From Node Interaction to Hop Interaction: New Effective and Scalable Graph Learning Paradigm

Jie Chen[1] Zilong Li[1] Yin Zhu[1] Junping Zhang[1] Jian Pu[2*]
[1] Shanghai Key Lab of Intelligent Information Processing, School of Computer Science, Fudan University, Shanghai 200433, China
[2] Institute of Science and Technology for Brain-Inspired Intelligence, Fudan University, Shanghai 200433, China
{chenj19, yinzhu20, jpzhang, jianpu}@fudan.edu.cn, zilongli21@m.fudan.edu.cn

Abstract

Existing Graph Neural Networks (GNNs) follow the message-passing mechanism that conducts information interaction among nodes iteratively. While considerable progress has been made, such node interaction paradigms still have the following limitation. First, the scalability limitation precludes the broad application of GNNs in large-scale industrial settings since the node interaction among rapidly expanding neighbors incurs high computation and memory costs. Second, the over-smoothing problem restricts the discrimination ability of nodes, i.e., node representations of different classes will converge to indistinguishable after repeated node interactions. In this work, we propose a novel hop interaction paradigm to address these limitations simultaneously. The core idea is to convert the interaction target among nodes to pre-processed multi-hop features inside each node. We design a simple yet effective HopGNN framework that can easily utilize existing GNNs to achieve hop interaction. Furthermore, we propose a multi-task learning strategy with a self-supervised learning objective to enhance HopGNN. We conduct extensive experiments on 12 benchmark datasets in a wide range of domains, scales, and smoothness of graphs. Experimental results show that our methods achieve superior performance while maintaining high scalability and efficiency. The code is at https://github.com/JC-202/HopGNN.

1. Introduction

Graph Neural Networks (GNNs) have recently become very popular and have demonstrated great results in a wide range of graph applications, including social networks [38], point cloud analysis [37] and recommendation systems [21]. The core success of GNNs lies in the message-passing mechanism that iteratively conducts information interaction among nodes [8, 17, 18]. Each node in a graph convolution layer first aggregates information from local neighbors and combines them with non-linear transformation to update the self-representation [20, 27, 42]. After stacking K layers, nodes can capture long-range K-hop neighbor information and obtain representative representations for downstream tasks [29, 45]. However, despite the success of such popular node interaction paradigms, the number of neighbors for each node would grow exponentially with layers [2, 40], resulting in the well-known *scalability* and *over-smoothing* limitation of GNNs.

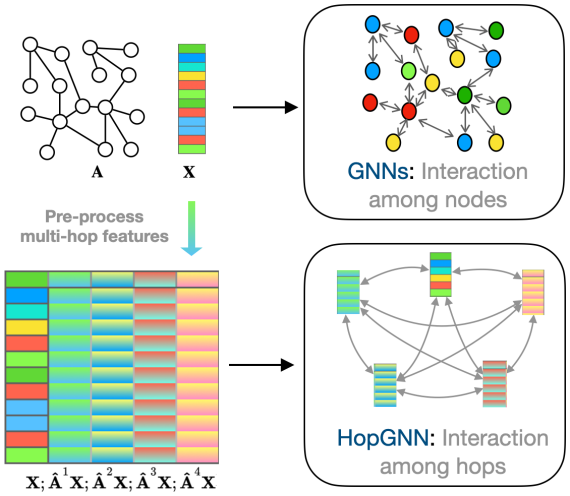

Figure 1. Comparison of node interaction and hop interaction. The hop interaction first pre-computes multi-hop features and then conducts non-linear interaction among different hops via GNNs, which enjoy high efficiency and effectiveness.

The *scalability* limitation precludes the broad application of GNNs in large-scale industrial settings since the node interaction among rapidly expanding neighbors incurs high computation and memory costs [15, 51]. Al-

though we can reduce the size of neighbors by sampling techniques [9, 20], it still executes node interaction iteratively during training, and the performance is highly sensitive to the sampling quality [15]. Recently, scalable GNNs that focus on simplifying or decoupling node interactions have emerged [16, 43, 52]. Such decoupled GNNs first precompute the linear aggregation of K-hop neighbors to generate node features and then utilize the MLP to each node without considering the graph structure during training and inference. However, despite high efficiency and scalability, such methods lead to suboptimal results due to the lack of nonlinear interactions among nodes.

Another limitation is *over-smoothing*, which restricts the discriminative ability of nodes, *i.e.*, node representations will converge to indistinguishable after repeated node interactions [5, 31]. On one hand, it causes performance degeneration when increasing layers of GNNs [27, 33]. On the other hand, in some heterophily graphs where connected nodes are usually from different classes, shallow GNNs are also surprisingly inferior to pure Multi-ayer Perceptrons (MLPs) [36, 53]. The reason is the interaction among massive local inter-class neighbors would blur class boundaries of nodes [6, 22, 53]. Recently, to carefully consider the neighbor influence and maintain the node discrimination, emerging advanced node interaction GNNs, such as deep GNNs with residual connections [11, 29] and heterophilic-graph-oriented GNNs with adaptive aggregation [4, 35, 39, 46], have achieved promising results. However, these advanced node interactions suffer high computational costs and fail to handle large-scale datasets.

These two limitations have typically been studied separately, as addressing one often necessitates compromising the other. However, *can we bridge the two worlds, enjoying the low-latency, node-interaction-free of decoupled GNNs and the high discrimination ability of advanced node interaction GNNs simultaneously?* We argue that it is possible to transform the node interaction into a new hop interaction paradigm without losing performance, but drastically reducing the computational cost. As shown in Figure 1, the core idea of hop interaction is to decouple the whole node interaction into two parts, the non-parameter hop feature preprocessing and non-linear interaction among hops. Inspired by the recommendation system, the non-linear interaction among different semantic features can enhance discrimination [19], *e.g.*, model the co-occurrence of career, sex and age of a user to identify its interest. By treating the precomputed L hop neighbors as L semantic features within each node, we can consider node classification as a feature interaction problem, *i.e.*, model the non-linear hop interaction to obtain discriminative node representations.

To this end, we design a simple yet effective HopGNN framework to address the above limitation simultaneously. It first pre-computes the multi-hop representation according to the graph structure. Then, without loss of generality, we can utilize GNNs over a multi-hop feature graph inside each node to achieve hop interaction flexibly and explicitly. Specifically, we implement an attention-based interaction layer and average pooling for the HopGNN to fuse multi-hop features and generate the final prediction. Furthermore, to show the generality and flexibility of our framework, we provide a multi-task learning strategy that combines the self-supervised objective to enhance performance.

Our contributions are summarized as follows:

1. *New perspective:* We propose a new graph learning paradigm going from node to hop interaction. It conducts non-linear interactions among pre-processed multi-hop neighbor features inside each node.

