GraphAIR: Graph Representation Learning with Neighborhood Aggregation and Interaction

Fenyu Hu, Yanqiao Zhu, Shu Wu, Weiran Huang, Liang Wang, *Fellow, IEEE,* and Tieniu Tan, *Fellow, IEEE*

Graph representation learning is of paramount importance for a variety of graph analytical tasks, ranging from node classification to community detection. Recently, graph convolutional networks (GCNs) have been successfully applied for graph representation learning. These GCNs generate node representation by aggregating features from the neighborhoods, which follows the "neighborhood aggregation" scheme. In spite of having achieved promising performance on various tasks, existing GCN-based models have difficulty in well capturing complicated non-linearity of graph data. In this paper, we first theoretically prove that coefficients of the neighborhood interacting terms are relatively small in current models, which explains why GCNs barely outperforms linear models. Then, in order to better capture the complicated non-linearity of graph data, we present a novel GraphAIR framework which models the neighborhood interaction in addition to neighborhood aggregation. Comprehensive experiments conducted on benchmark tasks including node classification and link prediction using public datasets demonstrate the effectiveness of the proposed method over the state-of-the-art methods.

Index Terms—Graph representation learning, neighborhood aggregation, neighborhood interaction, node classification, link prediction

✦

1 INTRODUCTION

Graph representation learning aims to transform nodes on the graph into low-dimensional dense vectors whilst still preserving the attribute features of nodes and structure features of graphs. These node embeddings can then be fed into downstream machine learning algorithms to facilitate graph analytical tasks, such as node classification [1, 2], link prediction [3], and community detection [4].

In recent years, there has been a surge of research interest in utilizing neural networks to handle graph-structured data. Among them, graph convolutional networks (GCNs) have been shown effective in graph representation learning. They can model complex attribute features and structure features of graphs and achieve the state-of-the-art performance on various tasks. The core of graph convolution is that nodes learn their representations by aggregating features from their neighbors, i.e. the "neighborhood aggregation" scheme. Recently, some graph convolutional models, which primarily differ in the neighborhood aggregation strategies, have been proposed [1, 2, 5, 6]. For example, GCN [1] can be seen as the approximation of aggregation on the first-order neighbors;

- F. Hu, Y. Zhu, S. Wu, L. Wang, and T. Tan are with the Center for Research on Intelligent Perception and Computing (CRIPAC), National Laboratory of Pattern Recognition (NLPR), Institute of Automation, Chinese Academy of Sciences (CASIA). Mail: {`fenyu.hu, yanqiao.zhu`}`@cripac.ia.ac.cn`, {`shu.wu, wangliang, tnt`}`@nlpr.ia.ac.cn`
- W. Huang is with School of Computer Science, Beijing University of Posts and Telecommunications. Work is done during his internship at CRIPAC.
- Fenyu Hu and Yanqiao Zhu are co-first authors. Corresponding author: Shu Wu.

GraphSAGE [7] designs several aggregators for inductive learning, where unlabeled data does not appear in the training process; GAT [2] introduces the attention mechanism to model influence of neighbors with learnable parameters.

From a historical perspective, machine learning research has gone through a long process of development, with one clear trend from simple and linear models to complex and non-linear models. For example, limitations of the linear support vector machine (SVM) motivated the development of non-linear and more expressive kernel-based SVM classifiers [8]. Besides, similar trends can be observed in the realm of image processing as real-world data distribution is usually rather complex. For example, simple and linear image filters [9] are gradually superseded by non-linear convolutional neural networks (CNNs) [10]. Driven by the significance of modeling complex and non-linear distributions of data, a question arises: *are existing GCNs capable enough to model the complex and non-linear distributions of graphs?* We find that most previous graph convolutional models (e.g., GCN and GAT) are usually shallow with only one or two non-linear activation function layers, which may restrict the model from well capturing the complicated non-linearity of graph data.

In this paper, we first theoretically prove that the effect of non-linear activation functions in GCNs is to introduce the interaction terms of neighborhood features. We then show that coefficients of the neighborhood interacting terms are relatively small in current GCN-based models. To this end, we present a general framework named GraphAIR (Aggregation and InteRaction). The key idea behind our approach is to *explicitly model the neighborhood interaction in addition to neighborhood aggregation, which can better capture the complex and non-linear node features.* As illustrated in Figure 1, GraphAIR consists of two parts, i.e. *aggre-*
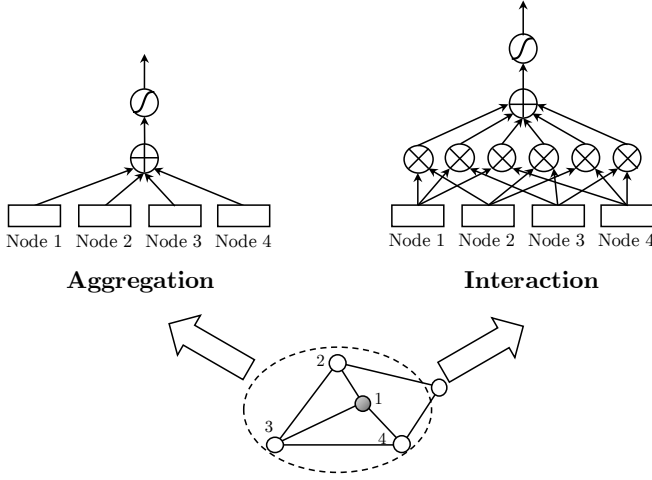
Fig. 1: A graphical illustration of the proposed GraphAIR model. The aggregation module sums up the neighborhood features; the interaction module models the pair-wise feature interaction among the neighborhoods.

gation and *interaction*. The aggregation module constructs node representations by combining features from neighborhoods; the interaction module explicitly models neighborhood interactions through multiplication.

