Bilinear Graph Neural Network with Neighbor Interactions *

Hongmin Zhu[1], **Fuli Feng**[2], **Xiangnan He**[1], **Xiang Wang**[2]
Yan Li[3], **Kai Zheng**[4] and **Yongdong Zhang**[1]
[1]University of Science and Technology of China
[2]National University of Singapore
[3]Beijing Kuaishou Technology Co., Ltd. Beijing, China
[4]University of Electronic Science and Technology of China
{zhuhm@mail., hexn@, zhyd73@}ustc.edu.cn, {dcsfeng, dcswxi}@nus.edu.sg
liyan@kuaishou.com, zhengkai@uestc.edu.cn

Abstract

Graph Neural Network (GNN) is a powerful model to learn representations and make predictions on graph data. Existing efforts on GNN have largely defined the graph convolution as a weighted sum of the features of the connected nodes to form the representation of the target node. Nevertheless, the operation of weighted sum assumes the neighbor nodes are independent of each other, and ignores the possible interactions between them. When such interactions exist, such as the co-occurrence of two neighbor nodes is a strong signal of the target node's characteristics, existing GNN models may fail to capture the signal. In this work, we argue the importance of modeling the interactions between neighbor nodes in GNN. We propose a new graph convolution operator, which augments the weighted sum with pairwise interactions of the representations of neighbor nodes. We term this framework as *Bilinear Graph Neural Network* (BGNN), which improves GNN representation ability with bilinear interactions between neighbor nodes. In particular, we specify two BGNN models named BGCN and BGAT, based on the well-known GCN and GAT, respectively. Empirical results on three public benchmarks of semi-supervised node classification verify the effectiveness of BGNN — BGCN (BGAT) outperforms GCN (GAT) by 1.6% (1.5%) in classification accuracy. Codes are available at: https://github.com/zhuhm1996/bgnn.

1 Introduction

GNN is a kind of neural networks that performs neural network operations over graph structure to learn node representations. Owing to the ability to learn more comprehensive node representations than the models that consider only node features [Yang *et al.*, 2016] or graph structure [Perozzi *et al.*, 2014], GNN has been a promising solution for a wide range of applications in social science [Chen *et al.*, 2018], computer vision [Kampffmeyer *et al.*, 2019], and recommendation [Wang *et al.*, 2019; He *et al.*, 2020] *etc.* To date, most graph convolution operations in GNNs are implemented as a linear aggregation (*i.e.,* weighted sum) over features of the neighbors of the target node [Kipf and Welling, 2017]. Although it improves the representation of the target node, such linear aggregation assumes that the neighbor nodes are independent of each other, ignoring the possible interactions between them.

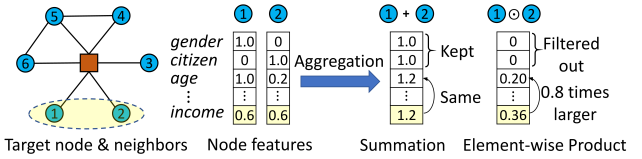


Figure 1: **Left**: A toy example of a target node and its neighbors in a transaction graph where nodes are described by a set of numerical features. **Right**: Results of aggregating the features of node 1 and 2 with summation and element-wise product operations, respectively.

Under some circumstances, the interactions between neighbor nodes could be a strong signal that indicates the characteristics of the target node. Figure 1 (left) illustrates a toy example of a target node and its neighbors in a transaction graph, where edges denote money transfer relations and nodes are described by a set of features such as age and income. The interaction between node 1 and 2, which indicates that both have high incomes, could be a strong signal to estimate the credit rating of the target node (an intuition is that a customer who has close business relations with rich friends would have a higher chance to repay a loan). Explicitly modeling such interactions between neighbors highlights the common properties within the local structure, which could be rather helpful for the target node's representation. In Figure 1 (right), we show that the summation-based linear aggregator — a common choice in existing GNNs — fails to highlight the income feature. In contrast, by using a multiplication-based aggregator that captures node interactions, the signal latent in shared high incomes is highlighted, and as an auxiliary effect, some less useful features are zeroed out.

*This work is supported by the National Natural Science Foundation of China (U19A2079, 61525206, 61972069, 61836007, 61832017). Fuli Feng is the corresponding author and contributes equally as Hongmin Zhu.

Nevertheless, it is non-trivial to encode such local node interactions in GNN. The difficulty mainly comes from two indispensable requirements of a feasible graph convolution operation: 1) *permutation invariant* [Xu *et al.*, 2019b], *i.e.,* the output should remain the same when the order of neighbor nodes is changed, so as to ensure the stability of a GNN; and 2) *linear complexity* [Kipf and Welling, 2017], *i.e.,* the computational complexity should increase linearly with respect to the number of neighbors, so as to make a GNN scalable on large graphs. To this end, we take inspiration from neural factorization machines [He and Chua, 2017] to devise a new *bilinear aggregator*, which explicitly expresses the interactions between every two nodes and aggregates all pair-wise interactions to enhance the target node's representation.

