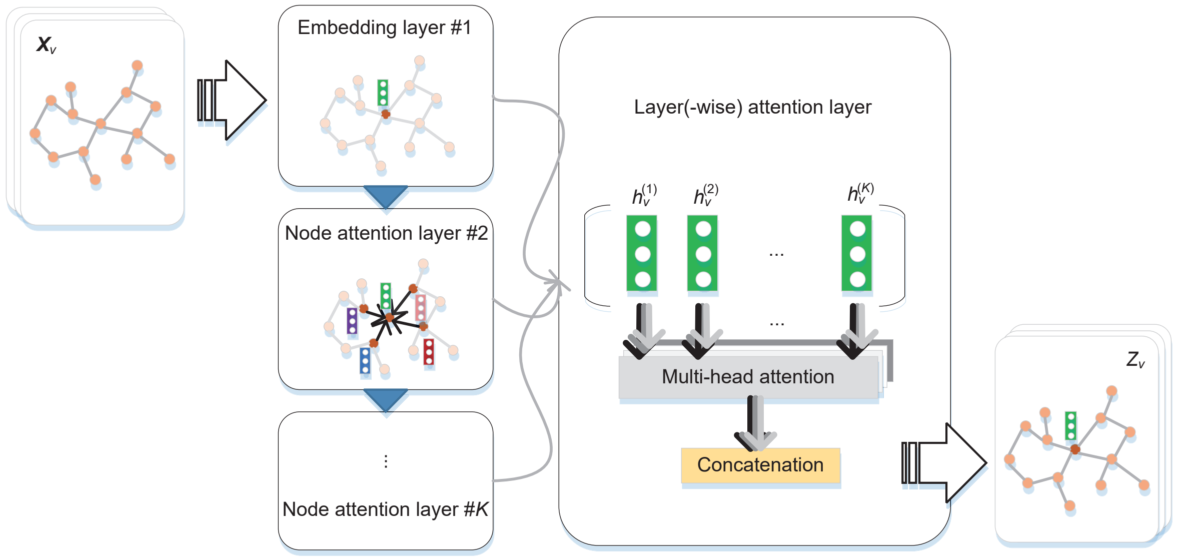

Fig. 2 Schematic layout of a K-layer nested graph attention network (NGAT) for graph representation learning. The architecture consists of three parts: (1) the embedding layer does a linear transformation with non-linear activation; (2) the node attention layer aggregates the feature information from neighbors in attention fashion; (3) the layer(-wise) attention layer selectively aggregates the feature information from different depths, and outputs the final embedding Z_v

Algorithm 1 Embedding algorithm of NGAT

```
Input: graph $\mathcal{G}$, node features $\boldsymbol{X}_v$, neighborhood function with self-loops $\tilde{\mathcal{N}}(v), \forall v \in \mathcal{V}$
Model information: depth $K$, parameter $\boldsymbol{W}$, node-wise aggregation function N-ATTN$(\cdot)$, and layer-wise L-ATTN$(\cdot)$
Output: node embedding $Z_v$
1: $h_v^{(1)} \leftarrow \sigma\left(\boldsymbol{X}_v \boldsymbol{W}^{(0)}\right), \forall v \in \mathcal{V}$
2: for $l = 1, 2, \ldots, K-1$ do
3:     for $v \in \mathcal{V}$ do
4:         $h_v^{(l+1)} \leftarrow$ N-ATTN$\left(\left\{h_u^{(l)}, \forall u \in \tilde{\mathcal{N}}(v)\right\}\right)$
5:     end for
6: end for
7: $h_v^{(K+1)} \leftarrow$ L-ATTN$\left(h_v^{(1)}, h_v^{(2)}, \ldots, h_v^{(K)}\right)$
8: $Z_v \leftarrow h_v^{(K+1)}, \forall v \in \mathcal{V}$
```

layer-wise attention, as will be introduced in Section 4.2. It can be expressed over all nodes as a simple matrix multiplication:

$$\boldsymbol{H}^{(1)} = \sigma\Big(\boldsymbol{X}\boldsymbol{W}^{(0)}\Big), \tag{1}$$

where $\boldsymbol{X} \in \mathbb{R}^{N \times d_i}$ is the node feature matrix, the weight matrix $\boldsymbol{W}^{(0)} \in \mathbb{R}^{d_i \times d_\mathrm{h}}$ is trainable as a part of the whole model, and d_h is the number of hidden units.

A node-wise attention layer aims to improve the breadth exploration. It proceeds along lines 3–5 in Algorithm 1. The node attention mechanism is denoted by the placeholder N-ATTN. In each layer, node embeddings are updated in two stages: attention coefficient calculation and feature passing. We describe each stage in detail.

1. Attention coefficient calculation

We draw inspiration from Thekumparampil et al. (2018), which used cosine similarity to measure the correlation between different nodes. A learnable linear transformation is applied to improve the expressivity, as follows:

$$e_{v,u}^{(l)} = \frac{h_v^{(l)}\boldsymbol{W}^{(l)} \cdot h_u^{(l)}\boldsymbol{W}^{(l)}}{\|h_v^{(l)}\boldsymbol{W}^{(l)}\|\|h_u^{(l)}\boldsymbol{W}^{(l)}\|}, \tag{2}$$

where $\boldsymbol{W}^{(l)} \in \mathbb{R}^{d_\mathrm{h} \times d_\mathrm{h}}$ is a trainable weight matrix. Compared to GAT which uses multi-layer perception (MLP) to calculate the attention value, cosine similarity is more economic, resulting in substantial savings in the gradient computation, as it does not need new parameters. To normalize these coefficients across all choices of different nodes, we use a softmax function that maps these coefficients into the probability distribution:

$$a_{v,u}^{(l)} = \mathrm{softmax}\left(e_{v,u}^{(l)}\right) = \frac{\exp\left(e_{v,u}^{(l)}\right)}{\sum_{k \in \tilde{\mathcal{N}}(v)} \exp\left(e_{v,k}^{(l)}\right)}. \tag{3}$$

In each layer, the attention coefficients are calculated only in the first-order neighborhood of node v ($\forall v \in \mathcal{V}$), denoted as $\tilde{\mathcal{N}}(v)$.