Nevertheless, several challenges exist in modeling the neighborhood interaction. Firstly, different nodes may have various numbers of adjacent neighbors, leading to different numbers of interaction pairs among neighbors. Thereby, defining a universal neighborhood interaction operator which is able to handle arbitrary numbers of interaction pairs is challenging. Secondly, it is preferable to propose a general plug-and-play interaction module instead of designing model-specific neighborhood interaction strategies for different GCN-based models.

To tackle the aforementioned challenges, we derive that the neighborhood interaction can be easily obtained through the multiplication of node embeddings. As a result, both of the neighborhood aggregation module and the neighborhood interaction module can be implemented by most existing graph convolutional layers.

In a nutshell, the main contributions of this paper are threefold. Firstly, to best of our knowledge, it is the first work to explicitly model neighborhood interaction for capturing non-linearity of graph-structured data. Secondly, the proposed GraphAIR can easily integrate off-the-shelf graph convolutional models, which shows favorable generality. Thirdly, extensive experiments conducted on benchmark tasks of node classification and link prediction show that GraphAIR achieves the state-of-the-art performance.

2   BACKGROUND AND PRELIMINARIES

In this section, we firstly introduce the notations used throughout the paper and then summarize some of the most common GCN models. Last, we briefly introduce residual learning which we employ in our model.

2.1   Notations

Let $\mathcal{G} = (\boldsymbol{A}, \boldsymbol{X})$ be an undirected graph with n nodes, where $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ is the adjacency matrix, $\boldsymbol{X} \in \mathbb{R}^{n \times m}$ is the feature attribute

matrix, and $\boldsymbol{x}_i \in \mathbb{R}^{1 \times m}$ denotes the attribute of node i. Please kindly note that in this paper we primarily focus on undirected graphs, but our proposed method can be easily generalized to work with weighted or directed graphs.

2.2   Aggregators in Graph Convolutional Models

As mentioned above, existing GCNs mainly differ in the neighborhood aggregation functions. The representative graph convolutional model such as GCN [1] and GAT [2] can be formulated as:

$$\boldsymbol{n}_i^{(k)} = \boldsymbol{h}_i^{(k)} \boldsymbol{W}^{(k)}, \tag{1}$$

$$\boldsymbol{h}_i^{(k+1)} = \sigma \left(\sum_{j \in \mathcal{N}_i} e_{ij} \boldsymbol{n}_j^{(k)} \right), \tag{2}$$

where $\boldsymbol{h}_i^{(k)} \in \mathbb{R}^{d_k}$ is the embedding of the ith node resulting from the kth graph convolutional layer, $\boldsymbol{W}^{(k)} \in \mathbb{R}^{d_k \times d_{k+1}}$ is a learnable weight matrix, e_{ij} is a scalar which indicates the importance of node j's features to node i, and $\sum_j e_{ij} = 1$. $\sigma(\cdot)$ is the activation function, e.g., $\mathrm{ReLU}(\cdot) = \max(0, \cdot)$ and $\mathcal{N}_i$ is the set containing the first-order neighbors of node i as well as node i itself. To obtain the node embedding, a linear transformation is first conducted to project features to a new feature subspace. Then, the node embedding can be updated by weighted summation over the projected features of its neighbors, followed by a non-linear activation function.

Different models adopt different strategies to design the aggregators. For GCN, it uses a predefined weight matrix $\hat{\boldsymbol{A}} = \tilde{\boldsymbol{D}}^{-\frac{1}{2}} \tilde{\boldsymbol{A}} \tilde{\boldsymbol{D}}^{-\frac{1}{2}}$ for summarization, where $\tilde{\boldsymbol{A}} = \boldsymbol{A} + \boldsymbol{I}$ is the adjacency matrix with self-loops and $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$. Here, entry a_{ij} of $\hat{\boldsymbol{A}}$ is a predefined weight factor for weighted summarization over neighborhoods, i.e. e_{ij} in Eq.(2). Unlike GCN, GAT makes use of the attention mechanism to explicitly learn e_{ij} as follows:

$$\begin{aligned} \alpha_{ij} &= g(\boldsymbol{n}_i, \boldsymbol{n}_j), \\ e_{ij} &= \mathrm{softmax}(\alpha_{ij}) = \frac{\exp(\alpha_{ij})}{\sum_{k \in \mathcal{N}_i} \exp(\alpha_{ik})}, \end{aligned} \tag{3}$$

where $g : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is a self-attention function, which can be simply implemented as a feed-forward neural network.

The implicit and insufficient neighborhood interaction involved in existing GCNs. It is seen from Eq. (2) that without the activation function, the node representation would depend linearly on the neighborhood features. Then, although mainstream models adopt non-linear activation functions, which is able to introduce the neighborhood interaction implicitly as a side effect, they still face challenges in learning the neighborhood interaction sufficiently. We take the sigmoid function $s(t) = \frac{1}{1+e^{-t}}$ as an example and approximate it with Taylor polynomials. Note that mainstream GCN-based models use piecewise non-saturating activation functions, such as ReLU and $\mathrm{LeakyReLU}(x) = \max(0.01x, x)$. These functions suppress negative values yet are still linear for positive values. Here we analyze the sigmoid function as it brings more non-linearity. Since the elements in the node embeddings are small[1], the high-order interacting terms among the neighborhoods are small as well. Then, we just analyze the coefficients of

---

1. Most existing graph convolutional models, including GCN, GraphSAGE, and GAT normalize the input and initialize the weights using Glorot initialization [11].

high-order interacting terms, which is claimed in the following proposition.