On this basis, we develop a new graph convolution operator which is equipped with both traditional linear aggregator and the newly proposed bilinear aggregator, and is proved to be permutation invariant. We name the new model as *Bilinear Graph Neural Network* (BGNN), which is expected to learn more comprehensive representations by considering local node interactions. We devise two BGNN models, named BGCN and BGAT, which are equipped with the GCN and GAT linear aggregator, respectively. Taking semi-supervised node classification as an example task, we evaluate BGCN and BGAT on three benchmark datasets to validate their effectiveness. Specifically, BGCN and BGAT outperform GCN and GAT by 1.6% and 1.5%, respectively. More fine-grained analyses show that the improvements on sparsely connected nodes are more significant, demonstrating the strengths of the bilinear aggregator in modeling node interactions. The main contributions of this paper are summarized as:

- We propose BGNN, a simple yet effective GNN framework, which explicitly encodes the local node interactions to augment conventional linear aggregator.
- We prove that the proposed BGNN model has the properties of permutation invariant and linear computation complexity which are of importance for GNN models.
- We conduct extensive experiments on three public benchmarks of semi-supervised node classification, validating the effectiveness of the proposed BGNN models.

2 Related Work

GNN generalizes traditional convolutional neural networks from Euclidean space to graph domain. According to the format of the convolution operations, existing GNN models can be divided into two categories: spatial GNN and spectral GNN [Zhang *et al.*, 2018]. We separately review the two kinds of models, and refer the mathematical connection between them to [Bronstein *et al.*, 2017].

Spectral GNN. Spectral GNN is defined as performing convolution operations in the Fourier domain with spectral node representations [Bruna *et al.*, 2014; Defferrard *et al.*, 2016; Kipf and Welling, 2017; Liao *et al.*, 2019; Xu *et al.*, 2019a]. Bruna *et al.* [Bruna *et al.*, 2014] define the convolution over the eigenvectors of graph Laplacian which are viewed as the Fourier basis. Considering the high computational cost of the eigen-decomposition, research on spectral GNN has been focused on approximating the decomposition with different mathematical techniques [Defferrard *et al.*, 2016; Kipf and Welling, 2017; Liao *et al.*, 2019; Xu *et al.*, 2019a]. For instance, [Defferrard *et al.*, 2016] introduce the Chebyshev polynomials with orders of K to approximate the eigen-decomposition. In [Kipf and Welling, 2017], Kipf and Welling simplify this model by limiting $K = 1$ and approximating the largest eigenvalue of Laplacian matrix by 2. In addition, Liao *et al.* [Liao *et al.*, 2019] employ the Lanczos algorithm to perform a low-rank approximation of the graph Laplacian. Recently, Wavelet transform is introduced to spectral GNN to discard the eigen-decomposition [Xu *et al.*, 2019a]. However, spectral GNN models are hard to be applied on large graphs such as social networks. This is because the convolution operations are required to be performed over the whole graph, posing unaffordable memory cost and incapacitating the widely applied batch training.

Spatial GNN. Spatial GNN instead performs convolution operations directly over the graph structure by aggregating the features from spatially close neighbors to a target node [Atwood and Towsley, 2016; Hamilton *et al.*, 2017; Kipf and Welling, 2017; Veličković *et al.*, 2018; Xu *et al.*, 2018; Xinyi and Chen, 2019; Veličković *et al.*, 2019; Xu *et al.*, 2019b; Feng *et al.*, 2019]. This line of research is mainly focused on developing aggregation methods from different perspectives. For instance, Kipf and Welling [Kipf and Welling, 2017] propose to use a linear aggregator (*i.e.,* weighted sum) that uses the reverse of node degree as the coefficient. To improve the representation performance, neural attention mechanism is introduced to learn the coefficients [Veličković *et al.*, 2018]. In addition to aggregating information from directly connected neighbors, augmented aggregators also account for multi-hop neighbors [Atwood and Towsley, 2016; Xu *et al.*, 2018]. Moreover, non-linear aggregators are also employed in spatial GNNs such as max pooling [Hamilton *et al.*, 2017], capsule [Veličković *et al.*, 2019], and Long Short-Term Memory (LSTM) [Hamilton *et al.*, 2017]. Furthermore, spatial GNN is extended to graphs with both static and temporal neighbors structure [Park and Neville, 2019] and representations in hyperbolic space [Chami *et al.*, 2019].

However, most existing aggregators (both linear and non-linear ones) forgo the importance of the interactions among neighbors. As built upon the summation operation, by nature, the linear aggregators assume that neighbors are independent. Most of the non-linear ones are focused on the property of neighbors at set level (*i.e.,* all neighbors), *e.g.,* the "skeleton" of the neighbors [Xu *et al.*, 2019b]. Taking one neighbor as the input of a time-step, LSTM-based aggregator could capture sequential dependency, which might include node interactions. However, it requires a predefined order on neighbor, violating permutation invariant and typically showing weak performance [Hamilton *et al.*, 2017]. Our work is different from those aggregators in that we explicitly consider pairwise node interactions in a neat and systematic way.

3 Bilinear Graph Neural Network

Preliminaries. Let $G = (\mathbf{A} \in \{0,1\}^{N\times N}, \mathbf{X} \in \mathbb{R}^{N\times F})$ be the graph of interest, where $\mathbf{A}$ is the binary adjacency matrix

where an element $A_{vi} = 1$ means that an edge exists between node v and i, and $\mathbf{X}$ is the original feature matrix for nodes that describes each node with a vector of size F (a row). We denote the neighbors of node v as $\mathcal{N}(v) = \{i|A_{vi} = 1\}$ which stores all nodes that have an edge with v, and denote the extended neighbors of node v as $\tilde{\mathcal{N}}(v) = \{v\} \cup \mathcal{N}(v)$ which contains the node v itself. For convenience, we use d_v to denote the degree of node v, *i.e.,* $d_v = |\mathcal{N}(v)|$, and accordingly $\tilde{d}_v = |\tilde{\mathcal{N}}(v)| = d_v + 1$. The model objective is to learn a representation vector $\mathbf{h}_v \in \mathbb{R}^D$ for each node v, such that its characteristics are properly encoded. For example, the label of node v can be directly predicted as a function output $y_v = f(\mathbf{h}_v)$, without the need of looking into the graph structure and original node features in G.