2. *General and flexible framework:* We design a simple yet effective HopGNN framework for hop interaction. Besides, the HopGNN is general and flexible to combine the self-supervised objective to easily enhance performance.

3. *State-of-the-art performance:* Experimental results show HopGNN achieves state-of-the-art performance on 12 graph datasets of diverse domains, sizes and smoothness while maintaining high scalability and efficiency.

2. Background and Related Works

2.1. Notation and Node Classification Problem

Consider a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, with N nodes and E edges. Let $\mathbf{A} \in \mathbb{R}^{N \times N}$ be the adjacency matrix, with $\mathbf{A}_{i,j} = 1$ if edge$(i, j) \in \mathcal{E}$, and 0 otherwise. $\mathbf{X} \in \mathbb{R}^{N \times d}$ and $\mathbf{Y} \in \mathbb{R}^{N \times c}$ represent the features and labels of nodes, repsectively. Given a set of labeled nodes $\mathcal{V}_{\mathcal{L}}$, the task of node classification is to predict the labels of the unlabeled nodes by exploiting the graph structure $\mathbf{A}$ and features $\mathbf{X}$.

Besides, the notion of homophily and heterophily corresponds to the smoothness of the signal $\mathbf{Y}$ on the graph $\mathcal{G}$. The edge homophily ratio of the graph is defined as $\mathcal{H}_{\text{edge}} = \frac{|\{(u,v):(u,v)\in\mathcal{E} \wedge y_u = y_v\}|}{|\mathcal{E}|}$, while $\mathcal{H}_{\text{edge}}$ tends to 1 means high homophily and low heterophily, and vice versa.

2.2. Graph Neural Networks

Standard node interaction GNNs. Each layer of most node interaction GNNs follows the message-passing mechanism [18] that is composed of two steps: (1) aggregate information from neighbors: $\mathbf{m}_i^l = \text{AGGREGATE}(\mathbf{h_j}^l, v_j \in \mathcal{N}_i)$; (2) update representation: $\mathbf{h_i}^{l+1} = \text{UPDATE}(\mathbf{h_i}^l, \mathbf{m}_i^l)$. To capture long-range information, standard node interaction GNNs alternately stack graph convolutions, linear layers and non-linear activation functions to generate representative representations of nodes [44]. Without loss of generality, the widely used GNNs follow the following form:

$$\mathbf{H}^L = \hat{\mathbf{A}}\, \sigma\left(\cdots \sigma\left(\hat{\mathbf{A}}\mathbf{X}\mathbf{W}\right)\cdots\right)\mathbf{W}^{L-1} \tag{1}$$

where $\hat{\mathbf{A}}$ is a normalized weighted matrix for feature propagation, $\mathbf{W}^l$ is a learnable weight matrix and σ is a non-linear activation function Relu. For example, the GCN [27] utilizes the symmetric normalized adjacent matrix as $\hat{\mathbf{A}}$, GraphSAGE [20] utilizes the random walk normalized version, and the GAT [42] applies the attention mechanism [41] to obtain a learnable weighted $\hat{\mathbf{A}}$.

Advanced node interaction GNNs. To avoid over-smoothing and heterophily problems that arise from simply stacking graph convolution layers [31, 36], most advanced node interaction GNNs adopt residual connections [30, 45] or adaptive aggregation strategies [7, 32, 36, 53] to extend standard GNNs. For instance, GCNII [11] is the SOTA deep GNN that combines GCN with initial connection and identity mapping. Geom-GCN [36] and WRGAT [39] transform the original graph by discovering the non-local semantic similarity neighbor. H2GCN [53] utilizes ego and neighbor separation and higher-order combination to improve the performance of GNNs under heterophily. GGCN [46] adopt signed messages from nodes' local neighbors and a degree correction mechanism for node-wise rescaling. FAGCN [4] and ACM-GCN [35] apply low-pass and high-pass filters in each graph convolution layer. Despite the high expressiveness of these advanced GNNs, they suffer from high computational costs and limited scalability to large-scale graphs.

Sampling-based GNNs. Sampling-based GNNs reduce memory consumption by sampling and minibatch training to approximate the full-batch GNNs. There are three categories of widely-used sampling strategies: 1) *Node-wise sampling*, GraphSAGE [20] randomly samples a fixed-size set of neighbors for each node in every layer, and VR-GCN [10] further analyzes the variance reduction for node sampling. 2) *Layer-wise sampling*, Fast-GCN [9] samples a fixed number of nodes at each layer, and ASGCN [24] proposes adaptive layer-wise sampling with better variance control. 3) *Graph-wise sampling*, ClusterGCN [13] first partitions the entire graph into clusters and then samples the nodes in the clusters, and GraphSAINT [49] directly samples a subgraph for mini-batch training. However, these sampling strategies still conduct node interactions that face high communication costs during training, and the model performance is highly sensitive to sampling quality [15].

Decoupled GNNs. Unlike standard GNNs, which sequentially stack non-linear graph convolution layers, decoupled GNNs simplify the graph convolution by decoupling the model into two steps: hop feature propagation and an MLP classifier for prediction. Depending on the propagation order, there are two typical ways to decouple these two operations: (1) *Pre-processing* that first pre-computes the feature propagation, *e.g.*, $\mathbf{X} = \sum_{i=0}^{L} \theta_i \hat{\mathbf{A}}^i \mathbf{X}$, and then applies the MLP classifier for each node representation $\mathbf{X}^L$ individually. SGC [43] simplifies GNNs into a linear model $\mathbf{Y} = \hat{\mathbf{A}}^L \mathbf{X} \mathbf{W}$, which achieves faster computation. However, it only considers the L hop features, and the linear layer limits its expressiveness. S2GC [52] extends the SGC by using the simple spectral graph convolution to average the propagated features in multi-hop. It also proposes utilizing MLP as a classifier. Different from S2GC, SIGN [16] uses the concatenation of multi-hop features, with an individual W for each hop transformation, to achieve better performance. (2) *Post-processing* that propagates multi-hop features after an MLP predictor: $\mathbf{Y} = \sum_{i=0}^{L} \theta_i \hat{\mathbf{A}}^i \, \mathrm{MLP}(\mathbf{X})$. To this end, APPNP [28] utilizes the approximate personal PageRank filter as a feature propagator, while GPRGNN [14] further utilizes a learnable generalized PageRank for the weights of theta to enhance performance. However, such post-processing still needs to propagate high-cost features during training, limiting its scalability compared to pre-processing.

Although decoupled GNNs efficiently propagate features, they usually suffer suboptimal results in heterophilic graphs due to the lack of non-linear node interactions. Our proposed approach builds on the efficiency of pre-computed hop features, as in decoupled GNNs. However, we introduce a novel paradigm that explicitly considers the non-linear interactions among hops, which is more expressive and discriminative without sophisticated node interactions.