Proposition 1. *When applying the sigmoid function $s(t)$ on the result of the linear combination as formulated in Eq. (2), the equivalent coefficient of high-order interacting terms of the neighborhood embeddings is at most $\frac{1}{48}$.*

Proof. The sigmoid function $s(t)$ can be approximated as Taylor polynomials at $t_0 = 0$:

$$s(t) \approx \sum_{p=0}^{P} \frac{s^{(p)}(0)}{p!} t^p = \frac{1}{2} + \frac{1}{4} t - \frac{1}{48} t^3 + \cdots + \frac{s^{(P)}(0)}{P!} t^P, \quad (4)$$

where P is the degree of the polynomial. The approximation error can be bounded using the Lagrange form of the remainder:

$$|R_p(t)| \leq \frac{|t|^{P+1}}{(P+1)!} M_p, \\ \text{where } |s^{P+1}(\theta)| \leq M_p, \quad \theta \in (-t, t). \quad (5)$$

Since the coefficient of the quadratic term is zero, we set $P = 2$ and analyze the contribution of high-order interacting terms. Then, replacing t with $\sum_{j \in \mathcal{N}_i} e_{ij} \boldsymbol{n}_j^{(k)}$, Eq. (2) can be written as follows:

$$\boldsymbol{h}_i^{(k+1)} = \frac{1}{2} + \frac{1}{4} \left(\sum_{j \in \mathcal{N}_i} e_{ij} \boldsymbol{n}_i^{(k)} \right) \\ + M \left(\sum_{j \in \mathcal{N}_i} \sum_{k \in \mathcal{N}_i} \sum_{l \in \mathcal{N}_i} e_{ij} e_{ik} e_{il} \boldsymbol{n}_j^{(k)} \cdot \boldsymbol{n}_k^{(k)} \cdot \boldsymbol{n}_l^{(k)} \right), \quad (6)$$

where M is the bound of the remainder whose absolute value is at most $\frac{1}{48}$, which concludes the proof. Detailed proof is given in Appendix C in the supplementary material. □

Remark. Proposition 1 states that the effect of non-linear activation functions in GCNs is to introduce the interaction terms of neighborhood features. Importantly, the coefficients of the neighborhood interacting terms in current GCN-based models are relatively small, leading to a negligible contribution to node representations. As existing GCNs are usually shallow with only one or two non-linear layers to avoid oversmoothing and overfitting [12], non-linearity of graph data cannot be learned sufficiently.

2.3 Residual Learning

In this paper, we employ residual learning to combine the neighborhood aggregation and interaction. Residual learning [13] is a widely-used building block for deep learning. Suppose $h(\boldsymbol{x})$ is the true and desired mapping and $\boldsymbol{x}$ is the suboptimal representation which serves as the input feature to the residual module. Residual learning can be formulated as:

$$h(\boldsymbol{x}) = f(\boldsymbol{x}) + \boldsymbol{x}, \quad (7)$$

where $f(\cdot)$ is a residual function. Practically, we can apply a few non-linear layers to obtain the suboptimal representation $\boldsymbol{x}$ and some other non-linear layers to implement the residual function f. The essence of residual learning lies in the skip connection, through which the earlier representations are able to flow to later layers. The skip connection enables more direct reuse of the suboptimal representation and improves the information flow during forward and backward propagation [13], which makes the network easier to be optimized. Many approaches [13, 14] have

shown that residual learning helps break away from the local optimum and improving the performance.

3 THE PROPOSED METHOD: GRAPHAIR

In this section, we firstly formulate the model of neighborhood interaction and then describe how the parameters of GraphAIR model can be learned. Finally, we summarize the overall model architecture and analyze the computational complexity.

3.1 Modeling the Neighborhood Interaction with Residual Functions

As discussed in Section 2.2, the node representation resulting from the neighborhood aggregation scheme is less likely to well capture complicated non-linearity of graphs because they learn the neighborhood interaction implicitly and inefficiently. In this section, we describe the embedding generation algorithm of GraphAIR, which aims to incorporate the neighborhood interaction into node representations. To begin with, a natural idea to model the quadratic terms of neighborhood interaction is formulated as:

$$\boldsymbol{h}_i^{\text{ir}} = \sum_{j \in \mathcal{N}_i} \sum_{k \in \mathcal{N}_i} \beta_{jk} \boldsymbol{n}_j \odot \boldsymbol{n}_k, \quad (8)$$

where $\boldsymbol{h}_i^{\text{ir}}$ is the neighborhood interaction representation of node i, β_{jk} denotes the coefficient of the quadratic term, and $\odot$ is the element-wise multiplication operator. However, it is infeasible to learn β_{jk} in our case. For each node i, there are $O(|\mathcal{N}_i|^2)$ coefficients to estimate, which exposes the risk of overfitting. To alleviate this problem, we simply assign β_{jk} as the product of importance weights e_{ij} and e_{ik}. The simplification is reasonable with the following aspects. For node i, if e_{ij} and e_{ik} are large, then the neighbor nodes j and k should be considered as important factors for the representation of node i. Compared to other interacting terms, the interaction between node j and k are likely to provide more relevant information about node i. Consequently, β_{jk} should be large. In contrast, if e_{ij} and e_{ik} are small, neighbor nodes j and k may have a slight impact on node i. Thus the interacting coefficient should be small as well. Formally, we arrive at:

$$\boldsymbol{h}_i^{\text{ir}} = \left(\sum_{j \in \mathcal{N}_i} e_{ij} \boldsymbol{n}_j \right) \odot \left(\sum_{k \in \mathcal{N}_i} e_{ik} \boldsymbol{n}_k \right) \\ = \left(\sum_{j \in \mathcal{N}_i} e_{ij} \boldsymbol{h}_j \boldsymbol{W} \right) \odot \left(\sum_{k \in \mathcal{N}_i} e_{ik} \boldsymbol{h}_k \boldsymbol{W} \right) = \boldsymbol{h}_i^{\text{agg}} \odot \boldsymbol{h}_i^{\text{agg}}, \quad (9)$$

where $\boldsymbol{h}_i^{\text{agg}} = \sum_{j \in \mathcal{N}_i} e_{ij} \boldsymbol{h}_j \boldsymbol{W}$ denotes the representation resulting from neighborhood aggregation.