The spatial GNN [Veličković *et al.*, 2018] achieves this goal by recursively aggregating the features from neighbors:

$$\mathbf{h}_v^{(k)} = AGG\big(\{\mathbf{h}_i^{(k-1)}\}_{i\in\tilde{\mathcal{N}}(v)}\big) = \sum_{i\in\tilde{\mathcal{N}}(v)} a_{vi}\mathbf{h}_i^{(k-1)}\mathbf{W}^{(k)}, \quad (1)$$

where $\mathbf{h}_v^{(k)}$ denotes the representation of target node v at the k-th layer/iteration, $\mathbf{W}^{(k)}$ is the weight matrix (model parameter) to do feature transformation at the k-th layer, and the initial feature representation $\mathbf{h}_v^{(0)}$ can be obtained from the original feature matrix $\mathbf{X}$.

The AGG function is typically implemented as a weighted sum with a_{vi} as the weight of neighbor i. In GCN [Kipf and Welling, 2017], a_{vi} is defined as $1/\sqrt{\tilde{d}_v\tilde{d}_i}$, which is grounded on the Laplacian theories. The recent advance on graph attention network (GAT) [Veličković *et al.*, 2018] learns a_{vi} from data, which has the potential to lead better performance than pre-defined choices. However, a limitation of such weighted sum is that no interactions between neighbor representations are modeled. Although using more powerful feature transformation function such as multi-layer perceptron (MLP) [Xu *et al.*, 2019b] can alleviate the problem, the process is rather implicit and ineffective. An empirical evidence is from [Beutel *et al.*, 2018], which shows that MLP is inefficient in capturing the multiplication relations between input features. In this work, we propose to explicitly inject the multiplication-based node interactions into AGG function.

3.1 Bilinear Aggregator

As demonstrated in Figure 1, the multiplication between two vectors is an effective manner to model the interactions — emphasizing common properties and diluting discrepant information. Inspired by factorization machines (FMs) [Rendle, 2010; He and Chua, 2017] that have been intensively used to learn the interactions among categorical variables, we propose a bilinear aggregator which is suitable for modeling the neighbor interactions in local structure:

$$BA\big(\{\mathbf{h}_i\}_{i\in\tilde{\mathcal{N}}(v)}\big) = \frac{1}{b_v}\sum_{i\in\tilde{\mathcal{N}}(v)}\sum_{j\in\tilde{\mathcal{N}}(v)\&i<j} \mathbf{h}_i\mathbf{W} \odot \mathbf{h}_j\mathbf{W}, \quad (2)$$

where $\odot$ is element-wise product; v is the target node to obtain representation for; i and j are node index from the extended neighbors $\tilde{\mathcal{N}}(v)$ — they are constrained to be different to avoid self-interactions that are meaningless and may even introduce extra noises. $b_v = \frac{1}{2}\tilde{d}_v(\tilde{d}_v - 1)$ denotes the number of interactions for the target node v, which normalizes the obtained representation to remove the bias of node degree. It is worth noting that we take the target node itself into account and aggregate information from extended neighbors, which although looks same as GNN, but for different reasons. In GNN, accounting for the target node is to retain its information during layer-wise aggregation, working like the residual learning [He *et al.*, 2016]. While in BGNN, our consideration is that the interactions between the target node and its neighbors may also carry useful signal. For example, for sparse nodes that have only one neighbor, the interaction between neighbors does not exist, and the interaction between the target and neighbor nodes can be particularly helpful.

Time Complexity Analysis. At the first sight, the bilinear aggregator considers all pairwise interactions between neighbors (including the target node), thus may have a quadratic time complexity *w.r.t.* the neighbor count, being higher than the weighted sum. Nevertheless, through a mathematical reformulation similar to that one used in FM, we can compute the aggregator in linear time — $\mathcal{O}(|\tilde{\mathcal{N}}(v)|)$ — the same complexity as weighted sum. To show this, we rewrite Equation (2) in its equivalent form as:

$$BA\big(\{\mathbf{h}_i\}_{i\in\tilde{\mathcal{N}}(v)}\big) = \frac{1}{2b_v}\Bigg(\sum_{i\in\tilde{\mathcal{N}}(v)}\sum_{j\in\tilde{\mathcal{N}}(v)} \mathbf{s}_i \odot \mathbf{s}_j - \sum_{i\in\tilde{\mathcal{N}}(v)} \mathbf{s}_i \odot \mathbf{s}_i\Bigg) = \frac{1}{2b_v}\Bigg(\Big(\underbrace{\sum_{i\in\tilde{\mathcal{N}}(v)} \mathbf{s}_i}_{\mathcal{O}(|\tilde{\mathcal{N}}(v)|)}\Big)^2 - \underbrace{\sum_{i\in\tilde{\mathcal{N}}(v)} \mathbf{s}_i^2}_{\mathcal{O}(|\tilde{\mathcal{N}}(v)|)}\Bigg), \quad (3)$$

where $\mathbf{s}_i = \mathbf{h}_i\mathbf{W} \in \mathbb{R}^D$. As can be seen, through mathematical reformulation, we can reduce the sum over pairwise element-wise products to the minus of two terms, where each term is a weighted sum of neighbor representations (or their squares) and can be computed in $\mathcal{O}(|\tilde{\mathcal{N}}(v)|)$ time. Note that multiplying weight matrix $\mathbf{W}$ is a standard operation in aggregator thus its time cost is omitted for brevity.