3. Methodology

3.1. HopGNN

As illustrated in Figure 2, our proposed HopGNN framework is composed of four steps, namely, hop preprocessing, hop encoding, hop interaction, hop fusion and prediction. We introduce them one by one in detail.

Hop Pre-processing. Extracting information from long-range neighbors is crucial for nodes to achieve representative representation [1, 45]. Unlike the node interaction GNNs, which stack L layers to reach L-hop neighbors and suffer high computational costs with limited scalability [51], our proposed framework is built upon a hop-information pre-processing step. It pre-computes multi-hop node information from the node's original feature vectors and the l-hop feature can be simplified as:

$$\tilde{\mathbf{X}} = [\mathbf{X}_p; \mathbf{X}_p^1 ...; \mathbf{X}_p^L], \quad \mathbf{X}_p^l = \hat{\mathbf{A}}^l \mathbf{X}, \tag{2}$$

where $\hat{\mathbf{A}}$ is the normalized adjacent matrix, and $\tilde{\mathbf{X}}$ contains the multi-hop neighbor information. This pre-processing does not require any learnable parameters and only needs to be computed once in a CPU or distributed system for large-scale graphs. As a result, it can make models naturally support mini-batch training and easily scale to large datasets since it eliminates the computational complexity of node aggregation during training and inference.

Hop Encoding. To obtain sufficient expressive power, it is necessary to have at least one learnable linear trans-

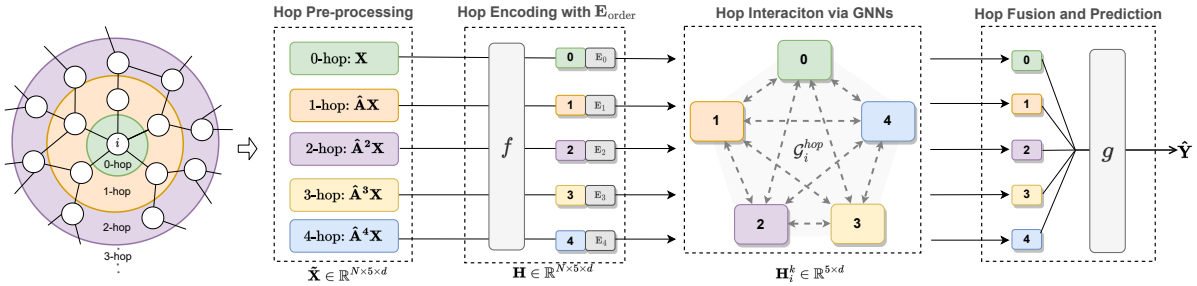

Figure 2. Overview of the proposed HopGNN framework with four steps, illustration with four hops. The core idea of HopGNN is to convert the interaction target of standard GNNs from nodes to pre-processed multi-hop features inside each node, which can enhance the nodes' discriminative power without node interaction.

formation to transform the input hop features into higher-level hop embedding. For parameter efficiency, we apply a shared parametered linear layer f to encode all hops. Moreover, to incorporate the semantic order information of the hops, we add a 1D learnable hop-order encoding vector $\mathbf{E}_{\text{order}} \in \mathbb{R}^{1\times L\times d}$ to each node hop embedding.

$$\mathbf{H} = \left[\mathrm{f}(\mathbf{X}_p); \mathrm{f}(\mathbf{X}_p^1); \cdots; \mathrm{f}(\mathbf{X}_p^L)\right] + \mathbf{E}_{\text{order}}. \tag{3}$$

The hop embedding $\mathbf{H} \in \mathbb{R}^{N\times L\times d}$ contains multi-hop neighbor information. The $\mathbf{E}_{\text{order}}$ can help the following order-insensitive hop interaction layer to capture hop-order information, which is important for the heterophily datasets, as further discussed in Section 4.3.

Hop Interaction. Inspired by [19], we argue that the underlying co-occurrence among different hop features inside each node contains the clue for its discrimination. Therefore, our goal is to model the non-linear interaction among hops to enhance discrimination without node interaction. Since message-passing is widely used in GNNs for non-linear interactions among nodes [18], to achieve hop interaction, we take advantage of well-designed GNNs by shifting the interaction target from nodes to multi-hop features inside each node. Specifically, for node v_i, we construct a fully-connected hop feature graph $\mathcal{G}_i^{\text{hop}}$, where each node v_l' in $\mathcal{G}_i^{\text{hop}}$ corresponds to an l-th hop feature of the original node v_i. Then, by leveraging any standard GNNs to model node interactions on the feature graph $\mathcal{G}_i^{\text{hop}}$, we can effectively capture multi-hop feature interactions of node v_i. To model high-order interactions among hops, we can stack K hop interaction layers with residual connections as follows:

$$\mathbf{H}_i^k = \mathrm{GNN}(\mathcal{G}_i^{\text{hop}}, \mathbf{H}_i^{k-1}) + \mathbf{H}_i^{k-1}, \quad k = 1\cdots K, \tag{4}$$

where $\mathbf{H}_i^k \in \mathbb{R}^{L\times d}$ is the L hop features of node v_i after the k-th hop interaction. In practice, we do not need to explicitly construct the hop feature graph $\mathcal{G}^{\text{hop}}$ for each node. Instead, without loss of generality, we implement the hop interaction GNN in Eq. (4) with a standard multi-head self-attention mechanism [41, 42] to capture the complex dependencies between hop features, *i.e.*, pairwise interactions between two hop features in different semantic subspaces. Specifically, the single head of self-attention on node v_i is calculated as follows:

$$\mathbf{A}_i^{\text{hop}} = \mathrm{softmax}\left(\frac{\mathbf{H}_i\mathbf{W}_{\mathbf{Q}}(\mathbf{H}_i\mathbf{W}_{\mathbf{K}})^\top}{\sqrt{d}}\right), \tag{5}$$

$$\mathrm{GNN}(\mathcal{G}_i^{\text{hop}}, \mathbf{H}_i) = \mathbf{A}_i^{\text{hop}}\mathbf{H}_i\mathbf{W}_{\mathbf{V}}, \tag{6}$$

where we omit the superscript k of layers for simplicity. Here, the $\mathbf{W}_{\mathbf{Q}}$, $\mathbf{W}_{\mathbf{K}}$, and $\mathbf{W}_{\mathbf{V}}$ are learnable projection matrices in each interaction layer. The resulting $\mathbf{A}_i^{\text{hop}}$ can be interpreted as a weighted feature interaction graph that captures the interaction strength among multi-hop features.