In order to introduce more non-linearity to our model, we apply non-linear activation function on the two representations resulting from neighborhood aggregation and neighborhood interaction respectively. Besides, to combine these two representations, we add them using a skip connection:

$$\boldsymbol{h}_i^{\text{air}} = \sigma(\boldsymbol{h}_i^{\text{agg}}) + \sigma(\boldsymbol{h}_i^{\text{ir}}). \quad (10)$$

However, although we adopt a skip connection here, we argue that we still cannot benefit from residual learning, where both of the suboptimal representation and the residual function are implemented by *different non-linear layers*. As formulated in Eqs. (9,10), the two representations resulting from neighborhood aggregation and interaction are based on the *same* weight matrix

$\boldsymbol{W}$, which means the variations of the two representations during the back-propagation process are highly correlated. According to Bengio et al. [15], it is important to disentangle the factors of variation to the representations as only a few factors tend to change at a time. Therefore, to make use of residual learning which can ease the optimization, we introduce another weight matrix $\boldsymbol{W}' \in \mathbb{R}^{d_k \times d_{k+1}}$ to disentangle learning the neighborhood interaction from neighborhood aggregation. Formally, instead of Eq. (9), we use the following equation to learn the neighborhood interaction in our model:

$$\boldsymbol{h}_i^{\text{ir}} = \left(\sum_{j \in \mathcal{N}_i} e_{ij} \boldsymbol{h}_j \boldsymbol{W} \right) \odot \left(\sum_{k \in \mathcal{N}_i} e_{ik} \boldsymbol{h}_k \boldsymbol{W}' \right) = \boldsymbol{h}_i^{\text{agg}} \odot \bar{\boldsymbol{h}}_i^{\text{agg}}, \tag{11}$$

where the first term $\boldsymbol{h}_i^{\text{agg}} = \sum_{j \in \mathcal{N}_i} e_{ij} \boldsymbol{h}_j \boldsymbol{W}$ denotes the representation resulting from neighborhood aggregation and the second term $\bar{\boldsymbol{h}}_i^{\text{agg}} = \sum_{k \in \mathcal{N}_i} e_{ik} \boldsymbol{n}_k \boldsymbol{W}'$ provides the other half node representation for multiplication in the interaction process. $\boldsymbol{h}_i^{\text{agg}}$ is the input representation to the residual module and $\boldsymbol{W}'$ is the learnable weight of the residual function. Note that both terms $\boldsymbol{h}_i^{\text{agg}}$ and $\bar{\boldsymbol{h}}_i^{\text{agg}}$ can be implemented by existing graph convolutional layers. Thus the proposed GraphAIR framework is compatible with most existing GCN-based models and it provides a plug-and-play module for the neighborhood interaction.

3.2   Learning the Parameters of GraphAIR

In this section, we introduce how to learn the parameters under the GraphAIR framework. As we aim to propose a general approach for graph representation learning, we can apply different kinds of graph-based loss function, such as the proximity ranking loss in link prediction tasks and the cross-entropy loss in node classification tasks. Without loss of generality, we take the task of node classification as an example.

To compute the probability that each node belongs to a certain class, existing GCN-based models usually employ one additional graph convolutional layer with a softmax classifier for prediction. Then, the output representation $\boldsymbol{z}_i$ is formulated as:

$$\boldsymbol{z}_i = g\left(\boldsymbol{h}_i^{(k)} \right) = \text{softmax} \left(\sum_{j \in \mathcal{N}_i} e_{ij} \boldsymbol{h}_j^{(k)} \boldsymbol{W}^{(k+1)} \right), \tag{12}$$

where $g(\cdot)$ is the prediction function, $\boldsymbol{W}^{(k+1)} \in \mathbb{R}^{d_k \times |\mathcal{Y}|}$, and $|\mathcal{Y}|$ is the number of classes. Then, the loss of node classification can be calculated as $\mathcal{L} = \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}_{\text{clf}}(\boldsymbol{z}_i, \boldsymbol{y}_i)$ where $\boldsymbol{y}_i$ is the true label for node i and $\mathcal{L}_{\text{clf}}$ is the cross-entropy loss.

To obtain more accurate node embeddings $\boldsymbol{h}_i^{\text{agg}}$ and $\bar{\boldsymbol{h}}_i^{\text{agg}}$, we apply two auxiliary classifiers on $\boldsymbol{h}_i^{\text{agg}}$ and $\bar{\boldsymbol{h}}_i^{\text{agg}}$. Subsequently, the resulting representation $\boldsymbol{h}_i^{\text{ir}}$ for the neighborhood interaction will be more precise as well. Then, as formulated in Eq. (12), we apply one additional graph convolutional layer on each of $\boldsymbol{h}_i^{\text{air}}$, $\boldsymbol{h}_i^{\text{agg}}$, and $\bar{\boldsymbol{h}}_i^{\text{agg}}$ to attain $\boldsymbol{z}_i^{\text{air}}$, $\boldsymbol{z}_i^{\text{agg}}$, and $\bar{\boldsymbol{z}}_i^{\text{agg}}$. Eventually, the overall objective function is the weighted sum of the three losses:

$$\mathcal{L}_{\text{total}} = \frac{1}{n} \sum_{i=1}^{n} \left[\lambda_1 \mathcal{L}_{\text{clf}}(\boldsymbol{z}_i^{\text{air}}, \boldsymbol{y}_i) + \lambda_2 \mathcal{L}_{\text{clf}}(\boldsymbol{z}_i^{\text{agg}}, \boldsymbol{y}_i) + \lambda_3 \mathcal{L}_{\text{clf}}(\bar{\boldsymbol{z}}_i^{\text{agg}}, \boldsymbol{y}_i) \right]$$
$$= \lambda_1 \mathcal{L}^{\text{air}} + \lambda_2 \mathcal{L}^{\text{agg}} + \lambda_3 \bar{\mathcal{L}}^{\text{agg}}, \tag{13}$$

where λ_1, λ_2, and λ_3 are hyperparameters controlling weights of the three loss functions. For training, we minimize the total loss $\mathcal{L}_{\text{total}}$, while for inference, we only use $\boldsymbol{z}_i^{\text{air}}$, since $\boldsymbol{z}_i^{\text{agg}}$ and $\bar{\boldsymbol{z}}_i^{\text{air}}$ are to ensure $\boldsymbol{h}_i^{\text{air}}$ is accurate enough.

3.3   Model Architecture and Complexity Analysis

We suppose there are $(K + 1)$ layers in the underlying graph convolutional model, where the last layer is employed for node classification. For GraphAIR, we employ two separate and symmetric branches, each of which consists of K graph convolutional layers to obtain $\boldsymbol{h}_i^{\text{agg}}$ and $\bar{\boldsymbol{h}}_i^{\text{agg}}$. Then, considering $\boldsymbol{h}_i^{\text{agg}}$ and $\bar{\boldsymbol{h}}_i^{\text{agg}}$ have aggregated enough information from neighborhoods, here we conduct the neighborhood interaction only once by multiplying $\boldsymbol{h}_i^{\text{agg}}$ and $\bar{\boldsymbol{h}}_i^{\text{agg}}$ for the sake of efficiency. Additionally, we employ three graph convolutional layers followed by softmax activation functions on $\boldsymbol{h}_i^{\text{air}}$, $\boldsymbol{h}_i^{\text{agg}}$, and $\bar{\boldsymbol{h}}_i^{\text{agg}}$. In summary, there will be $(2K + 3)$ layers in GraphAIR.

Each layer in GraphAIR has the same space and time complexity as the underlying model and the additional computation cost of GraphAIR is mainly introduced by the multiplication process for the neighborhood interaction. For the neighborhood interaction in Eq. (11), the cost is $O(nd)$ where d is the embedding dimension. For each layer of the existing graph convolutional model such as GCN and GAT, it takes $O(n^2 d)$ time to proceed Eq. (2). Therefore, the additional computation cost of neighborhood interaction is insignificant. That is to say, our proposed approach is as asymptotically efficient as the underlying graph convolutional model.

4   EVALUATION

We extensively evaluate our proposed GraphAIR model on the node classification task and link prediction using five public datasets. Besides, we also conduct ablation studies on the neighborhood interaction module. For readers of interest, we include comparison of training time and all details of the experimental configurations in the supplementary material.

4.1   Datasets

We use five widely-used datasets to evaluate model performance on both transductive learning and inductive learning scenarios. Specifically, three citation networks (Cora, Citeseer, Pubmed) are used for tranductive node classification and link prediction, one knowledge graph (NELL) is used for transductive node classification, and one multi-graph molecular network (PPI) is for inductive node classification. We exactly follow the setup in [1, 2, 16, 17]. The statistics of datasets used throughout the experiments are summarized in Table 1.

Citation networks. We build undirected citation networks from three datasets, where documents and citations are treated as nodes and edges respectively. We treat the bag-of-words of each document as the feature vector. Our goal is to predict the class of each document. Only twenty labels per class are used for training.

Knowledge graph. The dataset collected from the knowledge base of Never Ending Language Learning (NELL) contains entities, relations, and text description. For every triplet (e_1, r, e_2), where e_1 and e_2 are entities and r is the relationship between them, r will be assigned with two separate nodes r_1 and r_2. Then, we add two edges between (e_1, r_1) and (e_2, r_2). For the knowledge graph, we conduct the entity classification. Similarly, we use bag-of-words as feature vectors. Only one label per class is used for training.

TABLE 1: Dataset statistics.

Dataset	Cora	Citeseer	Pubmed	NELL	PPI
Task		Transductive			Inductive
Type		Citation network		Knowledge graph	Molecular
# Vertices	2,708	3,327	19,717	65,755	56,944
# Edges	5,429	4,732	44,338	266,144	818,716
# Classes	7	6	3	210	121
# Features	1,433	3,703	500	5,414	50
# Training nodes	140	120	60	210	44,906
# Test nodes	1,000	1,000	1,000	1,000	5,524
# Validation nodes	500	500	500	500	6,514

Molecular network. We use the PPI (protein-protein interaction) network that consists of twenty-four (24) graphs corresponding to different human tissues. Each node contains fifty (50) features composed of positional gene sets, motif gene sets, and immunological signatures. We select twenty (20) graphs as the training set, two (2) for validation, and two (2) for testing.

4.2 Experiments on Node Classification

4.2.1 Baseline Methods

We comprehensively compare our method with various traditional random-walk-based algorithms and state-of-the-art GCN-based methods. We closely follow the experimental setting of previous work; the performance of those baselines is reported as in their original papers.[2]

Transductive node classification. In the transductive setting, the baselines include skip-gram-based network embedding method DeepWalk [18], graph convolutional networks with higher-order Chebyshev filters (Planetoid) [16], graph convolution with one-hop neighbors (GCN) [1], and graph attention networks (GAT) [2]. In addition, we further compare the performance of the proposed model with the recently proposed simplified graph convolutional networks (SGCs) [19] which removes redundant nonlinear activations. Also, we modify graph isomorphic networks (GINs) [20] which utilize non-linear MLPs as the aggregation function for the node classification task. Note that since GIN was originally proposed for graph classification, we apply two GIN convolutional layers and remove the graph-level readout function for the transductive node classification task.