2. Feature passing

Feature passing is the key step of feature aggregation over the graph. The attention-guided neighborhood aggregation can be summarized as a weighted linear combination across all choices of neighboring nodes and the node itself, as follows:

$$h_v^{(l+1)} = \sigma\left(\sum_{u \in \tilde{\mathcal{N}}(v)} a_{v,u}^{(l)} h_u^{(l)} \boldsymbol{W}^{(l)}\right). \tag{4}$$

More compactly, in matrix form, an attention layer with a propagation matrix $\boldsymbol{P} \in \mathbb{R}^{N \times N}$ is defined as

$$\boldsymbol{H}^{(l+1)} = \sigma\Big(\boldsymbol{P}^{(l)}\boldsymbol{H}^{(l)}\boldsymbol{W}^{(l)}\Big). \tag{5}$$

Similar to the adjacency matrix $\boldsymbol{A}$, the entry $P_{v,u}^{(l)}$ equals to the positive attention coefficient only when node u is directly connected to node v, i.e., $P_{v,u}^{(l)} = a_{v,u}^{(l)}$ if $(v,u) \in \mathcal{E}$; otherwise, $P_{v,u}^{(l)} = 0$. The feature passing is dynamic because the propagation matrix may change over the layers. Moreover, using softmax normalization, each row of $\boldsymbol{P}^{(l)}$ sums to one, which ensures that the hidden state of the node will not be scaled after feature passing in the attention layer.

4.2 Layer-wise attention

For node classification, many other GNNs use the node embedding $h_v^{(K)}$ of the last iteration to predict labels. Yet the features of beginning iterations may preserve locality (close to the root node), and they are better for prediction. Thus, after executing K iterations of layer transformations that include the first embedding layer and the subsequent node-wise attention layers, the input to the layer-wise attention layer is a finite set of all hidden embeddings $\{h_v^{(1)}, h_v^{(2)}, \ldots, h_v^{(K)}\}$ for node v. To consider an efficient depth exploration, we provide two attention mechanisms that are operated on all hidden embeddings: parametric and non-parametric attention mechanisms. This procedure is denoted by the placeholder L-ATTN in Algorithm 1.

1. Parametric attention mechanism

The parametric attention mechanism works in the self-attention setting, which allows the input features to be the criteria for the attention itself (Vaswani et al., 2017). As depicted in Fig. 3a, we use a single-layer perceptron to calculate the attention coefficients. It can be expressed as follows:

$$\boldsymbol{C}_v = \operatorname{softmax}\left(\sigma\left(\frac{\boldsymbol{H}_v\boldsymbol{\Theta}}{\sqrt{d_\mathrm{h}}}\right)\right). \tag{6}$$

Here, $\boldsymbol{H}_v \in \mathbb{R}^{K\times d_\mathrm{h}}$ is the matrix that consists of all hidden embeddings of K layers, $\boldsymbol{C}_v \in \mathbb{R}^K$ is the calculated attention coefficient vector (the subscript v indicates that these notations are related to node v, $\forall v \in \mathcal{V}$), $\boldsymbol{\Theta} \in \mathbb{R}^{d_\mathrm{h}}$ is the learnable attention parameter shared across all nodes, and $1/\sqrt{d_\mathrm{h}}$ is a scaled operation.

To obtain sufficient attention information, in practice, we use the multi-attention mechanism to make the model more powerful. M versions of the self-attention mechanism perform the transformation shown in Eq. (6) in parallel, and then yield a concatenated Md_h-dimensional output vector as the final representation:

$$h_v^{(K+1)} = \Big\Vert_{i=1}^{M} \left(\boldsymbol{C}_v^i\right)^\mathrm{T} \boldsymbol{H}_v. \tag{7}$$

Here, $\boldsymbol{C}_v^i$ is the i^th attention coefficient vector calculated by the corresponding attention head, parameter $\boldsymbol{\Theta}$ in each attention head is independent, and $\Vert$ indicates concatenation.

2. Non-parametric attention mechanism

We also use a simpler attention mechanism that proceeds without any parameters. Similar to the mechanism applied in the node-wise attention layer, cosine similarity is used here to calculate the attention coefficients. As illustrated in Fig. 3b, we measure the correlation between the K^th hidden embedding (final iteration) and the previous $K-1$ hidden embeddings. The cosine function is defined as $\cos(x,y) = xy/(\Vert x\Vert\Vert y\Vert)$. After computing $K-1$ normalized attention coefficients, a weighted summation is performed to obtain the output embedding $h_v^{(K+1)}$. The complete non-parametric attention mechanism is as follows:

$$\begin{aligned} c_v^{(l)} &= \operatorname{softmax}\left(\cos\left(h_v^{(l)}, h_v^{(K)}\right)\right),\\ h_v^{(K+1)} &= \sum_{l=1}^{K-1} c_v^{(l)} h_v^{(l)}. \end{aligned} \tag{8}$$

The K^th hidden embedding $h_v^{(K)}$ is not used in layer aggregation, and it is considered only as the attention criterion for coefficient calculation.