Using the multi-head attention for hop interaction can be regarded as applying the GAT [42] as the hop interaction GNN over the hop feature graph $\mathcal{G}_i^{\text{hop}}$ in Eq (4). We also compared other GNN architectures [20, 27] to model hop interactions in Section 4.3 and found that they can also achieve comparable performance, which validates the generality and flexibility of our hop interaction framework.

Hop Fusion and Prediction. After hop interaction, we apply a fusion and prediction function g for each node to generate the final output. We first apply a mean fusion to average all hop representations $\mathbf{H}^K \in \mathbb{R}^{N\times L\times d}$ into a single representation $\mathbf{Z} \in \mathbb{R}^{N\times d}$. Here, we have tested other fusion mechanisms, but we did not observe noticeable improvements. Then, we utilize another simple linear layer with softmax activation for the final prediction $\hat{\mathbf{Y}} \in \mathbb{R}^{N\times c}$.

$$\mathbf{Z} = \mathrm{fusion}(\mathbf{H}^K), \quad \hat{\mathbf{Y}} = \mathrm{softmax}(\mathrm{linear}(\mathbf{Z})). \tag{7}$$

The training objective of HopGNN is the cross-entropy loss between the ground truth labels $\mathbf{Y}$ and the prediction $\hat{\mathbf{Y}}$:

$$\mathcal{L}_{\text{ce}} = -\sum_{i\in\mathcal{V}_L}\sum_{j=1}^{c}\mathbf{Y}_{ij}\ln\hat{\mathbf{Y}}_{ij}. \tag{8}$$

Although training HopGNN with only $\mathcal{L}_{\text{ce}}$ achieves competitive results, to show the generality and flexibility of our framework, we will discuss how to combine HopGNN with self-supervised objectives [11, 48] in the following section.

Table 1. Time and memory complexities. We consider time complexity for the feature propagation and transformation in the network and memory complexity for storing the activations of node embeddings (we ignore model weights since they are negligible compared to the activations). L and K are the number of hops and the non-linear transformation, respectively, and d is the feature dimension (assumed to be fixed for all layers). N, $|E|$ and s are the numbers of nodes, edges, and sampling neighborhoods, respectively. b is the minibatch size.

Category		Method	Minibatch	Pre-Processing Time	Training Time	Training Memory
Standard		GCN/GCNII/...	×	-	$O(LEd + LNd^2)$	$O(LNd)$
Sampling	Node	GraphSAGE	✓	$O(s^L N)$	$O(s^L Nd^2)$	$O(bs^L d)$
	Layer	FastGCN/AS-GCN	✓	-	$O(sLNd^2)$	$O(bsLd)$
	Graph	Cluster-GCN	✓	$O(E)$	$O(LEd + LNd^2)$	$O(bLd)$
	Graph	GraphSAINT	✓	$O(sN)$	$O(LEd + LNd^2)$	$O(bLd)$
Decoupling	Pre	SGC/S2GC	✓	$O(LEd)$	$O(Nd^2)$	$O(bd)$
	Pre	SIGN	✓	$O(LEd)$	$O(KNd^2)$	$O(bLd)$
	Post	APPNP/GPRGNN	×	-	$O(KNd^2 + LEd)$	$O(LNd)$
Hop Interaction		HopGNN	✓	$O(LEd)$	$O(KNd^2 + KNL^2 d)$	$O(bLd)$

3.2. Self-Supervised Enhancement

In this subsection, we show that it can easily incorporate the self-supervised learning (SSL) objective to further enhance the performance of HopGNN. Recall that the key idea of HopGNN is to conduct interactions among multi-hops to enhance the discrimination of nodes. The desired interaction would help nodes to capture the task-relevant features and drop task-irrelevant features. To encourage such property, recent feature-level SSL studies aim to maximize feature invariance between two views and minimize the redundancy between dimensions [48, 50]. Formally, in Barlow Twins [48], given the cross-correlation matrix $\mathcal{C} \in \{-1, 1\}^{d \times d}$ between two views, the SSL objective is:

$$\mathcal{L}_{\text{ssl}} \triangleq \underbrace{\sum_i (1 - \mathcal{C}_{ii})^2}_{\text{invariance term}} + \alpha \underbrace{\sum_i \sum_{j \neq i} \mathcal{C}_{ij}^2}_{\text{redundancy reduction term}}, \quad (9)$$

where α is a scalar to control the decorrelation strength. Inspired by them, we further use multi-task learning to train the HopGNN, with the $\mathcal{L}_{\text{ce}}$ as the main task and $\mathcal{L}_{\text{ssl}}$ as the auxiliary task, and the overall training objective is:

$$\mathcal{L}_{\text{final}} = \mathcal{L}_{\text{ce}} + \lambda \mathcal{L}_{\text{ssl}}. \quad (10)$$

In our case, as shown in Figure 3, the additional SSL objective is applied after the hop interaction to enhance the discrimination. To generate two views of hop interaction features, distinguished from the previous works to adopt sophisticated augmentation [11, 34], we simply forward HopGNN twice with different dropout units as augmentations to obtain $\mathbf{H}^K$ and $\mathbf{H}'^K$. Since the training of our model is independent of the graph structure, complex graph-based data augmentation operations are not needed. Then, we not only calculate $\mathcal{L}_{\text{ce}}$ with fusion and prediction steps on both views but also flatten and normalize them to

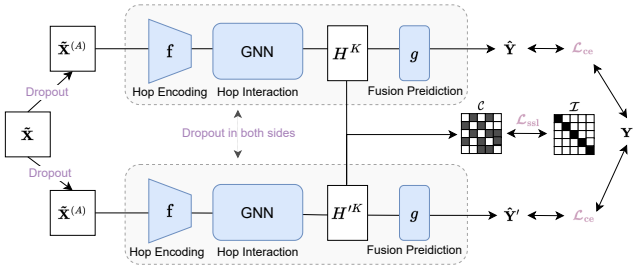

Figure 3. Multi-task learning with SSL for HopGNN. Illustration inspired by Barlow Twins [48].

calculate the cross-correlation matrix $\mathcal{C}$ for $\mathcal{L}_{\text{ssl}}$. Optimizing such the SSL objective can maximize the task-relevant mutual information and minimize the task-irrelevant information [50]. It would help the learned hop interaction representation $\mathbf{H}^K$ to extract minimal and sufficient information about downstream tasks from multi-hop neighbors.

Compared with other instance-level contrastive learning [12, 25], the memory costs of the feature-level $\mathcal{L}_{\text{ssl}}$ do not increase with the graph size since it focuses on the feature dimension, which is scalable and efficient. We also provide the result of HopGNN with contrastive learning in the Appendix.