Proof of Permutation Invariant. This property is intuitive to understand from the reduced Equation (3): when changing the order of input vectors, the sum of inputs (the first term) and the sum of the squares of inputs (the second term) are not changed. Thus the output is unchanged and the permutation invariant property is satisfied. To provide a rigorously proof, we give the matrix form of the bilinear aggregator, which also facilitates the matrix-wise implementation of BGNN. The matrix form of the bilinear aggregator is:

$$BA(\mathbf{H}, \mathbf{A}) = \frac{1}{2}\mathbf{B}^{-1}\Big(\big(\tilde{\mathbf{A}}\mathbf{H}\mathbf{W}\big)^2 - \tilde{\mathbf{A}}\big(\mathbf{H}\mathbf{W}\big)^2\Big), \quad (4)$$

where $\mathbf{H} \in \mathbb{R}^{N\times D}$ stores the representation vectors $\mathbf{h}$ for all nodes, $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ is the adjacency matrix of the graph with self-loop added on each node ($\mathbf{I} \in \mathbb{R}^{N\times N}$ is an identity matrix), $\mathbf{B}$ is a diagonal matrix with each element $B_{vv} = b_v$, and $(\cdot)^2$ denotes the element-wise product of two matrices.

Let $\mathbf{P} \in \mathbb{R}^{N\times N}$ be any permutation matrix that satisfies (1) $\mathbf{P}^T\mathbf{P} = \mathbf{I}$, and (2) for any matrix $\mathbf{M}$ if $\mathbf{PM}$ exists, then $\mathbf{PM} \odot$

$\mathbf{PM} = \mathbf{P}(\mathbf{M} \odot \mathbf{M})$ satisfies. When we apply the permutation $\mathbf{P}$ on the nodes, $\mathbf{H}$ changes to $\mathbf{PH}$, $\tilde{\mathbf{A}}$ changes to $\mathbf{P}\tilde{\mathbf{A}}\mathbf{P}^T$ and $\mathbf{B}$ changes to $\mathbf{PBP}^T$, which leads to:

$$BA(\mathbf{PH}, \mathbf{PAP}^T) = \frac{1}{2}\mathbf{PB}^{-1}\mathbf{P}^T\left(\mathbf{P}\left(\tilde{\mathbf{A}}\mathbf{HW}\right)^2 - \mathbf{P}\tilde{\mathbf{A}}\left(\mathbf{HW}\right)^2\right)$$
$$= \frac{1}{2}\mathbf{PB}^{-1}\mathbf{P}^T\mathbf{P}\left(\left(\tilde{\mathbf{A}}\mathbf{HW}\right)^2 - \tilde{\mathbf{A}}\left(\mathbf{HW}\right)^2\right) = \mathbf{P} \cdot BA(\mathbf{H}, \mathbf{A}),$$

which indicates the permutation invariant property.

3.2 BGNN Model

We now describe the proposed BGNN model. As the bilinear aggregator emphasizes node interactions and encodes different signal with the weighted sum aggregator, we combine them to build a more expressive graph convolutional networkWe adopt a simple linear combination scheme, defining a new graph convolution operator as:

$$\mathbf{H}^{(k)} = BGNN\left(\mathbf{H}^{(k-1)}, \mathbf{A}\right) \tag{5}$$
$$= (1-\alpha) \cdot AGG\left(\mathbf{H}^{(k-1)}, \mathbf{A}\right) + \alpha \cdot BA\left(\mathbf{H}^{(k-1)}, \mathbf{A}\right),$$

where $\mathbf{H}^{(k)}$ stores the node representations at the k-th layer (encoded k-hop neighbors). α is a hyper-parameter to trade-off the strengths of traditional GNN aggregator and our proposed bilinear aggregator. Figure 2 illustrates the model framework.

Since both AGG and BA are permutation invariant, it is trivial to find that this graph convolution operator is also permutation invariant. When α sets to 0, no node interaction is considered and BGNN degrades to GNN; when α sets to 1, BGNN only uses the bilinear aggregator to process the information from the neighbors. Our empirical studies show that an intermediate value between 0 and 1 usually leads to better performance, verifying the efficacy of modeling node interactions, and the optimal setting varies on different datasets.