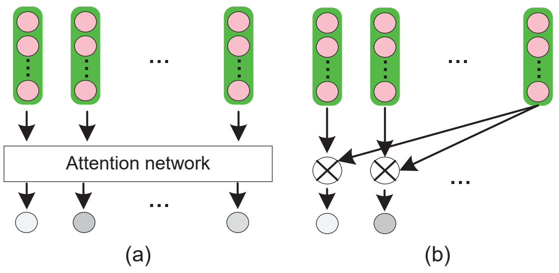

Fig. 3 Two versions of layer-wise attention: (a) parametric attention mechanism (self-attention); (b) non-parametric attention mechanism

4.3 Node embedding

Here, we summarize the node embedding procedure of NGAT. First, an embedding layer is used to transform the input features into higher-level features to obtain sufficient expressive power, and also to retain the inherent features of the nodes when feeding into the layer-wise attention layer. Next, we use $K-1$ node-wise attention layers followed by a layer-wise attention layer. Node-wise attention improves the neighborhood aggregation strategy by distinguishing the importance of each neighboring node, and layer-wise attention is to selectively leverage these hidden embeddings of all layers to generate an informative embedding Z_v for node v ($\forall v \in \mathcal{V}$).

4.4 Node classification

Similar to a standard MLP, we use linear regression with a softmax classifier for node classification:

$$\boldsymbol{S} = \operatorname{softmax}\left(\boldsymbol{Z}\boldsymbol{W}^{(K+1)}\right). \tag{9}$$

Here, $\boldsymbol{S} \in \mathbb{R}^{N\times d_\mathrm{c}}$ is the probability distribution of class prediction (d_c is the number of classes), $\boldsymbol{Z} \in \mathbb{R}^{N\times d_\mathrm{h}}$ is the final node embedding matrix, and $\boldsymbol{W}^{(K+1)} \in \mathbb{R}^{d_\mathrm{h}\times d_\mathrm{c}}$ is the trainable weight matrix.

The softmax function is applied in a row-wise manner. Note that if we apply the multi-head attention mechanism in layer-wise aggregation, the final node embedding will contain Md_h features. Accordingly, the weight matrix $\boldsymbol{W}^{(K+1)} \in \mathbb{R}^{Md_\mathrm{h}\times d_\mathrm{c}}$, where M is the number of attention heads. For semi-supervised node classification, we jointly train NGAT and the multi-class softmax classifier using the gradient descent method. The optimization goal is to minimize the cross-entropy loss over all labeled nodes:

$$\mathcal{L} = -\sum_{i\in\mathcal{Y}_L}\sum_{j=1}^{d_\mathrm{c}} Y_{ij}\ln S_{ij}, \tag{10}$$

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

The time complexity of a single node-wise attention layer computing d_h-dimensional embedding may be expressed as $O(Nd_\mathrm{h}^2 + |\mathcal{E}|d_\mathrm{h})$, where N and $|\mathcal{E}|$ are the numbers of nodes and edges in the graph, respectively. The time complexity of the layer-wise attention head aggregating K-layer hidden embeddings may be $O(NKd_\mathrm{h})$. The memory complexity is $O(N^2)$. If using the sparse representation for the adjacency matrix $\boldsymbol{A}$ and the node feature matrix $\boldsymbol{X}$, the memory requirement can be reduced to $O(|\mathcal{E}|)$.

5 Experiments

We evaluate NGAT on semi-supervised node classification datasets and compare its performance with those of other powerful GNNs. Our experimental results show that NGAT is highly competitive in learning representation of graph data.

5.1 Datasets

We use five benchmark datasets for our experiments. Three are well-known citation networks, i.e., PubMed (Namata et al., 2012), Cora, and Citeseer (Sen et al., 2008), where nodes and edges represent documents and citation links, respectively. The other two are new co-authorship networks: Microsoft Academic CS and Academic Physics from the KDD Cup 2016 Challenge (https://www.kdd.org/kdd-cup/view/kdd-cup-2016/Data). All datasets use the bag-of-words representations as node features, and the class labels indicate the research fields that each node (document/author) belongs to. The statistics of the datasets are reported in Table 1. We treat all graphs as undirected versions, and the adjacency matrix of each graph is binary.

5.2 Experimental setup and baselines

We construct an NGAT model with five node-wise attention layers ($K = 6$ with the first embedding layer). Each hidden layer has a fixed feature size of $d_\mathrm{h} = 32$. For the layer-wise attention mechanism, we evaluate both parametric and non-parametric versions, denoted by P-NGAT and NP-NGAT, respectively. P-NGAT applies the multi-head attention with four heads ($M = 4$). For optimization, we choose the Adam optimizer (Kingma and Ba, 2014) with learning rate of 0.005. We fix the dropout (Srivastava et al., 2014) rate to be $p = 0.5$ applied after all linear layers and node-wise attention layers, and add an L_2 regularization with $\lambda = 0.0005$ on the model parameters. In practice, we have tried other hyperparameter settings and evaluated them on the validation set but did not find much difference. We train for a maximum of 500 epochs. However, the actual training time is considerably shorter since we use an early stopping criterion. Specifically, training stops if the validation loss does not decrease for 20 epochs. We reset the parameters of NGAT to the state with the lowest validation loss. In each epoch, we feed all training nodes as a batch, and the model parameters are updated through back propagation to reduce the cross-entropy training loss. For baseline approaches, we test GCN (Kipf and Welling, 2017), AGNN (Thekumparampil et al., 2018), GAT (Veličković et al., 2018), and JK-Net with GCN

Table 1 Statistics of the node classification datasets used for experiments

Dataset	Number of nodes	Number of edges	Number of features	Number of classes	Train/Dev/Test size	Label rate
Cora	2708	5429	1433	7	140/500/1000	0.052
Citeseer	3327	4732	3703	6	120/500/1000	0.036
PubMed	19 717	44 338	500	3	60/500/1000	0.003
Academic CS	18 333	81 894	6805	15	300/500/1000	0.016
Academic Physics	34 493	247 962	8415	5	100/500/1000	0.003

feature concatenation (Xu et al., 2018) from the released implementations, and thereafter, cite the performances of other novel GNNs such as DGI (Veličković et al., 2019), SGC (Wu F et al., 2019), LanczosNet (Liao et al., 2019), and AdaLNet (Liao et al., 2019) from their original papers.