3.3. Discussions

Connection to decoupled GNNs. From the perspective of hop interaction, the decoupled GNNs learn a fixed linear combination of multi-hop features for all nodes, which is equivalent to applying fixed hop attention coefficients with diagonal parts and remaining off-diagonal as 0. However, such fixed coefficients ignoring the pairwise hop interaction would cause sub-optimal results since each hop's contribution for different nodes may be different. In contrast, our proposed HopGNN utilizes the self-attention mechanism to

learn the representations of different-hop neighbors based on their semantic correlation for each node, which helps the model learn more informative node representations.

Complexity analysis. Table 1 provides a detailed asymptotic complexity comparison between HopGNN and other representative GNN methods. 1) The standard node interaction GNNs need full-batch training, and the time complexity contains the feature propagation part $O(LEd)$ over edges and feature transformation $O(LNd^2)$ over nodes. Moreover, for memory complexity, we need to store the activation of node embedding in each layer, which has $O(LNd)$. Note that we ignore the memory usage of model weights here since they are negligible compared to the activations [15]. The time and memory complexity of full-batch node interaction GNNs are highly correlated to the size of the graphs and result in the scalability problem. 2) Most sampling GNNs reduce the training time and memory cost via mini-batching with the corresponding sampling neighbor size s. 3) For pre-computed decoupled GNNs, thanks to the pre-processing of feature propagation, the training complexity is the same as the traditional mini-batch training, *e.g.*, MLPs with feature transformation, which is usually smaller than the sampling methods. However, the post-computed decoupled GNNs still require feature propagation during training, leading to a cost similar to that of full-batch GNNs. 4) The computational cost of HopGNN is similar to that of the pre-computed decoupled GNN, which is also scalable to large-scale graphs. Compared with SIGN, the HopGNN explicitly conducts non-linear interactions among L hops, which is more expressive and requires $O(KNL^2d)$ additional time complexity of K interactions.

4. Experiments

4.1. Setup

Datasets. We comprehensively evaluate the performance of HopGNN on 12 benchmark datasets. These datasets vary in domain, size and smoothness, including three standard homophily citation datasets [27], six well-known heterophily datasets [36], two large-scale inductive datasets [49] and a large-scale transductive OGB products dataset [23]. For the homophily and heterophily benchmark datasets, we use the same 10 public fixed training/validation/test splits as provided in [36,53]. For large-scale datasets, we use their public splits in [23,49]. The statistics of these datasets are summarized in Tables 2 and 3. Details on these datasets can be found in Appendix.

Baselines. We compare HopGNN with various baselines, including (1) MLP; (2) standard node-interaction GNN methods: GCN [27], GAT [42], GraphSAGE [20] and GCNII [11]; (3) heterophilic GNNs with adaptive node interaction: H2GCN [53], WRGAT [39], ACM-GCN [35], GGCN [46] (4) sampling GNNs: FastGCN [9], AS-GCN [24], ClusterGNN [13], GraphSAINT [49]; and (5) decoupled GNNs: S2GC [52], SIGN [16], APPNP [28], GPRGNN [14]. We report results from previous works with the same experimental setup if available. If the results are not previously reported and codes are provided, we implement them based on the official codes and conduct a hyper-parameter search. We provide more baseline results in the Appendix due to space limits.

Implementation Details. Following the standard setting [4,23], we set the hidden dimension of HopGNN as 128 for the nine small-scale datasets and 256 for the three large-scale datasets. Although tuning the hops and layers usually leads to better results, for simplicity, we fix the number of hops as 6 and the interaction layer as 2 of HopGNNs for all datasets in Table 2 and 3. We use Adam [26] for optimization and LayerNorm [3] of each layer and tune the other hyper-parameters. Details can be found in the Appendix.

4.2. Overall Performance

Results on Homophily and Heterophily. From Table 2, we make the following observations: 1) Standard node interactions are sometimes inferior to MLP, such as Actor and Cornell, indicating that simply stacking node interaction may fail in the heterophily datasets. However, the heterophilic GNNs achieve better performance in general, *e.g.*, their average performance over nine datasets is all larger than 70. The reason is that such advanced node interaction can adaptively consider the influence of neighbors in each hop. 2) Compared with heterophilic GNNs, decoupled GNNs achieve sub-optimal results, *i.e.*, their overall performance is less than 70, due to missing the non-linear interaction among nodes. 3) HopGNN achieves significantly better performance than decouple GNN and is comparable with the SOTA heterophilic GNNs. Such results show that even without complex node interactions, the non-linear interaction among hops can enhance node discrimination in both homophily and heterophily. This validates the effectiveness and generality of the hop interaction paradigm. (4) Moreover, combining the SSL, the HopGNN+ consistently outperforms HopGNN and achieves the best overall performance, validating the effectiveness of the multi-task learning strategy and the compatibility of HopGNN.

Results on Large-scale Datasets. We compare the HopGNN with scalable GNNs, including pre-processing decoupled GNNs and sampling-based GNNs in Table 3, and find that: 1) The decoupled SIGN performs better in the inductive setting, and the subgraph-based GraphSAINT outperforms other baselines in the large-scale transductive product dataset. 2) HopGNN consistently outperforms all the baseline methods in these large-scale datasets. The results in the inductive setting of Flickr and Reddit substantiate the ability of HopGNN to generalize to unseen nodes. Moreover, the HopGNN+ combined feature-level SSL ob-

Table 2. Mean test accuracy ± stdev on 6 heterophily and 3 homophily real-world datasets over 10 public splits (48%/32%/20% of nodes for training/validation/test). The best performance is highlighted. ‡ denotes the results obtained from previous works [46, 53]