Multi-layer BGNN. Traditional GNN models [Kipf and Welling, 2017; Veličković *et al.*, 2018; Xu *et al.*, 2019b] encode information from multi-hop neighbors in a recursive manner by stacking multiple aggregators. For example, the 2-layer GNN model is formalized as,

$$GNN_2(\mathbf{X}, \mathbf{A}) = \underbrace{AGG}_{\text{2nd layer}}\left(\sigma\left(\underbrace{AGG}_{\text{1st layer}}(\mathbf{X}, \mathbf{A})\right), \mathbf{A}\right), \tag{6}$$

where σ is a non-linear activation function. Similarly, we can devise a 2-layer BGNN model in the same recursive manner:

$$\underbrace{BGNN}_{\text{2nd layer}}\left(\sigma\left(\underbrace{BGNN}_{\text{1st layer}}(\mathbf{X}, \mathbf{A})\right), \mathbf{A}\right). \tag{7}$$

However, such a straightforward multi-layer extension involves unexpected higher-order interactions. In the two-layer case, the second-layer representation will include partial 4th-order interactions among the two-hop neighbors, which are hard to interpret and unreasonable. When extending BGNN to multiple layers, saying K layers, we still hope to capture pairwise interactions, but between the K-hop neighbors. To this end, instead of directly stacking $BGNN$ layers, we define the 2-layer $BGNN$ model as:

$$BGNN_2(\mathbf{X}, \mathbf{A}) = (1-\alpha) \cdot GNN_2(\mathbf{X}, \mathbf{A}) \tag{8}$$
$$+ \alpha[(1-\beta) \cdot BA(\mathbf{X}, \mathbf{A}) + \beta \cdot BA(\mathbf{X}, \mathbf{A}^{(2)})],$$

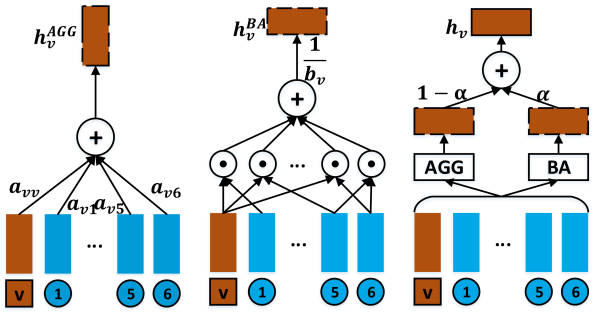
where $\mathbf{A}^{(2)} = binarize(\mathbf{AA})$ stores the 2-hop connectivities of the graph. $binarize$ is an entry-wise operation that transforms non-zero entries to 1. As such, a non-zero entry (v, i) in $\mathbf{A}^{(2)}$ means node v can reach node i within two hops. β is a hyper-parameter to trade-off the strengths of bilinear interactions within 1-hop neighbors and 2-hop neighbors.

Following the same principle, we define the K-layer BGNN as:

$$BGNN_K(\mathbf{X}, \mathbf{A}) = (1-\alpha) \cdot GNN_K(\mathbf{X}, \mathbf{A})$$
$$+ \alpha \cdot \left(\sum_{k=1}^{K} \beta_k \cdot BA(\mathbf{X}, \mathbf{A}^{(k)})\right), \; s.t., \sum_{k=1}^{K} \beta_k = 1, \tag{9}$$

where $\mathbf{A}^{(k)} = binarize\left(\underbrace{\mathbf{A} \cdots \mathbf{A}}_{k \text{ times}}\right)$ denotes the adjacency matrix of k-hop connectivities, and GNN_K denotes normal K-layer GNN that can be defined recursively such as a K-layer GCN or GAT. The time complexity of a K-layer BGNN is determined by the number of non-zero entries in $\mathbf{A}^{(K)}$. To reduce the actual complexity, one can follow the sampling strategy in GraphSage [Hamilton *et al.*, 2017], sampling a portion of high-hop neighbors rather than using all neighbors.

Model Training. BGNN is a differentiable model, thus it can be end-to-end optimized on any differential loss with gradient descent. In this work, we focus on the semi-supervised node classification task, optimizing BGNN with the cross-entropy loss on labeled nodes (same setting as the GCN work [Kipf and Welling, 2017] for a fair comparison). As the experimented data is not large, we implement the layer-wise graph convolution in its matrix form, leaving the batch implementation and neighbor sampling which can scale to large graphs as future work.



(a) AGG (b) BA (c) BGNN

Figure 2: An illustration of the traditional linear aggregator in GNN (a), bilinear aggregator (b), and BGNN aggregator (c).

4 Experiments

Datasets. Following previous works [Sen *et al.*, 2008; Yang *et al.*, 2016; Veličković *et al.*, 2018], we utilize three

Model	1-layer			RI	2-layer			RI
	Pubmed	Cora	Citeseer		Pubmed	Cora	Citeseer	
SemiEmb	-	-	-	-	71.1	59.0	59.6	26.4%
DeepWalk	-	-	-	-	65.3	67.2	43.2	39.6%
Planetoid	-	-	-	-	77.2	75.7	64.7	9.7%
GCN	76.9 ± 0.2	76.8 ± 0.2	69.1 ± 0.1	2.7%	79.0	81.5	70.3	3.2%
GAT	77.3 ± 0.4	78.3 ± 0.6	69.7 ± 1.1	1.6%	79.0 ± 0.3	83.0 ± 0.7	72.5 ± 0.7	1.5%
GIN	76.5 ± 0.1	77.5 ± 0.0	67.3 ± 0.9	3.5%	78.5 ± 0.2	79.7 ± 0.8	69.4 ± 0.6	4.6%
BGCN-A	77.7 ± 0.2	79.0 ± 0.2	70.1 ± 0.0	-	79.1 ± 0.2	80.0 ± 0.6	71.3 ± 0.3	-
BGCN-T	$\mathbf{78.0 \pm 0.2}$	78.7 ± 0.2	70.6 ± 0.1	-	79.4 ± 0.1	82.0 ± 0.1	71.9 ± 0.0	-
BGAT-A	77.6 ± 0.3	78.6 ± 1.2	71.0 ± 1.4	-	79.1 ± 0.4	82.9 ± 0.9	73.2 ± 0.7	-
BGAT-T	77.8 ± 0.2	$\mathbf{79.6 \pm 0.6}$	$\mathbf{71.4 \pm 1.3}$	-	$\mathbf{79.8 \pm 0.3}$	$\mathbf{84.2 \pm 0.4}$	$\mathbf{74.0 \pm 0.3}$	-