5.3 Results

1. Planetoid split

First, we evaluate all models on a common semi-supervised train/dev/test set according to Planetoid split (Yang et al., 2016), i.e., 20 nodes per class for training, 500 nodes for validation, and 1000 nodes for test. Except the training set, the remaining nodes are label-invisible during training. The quantitative details are presented in Table 1. Table 2 reports the experimental results on Planetoid split. For each dataset, we evaluate the performance of all models for 10 runs, and compute the average test accuracy with standard deviation. We observe that both NP-NGAT and P-NGAT achieve the best accuracy and significantly outperform all baseline models ($p < 0.05$ based on Student's t-test) on all five datasets, which shows that NGAT is highly competitive on the node classification task. Compared to JK-Net with six layers, which is similar to our model that uses a final aggregated layer for all hidden embeddings, the classification accuracy of using P-NGAT is improved by 2.5% on Cora, and 3.2% on PubMed. NP-NGAT also shows the same classification boost. This improvement shows that our attention-guided layer-wise aggregation is suitable for learning a structure-aware representation. Furthermore, NP-NGAT and P-NGAT both achieve significant performance in experiments, but there is a slight difference in the relatively large graphs (PubMed, Academic CS, and Academic Physics, where the number of nodes is 10 times that of Cora and Citeseer) such that P-NGAT outperforms NP-NGAT by 0.6% roughly.

2. Random split

In the Planetoid split experiment, we sample the same number of labeled nodes per class for training. In the second experiment, we use Cora, Citeseer, and PubMed citation datasets, and test all the models on random split. The experimental settings follow Buchnik and Cohen (2018) with some slight changes. We retain the train/dev/test set of the same size as in the Planetoid split, but randomly sample the labeled nodes for training. For example, there are still 140 labeled nodes for training in the Cora dataset, but different classes might have different numbers of labeled nodes in the training set. We evaluate the performance of all models in such a random split for 10 runs, and in each run, we carry out a new random sampling and report the experimental results in Table 3. P-NGAT and NP-NGAT still achieve the best classification accuracy on the three datasets. Compared to the results in Table 2, the test accuracy

Table 2 Node classification accuracy on Planetoid split from Yang et al. (2016)

	Model	Node classification accuracy (%)				
		Cora	Citeseer	PubMed	Academic CS	Academic Physics
From literature*	GCN	81.5	70.3	79.0	91.1±0.5**	92.8±1.0**
	GAT	83.0±0.7	72.5±0.7	79.0±0.3	90.5±0.6**	92.5±0.9**
	AGNN	83.1±0.1	71.7±0.1	79.9±0.1	–	–
	DGI	82.3±0.6	71.8±0.7	76.8±0.6	–	–
	SGC	81.0±0.0	71.9±0.1	78.9±0.0	–	–
	LanczosNet	79.5±1.8	66.2±1.9	78.3±0.3	–	–
	AdaLNet	80.4±1.1	68.7±1.0	78.1±0.4	–	–
Our experiments	GCN	81.6±0.6	70.6±0.8	78.5±0.8	89.8±0.4	91.1±1.3
	GAT	83.2±0.4	72.3±0.7	79.2±0.5	90.1±0.8	92.1±0.8
	AGNN	83.8±0.2	71.0±0.4	79.8±0.4	90.9±0.3	91.5±0.3
	JK-Net	82.6±0.8	72.4±1.0	77.9±0.5	91.4±0.4	92.4±0.7
Our models	NP-NGAT	84.9±0.4	**72.7**±0.6	80.8±0.8	91.0±0.5	92.6±0.4
	P-NGAT	**85.1**±0.3	72.6±0.9	**81.1**±0.6	**91.7**±1.1	**93.2**±0.8

The best result in the corresponding dataset is in bold. * means that the results are cited from their original papers; ** means that the reported numbers are taken from Shchur et al. (2018); – denotes no result for the corresponding dataset. The reported numbers in "Our experiments" are the test accuracy values averaged over 10 runs. NP-NGAT and P-NGAT both outperform the compared models on all datasets based on Student's t-test ($p < 0.05$)

of all models shows a decrease by about 3%–4% in the random split experiment. It is worth noting that JK-Net and NGAT have a relatively small drop compared to GCN, GAT, and AGNN. We think that it is due to the layer aggregation mechanism. As the node classes in the training set are unfairly distributed, a model is prone to be optimized toward the classes involving more labeled nodes. For those classes with fewer labeled nodes, the classification variance will accumulate as the number of layers increases. Hence, the features of earlier iterations are more helpful. JK-Net and NGAT both use node embeddings from all iterations while other models use only the node embedding of the final iteration, so the classification accuracy of models using layer aggregation is better than those of others.

3. Different aggregators

Table 4 shows the effect of using self-attention for layer-wise feature aggregation. We report the performance of self-attention (Self-Attn) and compare it to those of other simple aggregators that fuse the output of all hidden layers. The baseline "None" means not to use layer-wise aggregation. We explore three simple aggregators: Concatenation (Concat) directly combines the features of all layers; MaxPooling (MaxPool) is the most straightforward way to select the informative layer for each feature coordinate; AveragePooling (AvgPool) is applied in a point-wise manner to fuse all features equally. We construct a model using five node-wise attention layers; in each experiment, we combine this base model with different aggregators, and then report the test accuracy averaged over 10 runs. The train/dev/test set is split according to the Planetoid split experiment. It can be easily seen that compared to the "None" aggregator, using layer-wise feature aggregation can greatly improve the accuracy of node classification. Moreover, the result of using self-attention is better than those of the other three simple aggregators, which shows that our self-attention aggregator can learn node embeddings with abundant structural information very well.