	Texas	Wisconsin	Actor	Squirrel	Chameleon	Cornell	Citeseer	Pubmed	Cora	Avg
$\mathcal{H}_{\text{edge}}$	0.11	0.21	0.22	0.22	0.23	0.3	0.74	0.8	0.81	-
#Nodes	183	251	7,600	5,201	2,277	183	3,327	19,717	2,708	-
#Edges	295	466	26,752	198,493	31,421	280	4,676	44,327	5,278	-
#Classes	5	5	5	5	5	5	7	3	6	-
MLP‡	81.89±4.78	85.29±3.61	35.76±0.98	29.68±1.81	46.36±2.52	81.08±6.37	72.41±2.18	86.65±0.35	74.75±2.22	65.99
GCN‡	55.14±5.16	51.76±3.06	27.32±1.10	53.43±2.01	64.82±2.24	60.54±5.30	76.50±1.36	88.42±0.50	86.90±1.04	62.76
GAT‡	52.14±5.16	49.41±4.09	27.44±0.89	40.72±1.55	60.26±2.50	61.89±5.05	76.55±1.23	86.33±0.48	87.30±1.10	60.23
GraphSAGE‡	82.43±6.14	81.18±5.56	34.23±0.99	41.61±0.74	58.73±1.68	75.95±5.01	76.04±1.30	88.45±0.50	86.90±1.04	69.50
GCNII‡	77.57±3.83	80.39±3.40	37.44±1.30	38.47±1.58	63.86±3.04	77.86±3.79	77.33±1.48	90.15±0.43	88.37±1.25	70.16
H2GCN-1‡	84.86±6.77	86.67±4.69	35.86±1.03	36.42±1.89	57.11±1.58	82.16±4.80	77.07±1.64	89.40±0.34	86.92±1.37	70.72
H2GCN-2‡	82.16±5.28	85.88±4.22	35.62±1.30	37.90±2.02	59.39±1.98	82.16±6.00	76.88±1.77	89.59±0.33	87.81±1.35	70.87
ACM-GCN‡	87.84±4.40	88.43±3.22	36.28±1.09	54.40±1.88	66.93±1.85	85.14±6.07	77.32±1.70	90.00±0.52	87.91±0.95	74.92
WRGAT‡	83.62±5.50	86.98±3.78	36.53±0.77	48.85±0.78	65.24±0.87	81.62±3.90	76.81±1.89	88.52±0.92	87.95±1.18	72.90
GGCN‡	84.86±4.55	86.86±3.29	37.54±1.56	55.17±1.58	71.14±1.84	85.68±6.63	77.14±1.45	89.15±0.37	87.95±1.05	75.05
S2GC	68.65±8.05	71.57±9.01	34.17±0.92	41.63±0.98	58.55±5.15	75.25±7.82	76.08±0.45	88.31±0.38	87.73±2.90	66.88
SIGN	75.14±7.94	80.59±3.75	36.14±1.01	40.16±2.12	60.48±2.10	78.11±4.67	76.53±1.76	89.58±0.45	86.72±1.37	69.27
APPNP	78.37±6.01	81.42±4.34	34.64±1.51	33.51±2.02	47.50±1.76	77.02±7.01	77.06±1.73	87.94±0.56	87.71±1.34	67.24
GPRGNN	82.12±7.72	81.16±3.17	33.29±1.39	43.29±1.66	61.82±2.39	81.08±6.59	75.56±1.62	86.85±0.46	86.98±1.33	70.15
HopGNN	81.35±4.31	84.96±4.11	36.66±1.39	60.95±1.65	70.13±1.39	83.70±6.52	76.16±1.53	89.98±0.39	87.12±1.35	74.56
HopGNN+	82.97±5.12	85.69±5.43	37.09±0.97	64.23±1.33	71.21±1.45	84.05±4.48	76.69±1.56	90.28±0.42	87.57±1.33	75.53

Table 3. Comparison over a large-scale dataset. ‡ denotes the results obtained from previous works [47].

Type	Flickr Inductive	Reddit Inductive	Products Transductive
#Nodes	89,250	232,965	2,449,029
#Edges	899,756	11,606,919	61,859,140
#Classes	7	41	47
GCN‡	49.2 ± 0.3	93.3 ± 0	75.64 ± 0.21
SGC‡	50.2 ± 0.1	94.9 ± 0	74.87 ± 0.25
SIGN ‡	51.4 ± 0.1	96.8 ± 0	77.60 ± 0.13
S2GC	50.48 ± 0.07	94.04 ± 0.03	76.84 ± 0.20
GraphSAGE‡	50.1 ± 1.3	95.3 ± 0.1	78.29 ± 0.16
FastGCN ‡	50.4 ± 0.1	92.4 ± 0.1	73.46 ± 0.20
AS-GCN‡	50.4 ± 0.2	96.4 ± 0.1	-
ClusterGCN‡	48.1 ± 0.5	95.4 ± 0.1	78.97 ± 0.33
GraphSAINT‡	51.1 ± 0.1	96.6 ± 0.1	79.08 ± 0.24
HopGNN	52.49 ± 0.15	96.92 ± 0.05	79.96 ± 0.11
HopGNN+	52.68 ± 0.16	96.98 ± 0.04	80.08 ± 0.08

Table 4. Ablation studies on order embedding, fusion and interaction types for heterophily and homophily graphs.

		Heterophily		Homophily	
HopGNN		69.63		84.42	
w/o $\mathbf{E}_{\text{order}}$		60.93	↓8.70	83.96	↓0.46
Fusion	Attention	69.81	↑0.18	84.60	↑0.18
	Max	69.50	↓0.13	84.39	↓0.03
Interaction	None	65.59	↓4.04	84.35	↓0.07
	MLP	67.21	↓2.42	84.41	↓0.01
	GCN	68.18	↓1.45	84.38	↓0.04
	SAGE	69.11	↓0.52	84.45	↑0.03

jective still works well in large-scale scenarios.

4.3. Ablation Study

In this part, we study the role of the hop-order embedding $\mathbf{E}_{\text{order}}$, hop interaction and hop fusion type to validate the effectiveness of each component and the generality of the whole framework. Due to space limits, we report the average test accuracy of these variants across three homophilic and six heterophilic datasets, as shown in Table 4.

Hop-order embedding. Without hop-order embedding, the performance drops slightly in homophily (0.46) but dramatically in heterophily (8.7). This indicates that hop-order information is more crucial for order-insensitive hop interactions in challenging heterophilic scenarios. The reason may be that the contribution varies in different order hop of neighbors, which is also discussed in H2GNN [53] that high-order neighbors are expected to be homophily dominant and useful for heterophilic datasets.

Hop fusion. We also test the effects of max- and attention-based fusion mechanisms in HopGNN and observe their similar performance under both heterophily and homophily. Although the attention-based fusion achieves slightly higher results, we still choose the mean fusion as the default due to its simplicity and efficiency.

Hop interaction. For the hop interaction layer, we test the variant of no interaction, interaction with GCN and SAGE, and interaction with MLP. We make the following observations: 1) They perform closely under homophily since

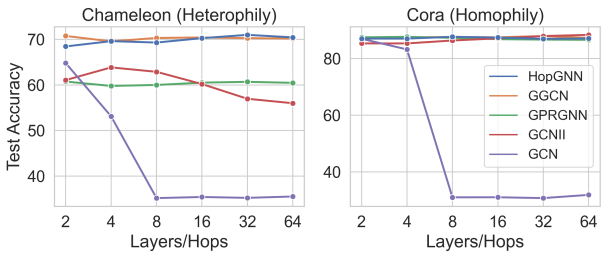

Figure 4. Classification accuracy with different layers/hops.