Table 1: Performance of the compared methods on the three datasets *w.r.t.* prediction accuracy (mean of 10 different runs). The performance of GCN (2-layer) and GAT (2-layer) are copied from their original papers. RI means the average relative improvement across datasets achieved by BGAT-T. We omit the models with more layers for the consideration of over-smoothing issue [Li *et al.*, 2018].

benchmark datasets of citation network—Pubmed, Cora and Citeseer [Sen *et al.*, 2008]. In these datasets, nodes and edges represent documents and citation relations between documents, respectively. Each node is represented by the bag-of-words features extracted from the content of the document. Each node has a label with one-hot encoding of the document category. We employ the same data split in previous works [Kipf and Welling, 2017; Yang *et al.*, 2016; Veličković *et al.*, 2018]. That is, 20 labeled nodes per class are used for training. 500 nodes and 1000 nodes are used as validation set and test set, respectively. Note that the train process can use all of the nodes' features. For this data split, we report the average test accuracy over ten different random initializations. To save space, we refer [Kipf and Welling, 2017] for the detailed statistics of the three datasets.

Compared Methods. We compare against the strong baselines mainly in two categories: *network embedding* and *GNN*. We select three widely used network embedding approaches: graph regularization-based network embedding (SemiEmb) [Weston *et al.*, 2012] and skip-gram-based graph embedding (DeepWalk [Perozzi *et al.*, 2014] and Planetoid [Yang *et al.*, 2016]). For GNNs, we select GCN [Kipf and Welling, 2017], GAT [Veličković *et al.*, 2018] and Graph Isomorphism Network (GIN) [Xu *et al.*, 2019b].

We devise two BGNNs which implement the AGG function as GCN and GAT, respectively. For each BGNN, we compare two variants with different scopes of the bilinear interactions: 1) BGCN-A and BGAT-A which consider all nodes within the k-hop neighbourhood, including the target node in the bilinear interaction. 2) BGCN-T and BGAT-T, which consider the interactions between the target node and the neighbor nodes within its k-hop neighbourhood.

Parameter Settings. We closely follow the GCN work [Kipf and Welling, 2017] to set the hyper-parameters of SemiEmb, DeepWalk, and Planetoid. We perform grid-search to select the optimal values for hyper-parameters of the remaining methods, including the dropout rate, the weight for l_2-norm (λ), the β trade-off the aggregated information from multi-hop nodes, and the α that balances the linear aggregator and bilinear aggregator. The dropout rates, λ, β and α are selected within $[0, 0.2, 0.4, 0.6]$, $[0, 1e\text{-}4, 5e\text{-}4, 1e\text{-}3]$, $[0, 0.1, 0.3, \cdots, 0.9, 1]$ and $[0, 0.1, 0.3, \cdots, 0.9, 1]$, respectively. All BGNN-based models are trained for 2,000 epochs with an early stopping strategy based on both convergence behavior and accuracy of the validation set.

4.1 Performance Comparison

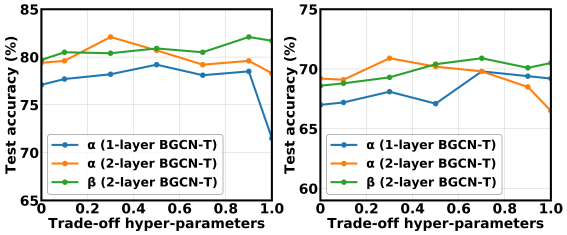
Table 1 shows the performance of the compared methods on the three datasets *w.r.t.* prediction accuracy on the data split exactly same as in [Kipf and Welling, 2017]. From the table, we have the following observations:

- In all cases, the proposed BGNN models achieves the best performance with average improvements over the baselines larger than 1.5%. The results validate the effectiveness of BGNN which is attributed to incorporating the pairwise interactions between the nodes in the local structure (*i.e.,* the ego network of the target node) when performing graph convolution.
- On average, BGAT (BGCN) outperforms vanilla GAT (GCN) by 1.5% (1.6%). These results further indicate the benefit of considering the interaction between neighbor nodes, which could augment the representation of a target node, facilitating its classification. Furthermore, the improvements of BGAT and BGCN in the 1-layer and 2-layer settings are close, which indicates that the interactions between both 1-hop neighbors and 2-hop neighbors are helpful for the representation of a target node.
- BGNN models, which have different scopes of the bilinear interactions, achieve different performance across datasets. In most cases, BGAT-T (BGCN-T) achieves performance better than BGAT-A (BGCN-A), signifying the importance of interactions with the target node.
- Among the baselines, GCN models perform better than embedding-based methods, indicating the effectiveness of graph convolution operation in learning node representations. GAT models perform better than GCN models. These results are consistent with findings in previous works [Kipf and Welling, 2017; Yang *et al.*, 2016; Veličković *et al.*, 2018].