4. Size of training set

Since labeled data are very important for semi-supervised learning, in the next experiment, we look into the influence of different numbers of labeled training nodes on classification performance. We retain the validation and test set of the same size as in the previous experiments, and change the size of the training set within $\{5, 10, 20, 30, 40, 50, 60\}$ labeled nodes per class. Here, we choose AGNN rather than GAT as the baseline of attention-based models, because our breadth exploration uses cosine similarity to calculate attention coefficients, which is similar to the setting of AGNN. Therefore, the comparison with AGNN will show the improvement of NGAT using layer aggregation. We run GCN, AGNN, JK-Net, and P-NGAT on the Cora dataset for 10 random seeds and report the average test accuracy in Fig. 4. We can find that with a smaller number of labeled nodes, P-NGAT is more superior to other models in terms of classification accuracy. We attribute this improvement in accuracy to the fact that NGAT stacks more layers than other models, so the label information can be propagated over a larger neighborhood. Therefore, this experiment shows that deep GNNs perform better in scenarios with sparse labels. When the number of labeled nodes increases, all test models reach a similar performance level.

Table 3 Node classification accuracy on random split for citation network datasets

Model	Node classification accuracy (%)		
	Cora	Citeseer	PubMed
GCN	79.0±1.3	67.5±1.9	76.2±2.3
GAT	80.1±1.0	68.4±1.9	77.0±2.9
AGNN	80.4±0.8	68.2±0.7	76.6±2.1
JK-Net	80.9±1.5	69.7±1.6	77.2±1.6
NP-NGAT	82.4±1.8	69.7±1.2	78.8±1.8
P-NGAT	**82.8**±1.4	**70.2**±1.5	**78.9**±1.6

The best results are in bold. Reported numbers are in the form of mean accuracy with standard deviation of 10 runs

Table 4 Test accuracy of different layer-wise aggregators on citation network datasets

Aggregator	Test accuracy (%)		
	Cora	Citeseer	PubMed
None	80.4±0.2	69.4±0.4	78.2±0.1
MaxPool	83.6±0.5	72.1±0.5	80.8±0.2
AvgPool	84.2±0.2	71.7±0.1	80.4±0.1
Concat	84.4±0.8	72.2±1.0	79.9±0.5
Self-Attn	**85.1**±0.3	**72.6**±0.7	**81.1**±0.6

The best results are in bold. Reported numbers are averaged over 10 runs

5. Model depth

In this experiment, we explore how the accuracy of node classification depends on the depth for different models. We add an ablation study to compare the performance when using only node-wise attention or layer-wise attention, and then empirically demonstrate NGAT's capacity to alleviate the problem of oversmoothing. The baseline model is GCN. Node-wise GCN (NGCN) denotes an NGAT with only node-wise attention, while layer-wise GCN (LGCN) is a model with only layer-wise attention. We report the test accuracy averaged over 10 runs on five datasets: Cora, Citeseer, PubMed, Academic CS, and Acadamic Physics using Planetoid split (Yang et al., 2016). The results are summarized in Fig. 5. For the datasets considered here, the

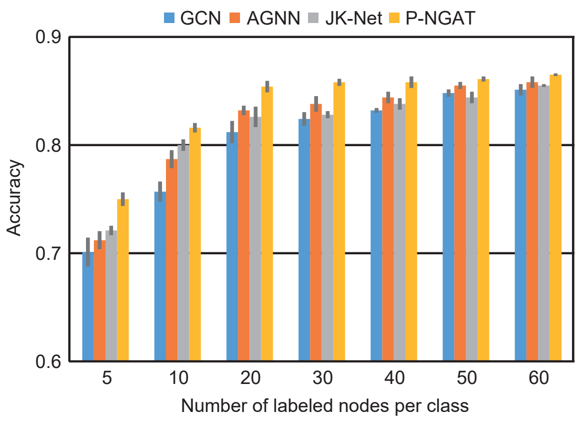

Fig. 4 Test accuracy averaged over 10 runs for different sizes of the training set (number of labeled nodes per class) on the Cora dataset. For smaller training set sizes, the performance of P-NGAT increases further

best results of GCN and NGCN are both obtained with less than five layers. If the GCN has more than five layers, the classification accuracy drops rapidly, while the NGCN with more than 10 layers encounters accuracy decrease. It shows that node-wise attention can improve the accuracy, but cannot alleviate oversmoothing clearly. In addition, we can observe that compared to the results on the Cora and Citeseer datasets, the accuracy decrease on the PubMed, Academic CS, and Academic Physics datasets (whose number of nodes is 10 times that of Cora and Citeseer) is relatively small. For example, on the Cora dataset, the accuracy of the 25-layer GCN decreases by 29% compared to the accuracy of the 2-layer GCN, while the decrease is 9% on the Academic Physics dataset. If we use only layer-wise attention, depicted by LGCN, we find that it can maintain high classification accuracy even with deep layers, which shows the ability of overcoming the oversmoothing issue. It is worth noting that with both node-wise attention and layer-wise attention, P-NGAT can alleviate the problem of oversmoothing, and further improve the classification accuracy. Therefore, we can empirically conclude that even with deep layers, the node embeddings learned by P-NGAT are informative and distinguishable.