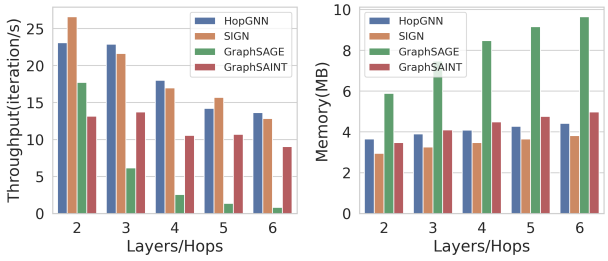

Figure 5. The comparison of *throughput* (left) and *memory usage* (right) in the Products dataset, both y-axis are in the log scale.

the nodes' neighbors share similar information, resulting in limited gain from interactions. 2) Removing the interaction incurs a performance reduction of 4.04 in heterophily datasets, and the interaction based on MLP also results in a 2.3 performance drop. These results are consistent with the limited performance of decoupled GNNs using linear combination or concatenation among hops. Such performance degeneration further validates the importance of hop-level interaction since it can investigate discriminative clues of nodes for the classification in heterophily. 3) For the interaction of GNN variants, both standard GCN and SAGE achieve competitive results, demonstrating that our framework is stable with different GNN-based hop interaction mechanisms. We choose the multi-head attention mechanism as the default due to its generalizability.

4.4. In-depth Analysis

Over-smoothing. To test the robustness of the models to over-smoothing, we compared the HopGNN with the classical GCN and different kinds of SOTA, including GRPGNN from decoupled GNNs, GCNII from deep GNNs, and GCNN from heterophilic GNNs. From Figure 4, we have the following observations: 1) The performance of the GCN drops rapidly as the number of layers increases. However, all the other models do not suffer over-smoothing in the homophilic datasets. 2) Both GCNII and GPRGNN significantly underperform the GGCN and HopGNN under all layers in Chameleon. This means that although decoupling and residual connection can solve the over-smoothing to some extent, they are insufficient for heterophily. 3) The GGCN conducts adaptive node interactions in each layer by carefully considering each hop neighbor's information, which is expressive but limits its scalability. Notability, without node interaction, the HopGNN still achieves robustness under different layers in both homophily and heterophily, validating the superiority of the hop interaction paradigm.

Efficiency. In Figure 5, we use the evaluation methodology from [15] to fairly compare the throughputs and actual memory usage of various representative methods. Specifically, we compare the HopGNN with SAGE variants under the same settings across layers during the training procedure on the largest Products dataset. We observe that: 1) The basic GraphSAGE is significantly slower and costs a huge amount of memory due to the neighbor explosion. Subgraph sampling makes GraphSAINT significantly reduce time and memory costs. 2) The decoupled SIGN achieves faster and smaller memory cost than GraphSAINT, but with sub-optimal results as shown in Table 3. 3) HopGNN costs slightly more memory than SIGN due to the additional hop interaction phase, but HopGNN achieves the best performance and is faster than GraphSAINT, which indicates a better trade-off between performance and scalability. The implementation details and training efficiency comparison can be found in Appendix.

5. Conclusion

We have presented a novel hop interaction paradigm, a practical solution to address the scalability and over-smoothing problem of GNNs simultaneously. It shifts the interaction target of GNNs from nodes to hops inside each node. Specifically, we design a simple yet effective HopGNN framework. It first pre-computes non-parameter aggregation of multi-hop features to reduce the computational cost during training and inference. Then, it conducts non-linear interactions among the multi-hop features of each node via GNNs to enhance their discriminative abilities. We also develop a multi-task learning strategy with the self-supervised objective to enhance the performance of the downstream task. Experiments on 12 benchmark datasets show that HopGNN achieves state-of-the-art performance while maintaining high scalability and efficiency. One future work is to investigate advanced interaction mechanisms among hops to enhance the performance of HopGNNs. It is also interesting to apply the HopGNN to other downstream tasks, such as graph classification and link prediction.

Acknowledgement This work is supported by the Shanghai Municipal Science and Technology Major Project (No. 2018SHZDZX01), National Natural Science Foundation of China (No. 62176059), and ZJLab, Shanghai Center for Brain Science and Brain-Inspired Technology.

References

[1] Sami Abu-El-Haija, Bryan Perozzi, Amol Kapoor, Nazanin Alipourfard, Kristina Lerman, Hrayr Harutyunyan, Greg Ver Steeg, and Aram Galstyan. Mixhop: Higher-order graph convolutional architectures via sparsified neighborhood mixing. In *International Conference on Machine Learning*, pages 21–29. PMLR, 2019. 3

[2] Uri Alon and Eran Yahav. On the bottleneck of graph neural networks and its practical implications. In *International Conference on Learning Representations*, 2020. 1

[3] Jimmy Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. *preprint arXiv:1607.06450*, 2016. 6

[4] Deyu Bo, Xiao Wang, Chuan Shi, and Huawei Shen. Beyond low-frequency information in graph convolutional networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 3950–3957, 2021. 2, 3, 6

[5] Deli Chen, Yankai Lin, Wei Li, Peng Li, Jie Zhou, and Xu Sun. Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 3438–3445, 2020. 2

[6] Jie Chen, Shouzhen Chen, Mingyuan Bai, Jian Pu, Junping Zhang, and Junbin Gao. Graph decoupling attention markov networks for semisupervised graph node classification. *IEEE Transactions on Neural Networks and Learning Systems*, 2022. 2

[7] Jie Chen, Shouzhen Chen, Zengfeng Huang, Junping Zhang, and Jian Pu. Exploiting neighbor effect: Conv-agnostic gnns framework for graphs with heterophily. *preprint arXiv:2203.11200*, 2022. 3

[8] Jie Chen, Weiqi Liu, and Jian Pu. Memory-based message passing: Decoupling the message for propagation from discrimination. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4033–4037. IEEE, 2022. 1

[9] Jie Chen, Tengfei Ma, and Cao Xiao. Fastgcn: Fast learning with graph convolutional networks via importance sampling. In *6th International Conference on Learning Representations, ICLR 2018*. OpenReview.net, 2018. 2, 3, 6

[10] Jianfei Chen, Jun Zhu, and Le Song. Stochastic training of graph convolutional networks with variance reduction. In *International Conference on Machine Learning*, pages 942–950. PMLR, 2018. 3

[11] Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. Simple and deep graph convolutional networks. In *International Conference on Machine Learning*, pages 1725–1735. PMLR, 2020. 2, 3, 4, 5, 6

[12] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International Conference on Machine Learning*, pages 1597–1607. PMLR, 2020. 5

[13] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 257–266, 2019. 3, 6

[14] Eli Chien, Jianhao Peng, Pan Li, and Olgica Milenkovic. Adaptive universal generalized pagerank graph neural network. In *International Conference on Learning Representations*, 2021. 3, 6