Inductive node classification. For inductive node classification, we mainly compare GraphAIR with inductive graph convolutional networks (GraphSAGE) [7] and graph attention networks (GAT) [2]. Note that GraphSAGE provides several variants of neighborhood aggregators: SAGE-GCN concatenates the features of the neighborhoods and the central node, SAGE-mean takes the average over neighborhood feature vectors, SAGE-LSTM combines neighborhood features by using a LSTM model, and SAGE-pool uses an element-wise max-pooling operator to aggregate the neighborhood information nonlinearly.

4.2.2 Experimental Configurations

We employ our GraphAIR framework on top of three representative models, including GCN, GraphSAGE, and GAT, which is denoted by AIR-GCN, AIR-SAGE, and AIR-GAT, respectively. Particularly, while GraphSAGE proposes several variants

2. In experiments, we found that the results reported in Hamilton et al. [7] after ten epochs did not converge to the best values. For a fair comparison with other models, we reuse its official implementation and report the results of the baselines after 200 epochs.

for neighborhood aggregation, among them only SAGE-mean satisfies the coefficient normalization in Eq. (11). Therefore, we select SAGE-mean as the base model for GraphAIR. For a fair comparison, we closely follow the same hyper-parameters setting as the underlying graph convolutional model, such as learning rate, dropout rate, weight decay factor, hidden dimensions, etc. Considering GIN is originally proposed for graph-level classification, the hidden dimensions are set to the same as GCN. In the experiment, we only tune the weights of three loss functions by grid search, where $\lambda_i \in [0.1, 0.2, \ldots, 1.5], \forall i \in \{1, 2, 3\}$. For the transductive setting, we use the features of all data but only the labels of the training set are used for training. For the inductive setting, we train our model without the validation data and testing data. In addition, we report the average accuracy of 20 measurements.

4.2.3 Results and Analysis

Transductive. We summarize the results of transductive node classification in Table 2a. Note that even we apply the sparse version implementation of GAT, it requires more than 64G memory on NELL dataset. Thus, the performance of GAT and AIR-GAT is not reported. From the tables, it is seen that GraphAIR achieves state-of-the-art performance over all datasets, which demonstrates the effectiveness of the proposed GraphAIR framework. SGC acquires comparable results to that of GCN, which corresponds to our conclusion in Proposition 1 that existing GCNs are not able to learn the nonlinearity of graph data sufficiently. For our proposed AIR-GCN, it outperforms its base model GCN by margins of 3.2%, 2.6%, 1.0%, and 2.5%. The same trends hold for AIR-GAT with its base model GAT as well. To sum up, the improvements demonstrate the effectiveness of modeling the nonlinear distributions of nodes.

In addition, another important observation is that, both AIR-GAT and AIR-GCN outperform the complex non-linear opponents such as GIN. Although MLPs are able to asymptotically approximate any complicated and non-linear functions theoretically, they tend to converge to undesired local minima in practice [21]. The experimental results prove the rationality of *explicitly* introducing neighborhood interaction.

Inductive. The results of inductive learning are shown in Table 2b. AIR-SAGE-mean outperforms its base model SAGE-mean by 1.8%. Besides, we can clearly observe that AIR-GAT achieves the best performance. It is worth noting that the previous state-of-the-art method has already reached pretty high performance and the proposed AIR-GAT still acquires the improvement of 1.3% over the vanilla GAT. Besides, it is suggested that the proposed GraphAIR framework is also generalizable for multiple graphs.

4.3 Experiments on Link Prediction

In order to further verify our proposed framework is general for other graph representation learning tasks, we conduct experiments on link prediction additionally. We choose citation networks as benchmark datasets and compare against various state-of-the-art methods, including graph autoencoders (GAE) [17] and variational graph autoencoders (VGAE) [17], as well as other baseline algorithms, including SC [22] and DeepWalk [18]. We employ our GraphAIR framework on the basis of GAE, which constructs the graph autoencoder with GCNs. The resulting model is denoted by AIR-GAE.

We report the performance in terms of area under the ROC curve (AUC) based on the performance of 20 runs. The mean

TABLE 2: Accuracies of node classification with the best performance highlighted in bold.

(a) Transductive						(b) Inductive	
Method	Cora	Citeseer	Pubmed	NELL		Method	PPI
DeepWalk	67.2%	43.2%	65.3%	58.1%		Random	39.6%
Planetoid	75.7%	64.7%	77.2%	61.9%		SAGE-GCN	55.6%
GIN+0	78.3%	62.9%	78.0%	65.5%		SAGE-mean	64.5%
GIN+ϵ	76.6%	63.8%	75.5%	63.5%		SAGE-LSTM	66.8%
GCN	81.5%	70.3%	79.0%	73.0%		SAGE-pool	73.8%
SGC	81.0% $\pm$ 0.0%	71.9% $\pm$ 0.1%	78.9% $\pm$ 0.0%	72.8%		GAT	97.3% $\pm$ 0.2%
GAT	83.0% $\pm$ 0.7%	72.5% $\pm$ 0.7%	79.0% $\pm$ 0.3%	–			
						AIR-SAGE-mean	66.3% $\pm$ 0.1%
AIR-GCN	**84.7% $\pm$ 0.1%**	**72.9% $\pm$ 0.1%**	**80.0% $\pm$ 0.1%**	**75.5% $\pm$ 0.2%**		AIR-GAT	**98.6% $\pm$ 0.2%**
AIR-GAT	**84.5% $\pm$ 0.7%**	**73.5% $\pm$ 0.6%**	**80.0% $\pm$ 0.2%**	–			

TABLE 3: AUC of link prediction in citation networks with the best performance highlighted in bold.