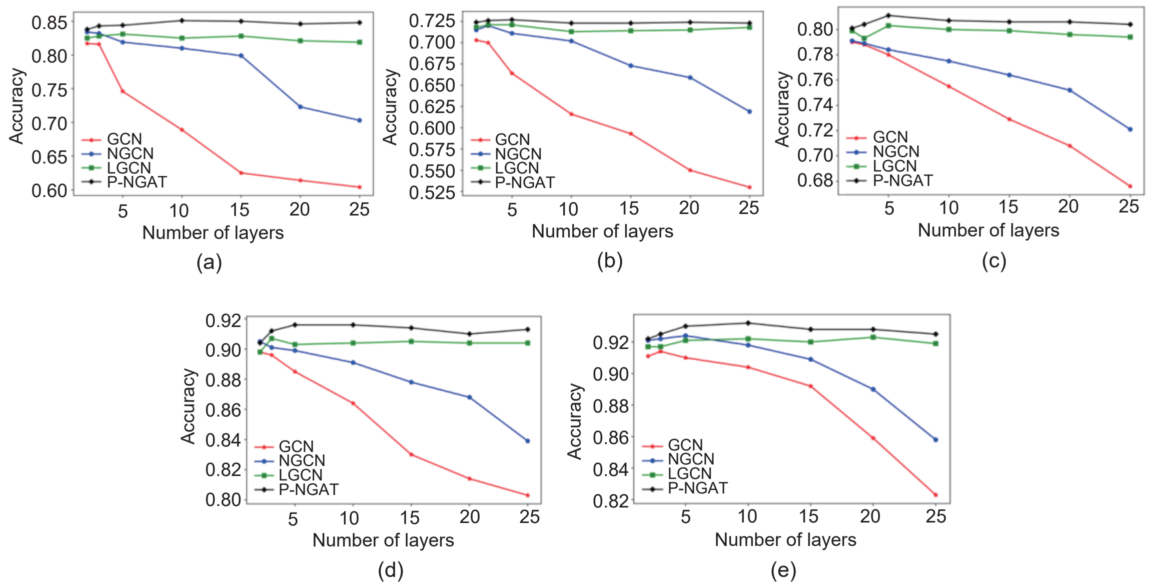

Fig. 5 Influence of model depth (number of layers) on node classification performance. The experimental results of P-NGAT (using self-attention layer-wise aggregation) are compared to those of a standard GCN model, an NGAT with only node-wise attention (NGCN), and an NGAT with only layer-wise attention (LGCN) on datasets Cora (a), Citeseer (b), PubMed (c), Academic CS (d), and Academic Physics (e). Markers denote test accuracy averaged over 10 runs

5.4 Analysis

Based on the above experimental results, we conclude that NGAT can learn high-quality node embeddings in a semi-supervised manner. To obtain a deep understanding of the layer-wise attention mechanism, we train the 10-layer and 20-layer P-NGAT on the Cora dataset for comparison. The train/dev/test set is split according to the Planetoid split.

First, we present the classification results of 1000 test nodes using the t-SNE algorithm (van der Maaten and Hinton, 2008), as shown in Fig. 6. Numerically speaking, the 10-layer P-NGAT can achieve an 84.6% average accuracy, while the 20-layer counterpart shows 84.4% accuracy. It can be seen that there is no significant decrease in performance as the model goes deeper. The same result can be seen between the panels in Figs. 6a and 6b, and the class cluster visualization of the 20-layer P-NGAT still has a clear distinction, which is close to the 10-layer version.

Second, we look for the preference of NGAT in different neighborhood depths. We define the concept of layer importance to characterize the contribution of the layers to the final node embeddings.

Definition 1 (Layer importance) In multi-head self-attention settings, $\boldsymbol{C}_v^i \in \mathbb{R}^K$ is the i^{th} attention vector computed by the corresponding head for node v ($\forall v \in \mathcal{V}$). The layer importance LI(l) is defined as

$$\text{LI}(l) = \text{softmax}\left(\sum_{v \in \mathcal{V}} \sum_{i=1}^{M} C_{v,l}^i\right),\ l \in \{1, 2, \ldots, K\}, \tag{11}$$

where M is the number of attention heads and $C_{v,l}^i$ is the l^{th} attention coefficient.