As reported in [Kipf and Welling, 2017] (Table 2), the performance of GCN on random data splits is significantly worse than the fixed data split. As such, following [Wu *et al.*, 2019], we also test the methods on 10 random splits of the training set while keeping the validation and test sets unchanged. Table 2 shows the performance of BGCN-T and GCN over ran-

Split	Model	Pubmed	Cora	Citeseer
Random	GCN	77.0 ± 1.3	79.7 ± 1.2	70.8 ± 0.9
	BGCN-T	$\mathbf{77.9 \pm 1.1}$	$\mathbf{80.3 \pm 1.1}$	$\mathbf{71.6 \pm 1.1}$
Fixed	GCN	79.0	81.5	70.3
	BGCN-T	$\mathbf{79.4 \pm 0.1}$	$\mathbf{82.0 \pm 0.1}$	$\mathbf{71.9 \pm 0.0}$

Table 2: Test accuracy of 2-layer GCN and BGCN-T on the three datasets with random data splits.



(a) Cora (b) Citeseer

Figure 3: Impacts of trade-off hyper-parameters.

dom splits. To save space, we omit the results of BGCN-A and BGAT-based models which show similar trends. As can be seen, BGCN-T still outperforms GCN with high significant level ($< 5\%$), which further validates the effectiveness of the proposed model. However, the performance of both BGCN-T and GCN suffers from random data split as compared to the fixed data split. This result is consistent with previous work [Wu *et al.*, 2019] and reasonable since the hyper-parameters are tuned on the fixed data split.

4.2 Study of BGNN

Impacts of Bilinear Aggregator. As the BA is at the core of BGNN, we first investigate its impacts on the performance by varying the value of α. Note that larger α means more contributions from the BA; BGNN will downgrade to vanilla GNN with only the linear aggregator by setting $\alpha = 0$, while being fully dependent on the BA by setting $\alpha = 1$. Figures 3a and 3b show the performance of BGCN-T with 1-layer and 2-layer on Cora and Citeseer datasets, respectively. We omit the results of other BGCN-based and BGAT-based models and results on Pubmed for saving space, which show similar trends. We have the following observations: 1) Under the two settings (1-layer and 2-layer), the performance of BGCN-T varies in a range from 67.5 to 82.1. It suggests a careful tuning of α would make our models achieve desired performance. 2) BGCN-T outperforms vanilla GCN in most cases. It again verifies that the BA is capable of capturing the complex patterns of information propagation, which are hard to reveal by the linear aggregator individually. 3) Surprisingly, the performance of BGCN-T with $\alpha = 1$ is much worse than the performance when α is set to the optimal value. One possible reason is that the BA mainly serves as the complementary component to the linear aggregator, hardly working alone to achieve the comparable performance.

Impacts of Multi-Hop Neighbors. We also study the effects of β, in order to explore such trade-off between the aggregated information from different hops. Note that setting β as 0 and 1 denotes the individual modeling of one- and two-hop neighbors, respectively. As Figure 3 shows, we observe that involving the pairwise interactions from one- and two-hop neighbors simultaneously achieves better performance. It again verifies the effectiveness of stacking more BAs.

GCN	BGCN-T	Pubmed		Cora		Citeseer	
		Degree	Ratio	Degree	Ratio	Degree	Ratio
√	√	63.7	0.83	37.8	0.75	16.4	0.77
√	×	61.2	0.65	32.3	0.77	7.8	0.77
×	√	45.7	0.59	28.5	0.76	7.6	0.71

Table 3: Analysis of aggregators on the test set.

4.3 In-Depth Analysis of Aggregators

We perform in-depth analysis of different aggregators to clarify their working mechanism with respect to two node characteristics — 1) **Degree**, which denotes the average numbers of (one- and two-hop) neighbors surrounding the target node, and 2) **Ratio**, we first count the number of (one- and two-hop) neighbors which have the same label with the target node, and then divide this number by the number of all one- and two-hop neighbors. We summarize our statistical results in Table 3, wherein the symbol $\surd$ and $\times$ denote whether the target nodes are correctly classified or misclassified, respectively. That is, we categorize the testing nodes into three groups according to the correctness of predictions from GCN and BGCN-T. Jointly analyzing the three categories corresponding to the three rows in Table 3, we have the following findings: 1) Focusing on the third category with the least degree, BGCN-T consistently outperforms GCN, suggesting that the bilinear aggregator is able to distill useful information from sparser neighbors. 2) Comparing the third category to the second one, we observe that BGCN-T is able to endow the predictor node denoising ability. That is, BGCN-T can effectively aggregate information from the neighbors with consistent labels, filtering out the useless information from the irrelevant neighbors. 3) We also realize the limitations of BGCN-T from the second category — the bilinear interaction might need more label-consistent neighbors (*i.e.,* larger ratio), when more neighbors are involved (*i.e.,* larger degree).

5 Conclusion

In this paper, we proposed BGNN, a new graph neural network framework, which augments the expressiveness of vanilla GNN by considering the interactions between neighbor nodes. The neighbor node interactions are captured by a simple but carefully devised bilinear aggregator. The simpleness of the bilinear aggregator makes BGNN have the same model complexity as vanilla GNN *w.r.t.* the number of learnable parameters and analytical time complexity. Furthermore, the bilinear aggregator is proved to be permutation invariant which is an important property for GNN aggregators [Hamilton *et al.*, 2017; Xu *et al.*, 2019b]. We applied the proposed BGNN on the semi-supervised node classification task, achieving state-of-the-art performance on three benchmark datasets. In future, we plan to explore the following research directions: 1) encoding high-order interactions among multiple neighbors, 2) exploring the effectiveness of deeper BGNNs with more than two layers, and 3) developing AutoML technique [Feurer *et al.*, 2015] to adaptively learn the optimal α and β for each neighbor.