[15] Keyu Duan, Zirui Liu, Peihao Wang, Wenqing Zheng, Kaixiong Zhou, Tianlong Chen, Xia Hu, and Zhangyang Wang. A comprehensive study on large-scale graph training: Benchmarking and rethinking. In *Advances in Neural Information Processing Systems*, 2022. 1, 2, 3, 6, 8

[16] Fabrizio Frasca, Emanuele Rossi, Davide Eynard, Ben Chamberlain, Michael Bronstein, and Federico Monti. Sign: Scalable inception graph neural networks. *preprint arXiv:2004.11198*, 2020. 2, 3, 6

[17] Johannes Gasteiger, Janek Groß, and Stephan Günnemann. Directional message passing for molecular graphs. In *International Conference on Learning Representations*, 2020. 1

[18] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International Conference on Machine Learning*, pages 1263–1272. JMLR. org, 2017. 1, 2, 4

[19] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. Deepfm: A factorization-machine based neural network for ctr prediction. In *IJCAI*, 2017. 2, 4

[20] William L Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pages 1025–1035, 2017. 1, 2, 3, 4, 6

[21] Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, Yongdong Zhang, and Meng Wang. Lightgcn: Simplifying and powering graph convolution network for recommendation. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*, pages 639–648, 2020. 1

[22] Yifan Hou, Jian Zhang, James Cheng, Kaili Ma, Richard T. B. Ma, Hongzhi Chen, and Ming-Chang Yang. Measuring and improving the use of graph information in graph neural networks. In *International Conference on Learning Representations*, 2020. 2

[23] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. 6

[24] Wenbing Huang, Tong Zhang, Yu Rong, and Junzhou Huang. Adaptive sampling towards fast graph representation learning. *Advances in Neural Information Processing Systems*, 31, 2018. 3, 6

[25] Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan. Supervised contrastive learning. *Advances in Neural Information Processing Systems*, 33:18661–18673, 2020. 5

[26] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *preprint arXiv:1412.6980*, 2014. 6

[27] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017. 1, 2, 3, 4, 6

[28] Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. Predict then propagate: Graph neural networks meet personalized pagerank. In *International Conference on Learning Representations*, 2019. 3, 6

[29] Guohao Li, Matthias Müller, Guocheng Qian, Itzel Carolina Delgadillo Perez, Abdulellah Abualshour, Ali Kassem Thabet, and Bernard Ghanem. DeepGCNs: Making GCNs go as deep as CNNs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021. 1, 2

[30] Guohao Li, Matthias Müller, Ali K. Thabet, and Bernard Ghanem. Deepgcns: Can gcns go as deep as cnns? In *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*, pages 9266–9275. IEEE, 2019. 3

[31] Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3538–3545, 2018. 2, 3

[32] Xiang Li, Renyu Zhu, Yao Cheng, Caihua Shan, Siqiang Luo, Dongsheng Li, and Weining Qian. Finding global homophily in graph neural networks when meeting heterophily. In *International Conference on Machine Learning*, pages 13242–13256. PMLR, 2022. 3

[33] Meng Liu, Hongyang Gao, and Shuiwang Ji. Towards deeper graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 338–348, 2020. 2

[34] Yixin Liu, Ming Jin, Shirui Pan, Chuan Zhou, Yu Zheng, Feng Xia, and Philip Yu. Graph self-supervised learning: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 2022. 5

[35] Sitao Luan, Chenqing Hua, Qincheng Lu, Jiaqi Zhu, Mingde Zhao, Shuyuan Zhang, Xiao-Wen Chang, and Doina Precup. Revisiting heterophily for graph neural networks. In *Advances in Neural Information Processing Systems*, 2022. 2, 3, 6

[36] Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. Geom-gcn: Geometric graph convolutional networks. In *International Conference on Learning Representations*, 2020. 2, 3, 6

[37] Weijing Shi and Raj Rajkumar. Point-gnn: Graph neural network for 3d object detection in a point cloud. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1711–1719, 2020. 1

[38] Jianhua Sun, Qinhong Jiang, and Cewu Lu. Recursive social behavior graph for trajectory prediction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020. 1

[39] Susheel Suresh, Vinith Budde, Jennifer Neville, Pan Li, and Jianzhu Ma. Breaking the limit of graph neural networks by improving the assortativity of graphs with local mixing patterns. In *KDD*, pages 1541–1551, 2021. 2, 3, 6

[40] Jake Topping, Francesco Di Giovanni, Benjamin Paul Chamberlain, Xiaowen Dong, and Michael M. Bronstein. Understanding over-squashing and bottlenecks on graphs via curvature. In *International Conference on Learning Representations*, 2022. 1

[41] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017. 3, 4

[42] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018. 1, 3, 4, 6

[43] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. Simplifying graph convolutional networks. In *International Conference on Machine Learning*, pages 6861–6871. PMLR, 2019. 2, 3

[44] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4–24, 2021. 2

[45] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. In *International Conference on Machine Learning*, pages 5453–5462. PMLR, 2018. 1, 3

[46] Yujun Yan, Milad Hashemi, Kevin Swersky, Yaoqing Yang, and Danai Koutra. Two sides of the same coin: Heterophily and oversmoothing in graph convolutional neural networks. In *2022 IEEE International Conference on Data Mining (ICDM)*, pages 1287–1292. IEEE, 2022. 2, 3, 6, 7

[47] Jiaxuan You, Zhitao Ying, and Jure Leskovec. Design space for graph neural networks. *Advances in Neural Information Processing Systems*, 33:17009–17021, 2020. 7

[48] Jure Zbontar, Li Jing, Ishan Misra, Yann LeCun, and Stéphane Deny. Barlow twins: Self-supervised learning via redundancy reduction. In *International Conference on Machine Learning*, pages 12310–12320. PMLR, 2021. 4, 5

[49] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor K. Prasanna. Graphsaint: Graph sampling based inductive learning method. In *8th International Conference on Learning Representations, ICLR*, 2020. 3, 6

[50] Hengrui Zhang, Qitian Wu, Junchi Yan, David Wipf, and Philip S Yu. From canonical correlation analysis to self-supervised graph neural networks. *Advances in Neural Information Processing Systems*, 34:76–89, 2021. 5

[51] Shichang Zhang, Yozen Liu, Yizhou Sun, and Neil Shah. Graph-less neural networks: Teaching old mlps new tricks via distillation. In *International Conference on Learning Representations*, 2022. 1, 3

[52] Hao Zhu and Piotr Koniusz. Simple spectral graph convolution. In *International Conference on Learning Representations*, 2020. 2, 3, 6

[53] Jiong Zhu, Yujun Yan, Lingxiao Zhao, Mark Heimann, Leman Akoglu, and Danai Koutra. Beyond homophily in graph neural networks: Current limitations and effective designs. In *Advances in Neural Information Processing Systems*, volume 33, 2020. 2, 3, 6, 7