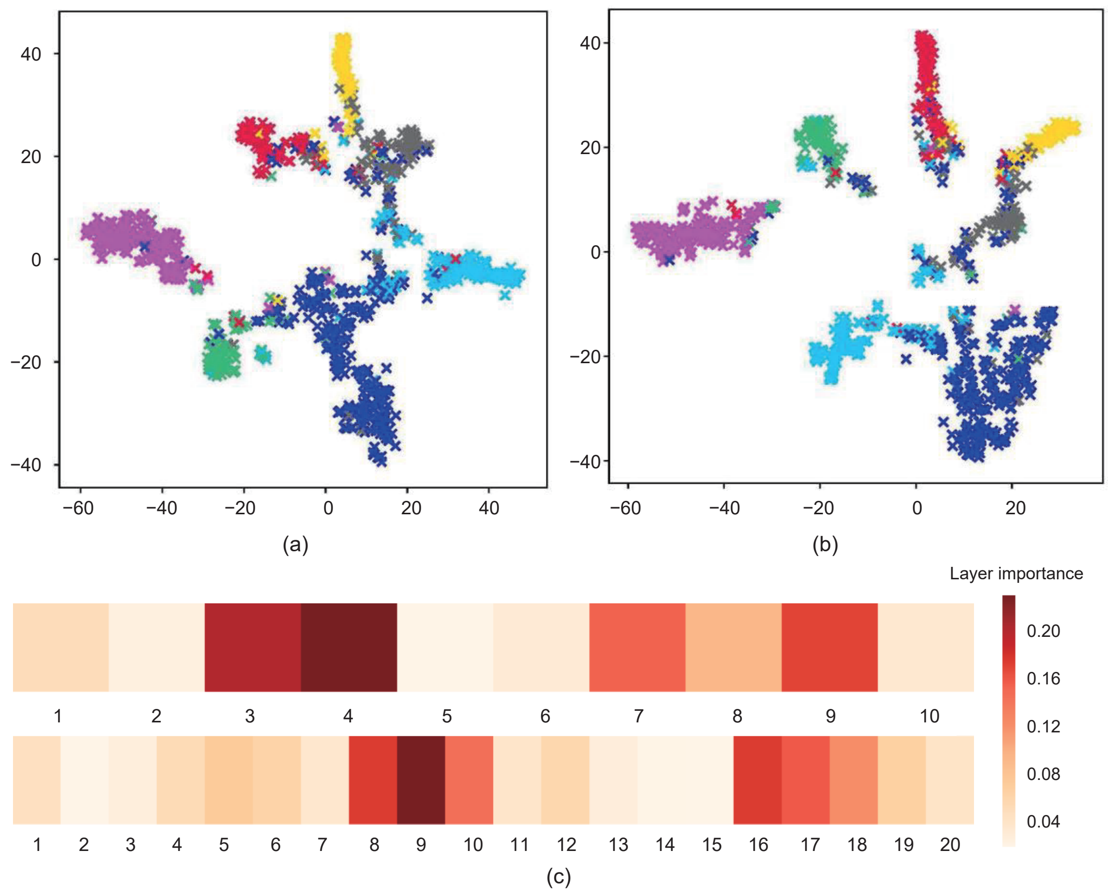

Fig. 6 Insight into NGAT performance on the Cora dataset. (a) and (b) are the t-SNE visualization of the classification of 1000 test nodes. Different colors mean different classes. Panel (a) is a 10-layer P-NGAT and panel (b) shows a 20-layer model. (c) depicts the layer importance of each hidden layer for representation. The numbers below the panel represent the index of the layer, and darker color means greater importance score

Fig. 6c visualizes the layer importance of each hidden layer in heatmap form. We find that the 10-layer P-NGAT emphasizes the node embeddings of the 3rd and 4th layers, while the 20-layer P-NGAT emphasizes the layers from the 8th to 10th and from the 16th to 18th. Therefore, despite being a 10-layer or deeper architecture, NGAT pays sufficient attention to those shallow layers, and selectively fuses these features from all layers to generate a depth-adaptive node embedding.

Lastly, we are interested in the nodes that are misclassified. We construct a six-layer P-NGAT, and present the quantitative statistics of truly and falsely classified nodes in Table 5. Table 5 shows that every truly classified node v_t has an average of 3.91 neighboring nodes. Among these neighbors, 89.4% nodes have the same label as v_t, while the falsely classified nodes have smaller values in both the fields. We can conclude that the local structure and label sharing are beneficial to node embeddings. For each node in the graphs, more neighbors provide more mutual information, and more neighbors with the same label are helpful to generate a label-aware node embedding.

Table 5 Quantitative statistics of the test prediction results on the Cora dataset

Case	Number of nodes	Avg.N	Avg.C (%)
True	854	3.91	89.4
False	146	3.24	46.4

Avg.N: average number of directly connected nodes; Avg.C: average proportion of neighboring nodes that have the same label as the classified node

6 Conclusions

In this paper, we have proposed a novel nested graph attention network (NGAT), which can learn high-quality node embeddings on graphs with very few labeled nodes. The intuitive motivation is to reduce the harm of oversmoothing, which renders node embeddings indistinguishably and hurts the classification accuracy as the number of network layers increases. NGAT improves both breadth and depth exploration, and allows the stacking of more layers within a feasible scope. Our work can be standardized as an extension of JK-Net (Xu et al., 2018). We have adopted a simple but powerful architecture for both neighborhood and layer aggregations. On many node classification benchmarks, NGAT achieves state-of-the-art performance compared to some novel GNNs. For future work, we aim to first explore ways such as sampling methods (Chen et al., 2018; Zou et al., 2019) to reduce the computational complexity of node embedding. Second, we try to apply NGAT to graph classification tasks.

Contributors

Jianke HU and Yin ZHANG designed the research. Jianke HU processed the data and drafted the paper. Yin ZHANG helped organize the paper. Jianke HU and Yin ZHANG revised and finalized the paper.

Compliance with ethics guidelines

Jianke HU and Yin ZHANG declare that they have no conflict of interest.

References

Atwood J, Towsley D, 2016. Diffusion-convolutional neural networks. Proc 30th Int Conf on Neural Information Processing Systems, p.2001-2009.

Belkin M, Niyogi P, Sindhwani V, 2006. Manifold regularization: a geometric framework for learning from labeled and unlabeled examples. *J Mach Learn Res*, 7:2399-2434.

Bruna J, Zaremba W, Szlam A, et al., 2014. Spectral networks and locally connected networks on graphs. https://arxiv.org/abs/1312.6203

Buchnik E, Cohen E, 2018. Bootstrapped graph diffusions: exposing the power of nonlinearity. Proc ACM Int Conf on Measurement and Modeling of Computer Systems, p.8-10. https://doi.org/10.1145/3219617.3219621

Chapelle O, Scholkopf B, Zien A, 2009. Semi-supervised learning (Chapelle, O. et al., Eds.; 2006) [book reviews]. *IEEE Trans Neur Netw*, 20(3):542. https://doi.org/10.1109/TNN.2009.2015974

Chen J, Ma TF, Xiao C, 2018. FastGCN: fast learning with graph convolutional networks via importance sampling. https://arxiv.org/abs/1801.10247

Defferrard M, Bresson X, Vandergheynst P, 2016. Convolutional neural networks on graphs with fast localized spectral filtering. Proc 30th Int Conf on Neural Information Processing Systems, p.3844-3852.