References

[Atwood and Towsley, 2016] James Atwood and Don Towsley. Diffusion-convolutional neural networks. In *NeurIPS*, pages 1993–2001, 2016.

[Beutel *et al.*, 2018] Alex Beutel, Paul Covington, Sagar Jain, Can Xu, Jia Li, Vince Gatto, and Ed H Chi. Latent cross: Making use of context in recurrent recommender systems. In *WSDM*, pages 46–54, 2018.

[Bronstein *et al.*, 2017] Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Mag*, 34(4):18–42, 2017.

[Bruna *et al.*, 2014] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. *ICLR*, 2014.

[Chami *et al.*, 2019] Ines Chami, Zhitao Ying, Christopher Ré, and Jure Leskovec. Hyperbolic graph convolutional neural networks. In *NeurIPS*, pages 4869–4880, 2019.

[Chen *et al.*, 2018] Jie Chen, Tengfei Ma, and Cao Xiao. Fastgcn: fast learning with graph convolutional networks via importance sampling. *ICLR*, 2018.

[Defferrard *et al.*, 2016] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *NeurIPS*, pages 3844–3852, 2016.

[Feng *et al.*, 2019] Fuli Feng, Xiangnan He, Xiang Wang, Cheng Luo, Yiqun Liu, and Tat-Seng Chua. Temporal relational ranking for stock prediction. *ACM Transactions on Information Systems (TOIS)*, 37(2):1–30, 2019.

[Feurer *et al.*, 2015] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. Efficient and robust automated machine learning. In *NeurIPS*, pages 2962–2970, 2015.

[Hamilton *et al.*, 2017] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NeurIPS*, pages 1024–1034, 2017.

[He and Chua, 2017] Xiangnan He and Tat-Seng Chua. Neural factorization machines for sparse predictive analytics. In *SIGIR*, pages 355–364, 2017.

[He *et al.*, 2016] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016.

[He *et al.*, 2020] Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, Yongdong Zhang, and Meng Wang. Lightgcn: Simplifying and powering graph convolution network for recommendation. In *SIGIR*, 2020.

[Kampffmeyer *et al.*, 2019] Michael Kampffmeyer, Yinbo Chen, Xiaodan Liang, Hao Wang, Yujia Zhang, and Eric P Xing. Rethinking knowledge graph propagation for zero-shot learning. In *CVPR*, pages 11487–11496, 2019.

[Kipf and Welling, 2017] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *ICLR*, 2017.

[Li *et al.*, 2018] Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *AAAI*, 2018.

[Liao *et al.*, 2019] Renjie Liao, Zhizhen Zhao, Raquel Urtasun, and Richard Zemel. Lanczosnet: Multi-scale deep graph convolutional networks. In *ICLR*, 2019.

[Park and Neville, 2019] Hogun Park and Jennifer Neville. Exploiting interaction links for node classification with deep graph neural networks. In *IJCAI*, pages 3223–3230, 2019.

[Perozzi *et al.*, 2014] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *KDD*, pages 701–710, 2014.

[Rendle, 2010] Steffen Rendle. Factorization machines. In *ICDM*, pages 995–1000, 2010.

[Sen *et al.*, 2008] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008.

[Veličković *et al.*, 2018] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *ICLR*, 2018.

[Veličković *et al.*, 2019] Petar Veličković, William Fedus, William L. Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. Deep graph infomax. In *ICLR*, 2019.

[Wang *et al.*, 2019] Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. Neural graph collaborative filtering. In *SIGIR*, pages 165–174, 2019.

[Weston *et al.*, 2012] Jason Weston, Frédéric Ratle, Hossein Mobahi, and Ronan Collobert. Deep learning via semi-supervised embedding. In *Neural Networks: Tricks of the Trade*, pages 639–655. Springer, 2012.

[Wu *et al.*, 2019] Felix Wu, Tianyi Zhang, Amauri Holanda de Souza Jr, Christopher Fifty, Tao Yu, and Kilian Q Weinberger. Simplifying graph convolutional networks. *ICML*, pages 6861–6871, 2019.

[Xinyi and Chen, 2019] Zhang Xinyi and Lihui Chen. Capsule graph neural network. In *ICLR*, 2019.

[Xu *et al.*, 2018] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. *ICML*, pages 8676–8685, 2018.

[Xu *et al.*, 2019a] Bingbing Xu, Huawei Shen, Qi Cao, Yunqi Qiu, and Xueqi Cheng. Graph wavelet neural network. In *ICLR*, 2019.

[Xu *et al.*, 2019b] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *ICLR*, 2019.

[Yang *et al.*, 2016] Zhilin Yang, William W Cohen, and Ruslan Salakhutdinov. Revisiting semi-supervised learning with graph embeddings. *ICML*, pages 86–94, 2016.

[Zhang *et al.*, 2018] Ziwei Zhang, Peng Cui, and Wenwu Zhu. Deep learning on graphs: A survey. *arXiv preprint arXiv:1812.04202*, 2018.