Method	Cora	Citeseer	Pubmed
SC	84.6% $\pm$ 0.01%	80.5% $\pm$ 0.01%	84.2% $\pm$ 0.02%
DeepWalk	83.1% $\pm$ 0.01%	80.5% $\pm$ 0.02%	84.4% $\pm$ 0.00%
GAE	91.0% $\pm$ 0.02%	89.5% $\pm$ 0.04%	96.4% $\pm$ 0.00%
VGAE	91.4% $\pm$ 0.01%	90.8% $\pm$ 0.02%	94.4% $\pm$ 0.02%
AIR-GAE	**95.4% $\pm$ 0.01%**	**95.0% $\pm$ 0.01%**	**99.2% $\pm$ 0.02%**

TABLE 4: Accuracies of node classification in the ablation study.

Method	Cora	Citeseer	Pubmed
GCN	81.5%	70.3%	79.0%
DP-GCN	82.3% $\pm$ 0.1%	71.0% $\pm$ 0.1%	79.0% $\pm$ 0.2%
self-IR-GCN	82.6% $\pm$ 0.0%	70.8% $\pm$ 0.2%	79.2% $\pm$ 0.1%
AIR-GCN	84.7% $\pm$ 0.1%	72.9% $\pm$ 0.1%	80.0% $\pm$ 0.1%

performance and standard error are presented in Table 3. It is shown from the table that the proposed AIR-GAE outperforms its vanilla opponents GAE and VGAE, which once again verifies the necessity to incorporate the neighborhood interaction to neighborhood aggregation. Please note that previous state-of-the-art methods have already obtained high enough performance on the Pubmed dataset and our method AIR-GAE pushes the boundary with absolute improvements of 2.8%, achieving 99.2% in terms of AUC. Also, it can be observed that the proposed method obtains much more obvious improvements, compared with the performance of node classification. We suspect that this is primarily because models for the link prediction task usually employ pairwise decoders for calculating the probability of the link between two nodes. For example, GAE and VGAE assume the probability that there exists an edge between two nodes is proportional to the dot product of the embeddings of these two nodes. Therefore, our approach, which explicitly models the neighborhood interaction through the multiplication of the embeddings of two nodes, is inherently related to the link prediction task and obtains more improvements.

4.4 Ablation Studies on the Neighborhood Interaction Module

As we analyzed in Section 3.3, the number of parameters in GraphAIR is almost two times than that of the underlying graph convolutional model. In this section, we conduct ablation studies to answer the following questions:

- **Q1**: How much improvement has the proposed neighborhood interaction module brought?
- **Q2**: Does the disentangled residual learning strategy bring sufficient improvements?

To answer Q1 and verify the effectiveness of GraphAIR is introduced by the proposed neighborhood interaction module

rather than the larger number of parameters in the model, we remove the neighborhood interaction module of AIR-GCN. Then, the resulting model has exactly the same parameters as AIR-GCN. As there are almost double parameters than vanilla GCN in the resulting model, we denote the resulting model as DP-GCN (Double-Parameter GCN).

To answer Q2, we employ only one branch of graph convolutional networks consisting $(K + 1)$ layers to produce the output representations. To obtain neighborhood interaction h^{ir}, we directly make use of the self-interaction strategy described in Eq. (9) instead of Eq. (11). The resulting model is termed as self-IR-GCN.

For a fair comparison, other experimental configurations are kept the same as AIR-GCN. The results of node classification are presented in Table 4. It is seen from the table that the proposed AIR-GCN achieves the best performance and outperforms DP-GCN and self-IR-GCN. For Q1, we can observe that DP-GCN only obtains slightly better accuracy on Cora and Citeseer and almost the same performance as the vanilla GCN on Pubmed. It can be verified that the neighborhood interaction module mainly contributes to the performance improvement of the proposed AIR-GCN model. For Q2, it is seen that the performance of self-IR-GCN only gets slightly improved on three datasets, which demonstrates the rationality of modeling neighborhood interaction. However, disengaging the neighborhood interaction from neighborhood aggregation can bring more improvements.

5 RELATED WORK

There have been a lot of attempts in recent literature to employ neural networks for graph representation learning. Among them, graph convolutional neural networks (GCNs) receive a lot of research interests. GCN-based models generally follow the neighborhood aggregation scheme. To be specific, the model passes the input signals from neighborhoods through filters to aggregate information. Many approaches design different strategies to aggregate information from nodes' neighborhood. According to

different strategies, these models can be roughly grouped into two categories, i.e. spectral-based approaches and spatial-based approaches.

One the one hand, spectral methods depend on the Laplacian eigenbasis to define parameterized filters. The first work [23] introduce convolutional operations in the Fourier domain by computing the eigendecomposition of the graph Laplacian, which results in potentially heavy computational burden. Following its work, Defferrard et al. [24] propose to approximate filters using Chebyshev expansion of the graph Laplacian. Then, graph convolutional neural networks (GCNs) [1] have been widely applied for graph representation learning. The core of GCNs is the neighborhood aggregation scheme which generates node embedding by combining information from neighborhoods. Since GCN only captures local information, DGCN [5] then proposes to construct an information matrix to encode global consistency.

On the other hand, the spatial approaches directly operate on spatially close neighbors. To enable parameter sharing of filters across neighbors of different sizes, Duvenaud et al. [25] first propose to learn weight matrices for different node degrees. MoNet [26] proposes a spatial-domain model to provide a unified convolutional network on graphs. To compute node representations in an inductive manner, GraphSAGE [7] samples fixed-size neighborhoods of nodes and performs aggregation over them. Similarly, Gao et al. [27] select a fixed number of neighbors and enable the use of conventional convolutional operations on Euclidean spaces. Recently, GAT [2] introduces attention mechanisms to graph neural networks, which computes hidden representations by attending over neighbors with a self-attention strategy.