Grover A, Leskovec J, 2016. node2vec: scalable feature learning for networks. Proc 22nd ACM SIGKDD Int Conf on Knowledge Discovery and Data Mining, p.855-864. https://doi.org/10.1145/2939672.2939754

Hamilton WL, Ying R, Leskovec J, 2017. Inductive representation learning on large graphs. Proc 31st Int Conf on Neural Information Processing Systems, p.1025-1035.

He KM, Zhang XY, Ren SQ, et al., 2016. Deep residual learning for image recognition. Proc IEEE Conf on Computer Vision and Pattern Recognition, p.770-778. https://doi.org/10.1109/CVPR.2016.90

Kingma DP, Ba J, 2014. Adam: a method for stochastic optimization. https://arxiv.org/abs/1412.6980

Kipf TN, Welling M, 2017. Semi-supervised classification with graph convolutional networks. https://arxiv.org/abs/1609.02907

Klicpera J, Bojchevski A, Günnemann S, 2019. Predict then propagate: graph neural networks meet personalized pagerank. https://arxiv.org/abs/1810.05997v4

Knyazev B, Taylor GW, Amer MR, 2019. Understanding attention and generalization in graph neural networks. Proc 33rd Conf on Neural Information Processing Systems, p.4204-4214.

Krizhevsky A, Sutskever I, Hinton GE, 2012. ImageNet classification with deep convolutional neural networks. Proc 25th Int Conf on Neural Information Processing Systems, p.1097-1105.

Lee J, Lee I, Kang J, 2019. Self-attention graph pooling. https://arxiv.org/abs/1904.08082

Li QM, Han ZC, Wu XM, 2018. Deeper insights into graph convolutional networks for semi-supervised learning. Proc 32nd AAAI Conf on Artificial Intelligence, p.3538-3545.

Liao RJ, Zhao ZZ, Urtasun R, et al., 2019. LanczosNet: multi-scale deep graph convolutional networks. https://arxiv.org/abs/1901.01484v1

Namata G, London B, Getoor L, et al., 2012. Query-driven active surveying for collective classification. Proc 10th Int Workshop on Mining and Learning with Graphs, Article 8.

Niepert M, Ahmed M, Kutzkov K, 2016. Learning convolutional neural networks for graphs. Proc 33rd Int Conf on Machine Learning, p.2014-2023.

Perozzi B, Al-Rfou R, Skiena S, 2014. DeepWalk: online learning of social representations. Proc 20th ACM SIGKDD Int Conf on Knowledge Discovery and Data Mining, p.701-710. https://doi.org/10.1145/2623330.2623732

Ribeiro LFR, Saverese PHP, Figueiredo DR, 2017. struc2vec: learning node representations from structural identity. Proc 23rd ACM SIGKDD Int Conf on Knowledge Discovery and Data Mining, p.385-394. https://doi.org/10.1145/3097983.3098061

Sen P, Namata G, Bilgic M, et al., 2008. Collective classification in network data. *AI Mag*, 29(3):93. https://doi.org/10.1609/aimag.v29i3.2157

Shchur O, Mumme M, Bojchevski A, et al., 2018. Pitfalls of graph neural network evaluation. https://arxiv.org/abs/1811.05868

Simonyan K, Zisserman A, 2014. Very deep convolutional networks for large-scale image recognition. https://arxiv.org/abs/1409.1556

Srivastava N, Hinton G, Krizhevsky A, et al., 2014. Dropout: a simple way to prevent neural networks from overfitting. *J Mach Learn Res*, 15(1):1929-1958.

Thekumparampil KK, Wang C, Oh S, et al., 2018. Attention-based graph neural network for semi-supervised learning. https://arxiv.org/abs/1803.03735

van der Maaten L, Hinton G, 2008. Visualizing data using t-SNE. *J Mach Learn Res*, 9:2579-2605.

Vaswani A, Shazeer N, Parmar N, et al., 2017. Attention is all you need. Proc 31st Int Conf on Neural Information Processing Systems, p.6000-6010.

Veličković P, Cucurull G, Casanova A, et al., 2018. Graph attention networks. https://arxiv.org/abs/1710.10903v1

Veličković P, Fedus W, Hamilton WL, et al., 2019. Deep graph infomax. https://arxiv.org/abs/1809.10341

Wu F, Zhang TY, de Souza AHJr, et al., 2019. Simplifying graph convolutional networks. https://arxiv.org/abs/1902.07153

Wu ZH, Pan SR, Chen FW, et al., 2019. A comprehensive survey on graph neural networks. https://arxiv.org/abs/1901.00596

Xu K, Li CT, Tian YL, et al., 2018. Representation learning on graphs with jumping knowledge networks. https://arxiv.org/abs/1806.03536

Xu K, Hu WH, Leskovec J, et al., 2019. How powerful are graph neural networks? https://arxiv.org/abs/1810.00826

Yang ZL, Cohen W, Salakhudinov R, 2016. Revisiting semi-supervised learning with graph embeddings. Proc 33rd Int Conf on Machine Learning, p.40-48.

Zhou J, Cui GQ, Zhang ZY, et al., 2018. Graph neural networks: a review of methods and applications. https://arxiv.org/abs/1812.08434

Zhu XJ, Ghahramani Z, Lafferty J, 2003. Semi-supervised learning using Gaussian fields and harmonic functions. Proc 20th Int Conf on Machine Learning, p.912-919.

Zou DF, Hu ZN, Wang YW, et al., 2019. Layer-dependent importance sampling for training deep and large graph convolutional networks. Proc 33rd Int Conf on Neural Information Processing Systems, p.11247-11256.