Recently, some methods are proposed to focus on linearity and non-linearity of graphs respectively. On the one hand, simplified graph convolutional networks (SGCs) [19] try to reduce the complexity and eliminate redundant computation of GCN by successively removing non-linear activation functions. SGC makes assumptions that non-linearity between GCN layers is not critical to the model performance and the majority of the benefit is brought by the neighborhood aggregation scheme. While being more computationally efficient, SGC achieves comparable empirical performance to vanilla GCN.

There are other methods arguing that modeling non-linear distributions of node features can bring improvements. For example, GraphSAGE-LSTM [7] employs the long-short-term memory (LSTM) module to learn the complex relationships between the nodes. Empirically, GraphSAGE-LSTM outperforms other aggregation functions such as GraphSAGE-mean and GraphSAGE-GCN. Graph isomorphic networks (GIN) [20] apply multilayer perceptrons (MLPs) in each graph convolutional layer, which is able to model complex non-linearity of graphs. Although theoretically it is well known that MLPs are universal approximators [28], there is no formal theorem giving instructions on how to asymptotically approximate the desired function (Patterson 1998, p. 182; Fausett 1994, p. 328). Different from GraphSAGE-LSTM and GIN, to best of our knowledge, our work is the first to point out that most existing GCNs may not well capture non-linearity of graph data and we demonstrate the effectiveness of explicitly modeling non-linearity of graphs.

6 CONCLUSION

In this paper, we have firstly proved that existing mainstream GCN-based models have difficulty in well capturing the complicated non-linearity of graph data. Then, in order to better capture the complicated and non-linear distributions of nodes, we have proposed a novel GraphAIR framework that explicitly models the neighborhood interaction in addition to the neighborhood aggregation scheme. By employing residual learning strategy, we disentangle learning the neighborhood interaction from the neighborhood aggregation, which makes the optimization easier. The proposed GraphAIR is compatible with most existing graph convolutional models and it can provide a plug-and-play module for the neighborhood interaction. Finally, GraphAIR based on well-known models including GCN, GraphSAGE, and GAT have been thoroughly investigated through empirical evaluation. Extensive experiments on benchmark tasks including node classification and link prediction demonstrate the effectiveness of our model.

REFERENCES

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

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

[3] M. Zhang and Y. Chen, "Link prediction based on graph neural networks," in *NIPS*, 2018, pp. 5167–5177.

[4] Z. Chen, L. Li, and J. Bruna, "Supervised community detection with line graph neural networks," in *ICLR*, 2019.

[5] C. Zhuang and Q. Ma, "Dual graph convolutional networks for graph-based semi-supervised classification," in *WWW*, 2018, pp. 499–508.

[6] K. Xu, C. Li, Y. Tian, T. Sonobe, K.-i. Kawarabayashi, and S. Jegelka, "Representation learning on graphs with jumping knowledge networks," in *ICML*, 2018, pp. 5453–5462.

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

[8] B. E. Boser, I. M. Guyon, and V. N. Vapnik, "A training algorithm for optimal margin classifiers," in *COLT*, 1992, pp. 144–152.

[9] C. Harris and M. Stephens, "A combined corner and edge detector," in *Alvey Vision Conference*, 1988, pp. 147–151.

[10] Y. LeCun, B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural Computation*, vol. 1, no. 4, pp. 541–551, 1989.

[11] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *AISTATS*, 2010, pp. 249–256.

[12] Q. Li, Z. Han, and X.-M. Wu, "Deeper insights into graph convolutional networks for semi-supervised learning," in *AAAI*, 2018, pp. 3538–3545.

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

[14] H. Li, Z. Xu, G. Taylor, C. Studer, and T. Goldstein, "Visualizing the loss landscape of neural nets," in *NeurIPS*, 2018, pp. 6389–6399.

[15] Y. Bengio, A. C. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *T-PAMI*, vol. 35, no. 8, pp. 1798–1828, 2013.

[16] Z. Yang, W. W. Cohen, and R. Salakhutdinov, "Revisiting semi-supervised learning with graph embeddings," in *ICML*, 2016, pp. 40–48.

[17] T. N. Kipf and M. Welling, "Variational Graph Auto-Encoders," in *Bayesian Deep Learning Workshop (NIPS 2016)*, 2016.

[18] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *KDD*, 2014, pp. 701–710.

[19] F. Wu, T. Zhang, A. H. de Souza Jr., C. Fifty, T. Yu, and K. Q. Weinberger, "Simplifying graph convolutional networks," in *ICML*, 2019, pp. 6861–6871.

[20] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" in *ICLR*, 2019.

[21] R. Rojas, *Neural Networks: A Systematic Introduction*, 1996.

[22] L. Tang and H. Liu, "Leveraging Social Media Networks for Classification," *Data Mining and Knowledge Discovery*, vol. 23, no. 3, pp. 447–478, 2011.

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

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

[25] D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams, "Convolutional networks on graphs for learning molecular fingerprints," in *NIPS*, 2015, pp. 2224–2232.

[26] F. Monti, D. Boscaini, J. Masci, E. Rodolà, J. Svoboda, and M. M. Bronstein, "Geometric deep learning on graphs and manifolds using mixture model cnns," in *CVPR*, 2017, pp. 5425–5434.

[27] H. Gao, Z. Wang, and S. Ji, "Large-scale learnable graph convolutional networks," in *KDD*, 2018, pp. 1416–1424.

[28] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural Networks*, vol. 4, no. 2, pp. 251–257, 1991.

[29] D. W. Patterson, *Artificial Neural Networks: Theory and Applications*, 1998.

[30] L. Fausett, *Fundamentals of Neural Networks: Architectures, Algorithms, and Applications*, 1994.