Can Graph Neural Networks Go Deeper Without Over-Smoothing? Yes, With a Randomized Path Exploration!

Kushal Bose and Swagatam Das, *Senior Member, IEEE*

***Abstract*—Graph Neural Networks (GNNs) have emerged as one of the most powerful approaches for learning on graph-structured data, even though they are mostly restricted to being shallow in nature. This is because node features tend to become indistinguishable when multiple layers are stacked together. This phenomenon is known as over-smoothing. This paper identifies two core properties of the aggregation approaches that may act as primary causes for over-smoothing. These properties are namely recursiveness and aggregation from higher to lower-order neighborhoods. Thus, we attempt to address the over-smoothing issue by proposing a novel aggregation strategy that is orthogonal to the other existing approaches. In essence, the proposed aggregation strategy combines features from lower to higher-order neighborhoods in a non-recursive way by employing a randomized path exploration approach. The efficacy of our aggregation method is verified through an extensive comparative study on the benchmark datasets w.r.t. the state-of-the-art techniques on semi-supervised and fully-supervised learning tasks.**

***Index Terms*—Aggregation, convolution, graph neural networks, message passing, over-smoothing.**

I. Introduction

AMIDST the rise of Geometric Deep Learning [1], [2], it is no wonder that the graph structure draws adequate attention for analyzing data beyond the conventional Euclidean domain. The milestone work on Graph Neural Networks (GNNs) [3] is crowned as one of the go-to tools to facilitate learning on graph-structured data. Consequently, application of deep learning techniques on graphs significantly enhances the performances of various downstream learning tasks such as node classification [4], [5], link prediction [6], [7], graph classification [8], [9], [10], and biological networks [11], [12].

However, the commendable performance of GNNs may be considered restrictive to their full potential, given the shallower architecture. Even though models like GCN [5], GAT [13], and GraphSage [14] exhibit impressive performance by using a limited number of hidden layers, they are still not capable of learning from the higher-order neighborhoods. One way to alleviate these restrictions is to aggregate features from the multi-hop neighbors using a deeper architecture.

Unfortunately, stacking multiple layers similar to Convolutional Neural Networks (CNNs) such as ResNet [15] in GNN is not so straightforward. This is because the increased number of successive features aggregation in a deeper architecture may make the combined node features almost indistinguishable. This phenomenon is known as *Over-smoothing* [16], which has been shown to degrade the performance of node classification tasks as well as the expressive power of the GNN models [17].

The features of the nodes belonging to a connected component become indistinguishable due to the effect of over-smoothing. The primary reason behind the occurrence of over-smoothing is the overlapping sets of neighbors of the concerned nodes. The overlap among the sets of neighbors gradually increases when the size of the neighborhood grows. In fact, the overlapping set of neighbors is not only responsible for the over-smoothing, but the aggregation mode also plays a critical role in the feature over-smoothing. The well-known architectures like Graph Convolutional Networks (GCN) [5], Graph Attention Networks (GAT) [13], GraphSage [14], APPNP [18], Graph Diffusion Convolution (GDC) [19] etc. aggregates node features from higher to lower order neighborhoods in a recursive manner as depicted in the Fig. 2. The redundancy involved in the recursive higher to lower order computation may lead to repeated consideration of the similar neighboring nodes during the feature calculation. This redundancy acts as a primary reason for the over-smoothing. Despite having the problem of over-smoothing, deep models offer a significant advantage over shallow models as the formers can learn from the multi-hop neighborhood. Therefore, the drawbacks mentioned earlier and the necessity of deep models motivate us to propose a novel aggregation technique that prevents over-smoothing.

Numerous attempts like DeepGCN [20], JKNet [21], PairNorm [22], DGN [23], DropEdge [24], AdaEdge [25] etc. were made to mitigate the issue of over-smoothing, but the aforesaid approaches are either different variants of GCN, GAT, or GraphSage or the combinations of the existing architectures where underlying aggregation strategy remains unaltered. The question of tackling over-smoothing by improving the existing neighborhood aggregation techniques still needs to be answered.

Manuscript received 18 May 2022; revised 12 January 2023; accepted 13 February 2023. This paper was produced by the IEEE Publication Technology Group, Piscataway, NJ, USA. *(Corresponding author: Swagatam Das.)*

The authors are with the Electronics and Communication Sciences Unit, Indian Statistical Institute, Kolkata 700108, India (e-mail: kushalbose92@gmail.com; swagatamdas19@yahoo.co.in).

Digital Object Identifier 10.1109/TETCI.2023.3249255

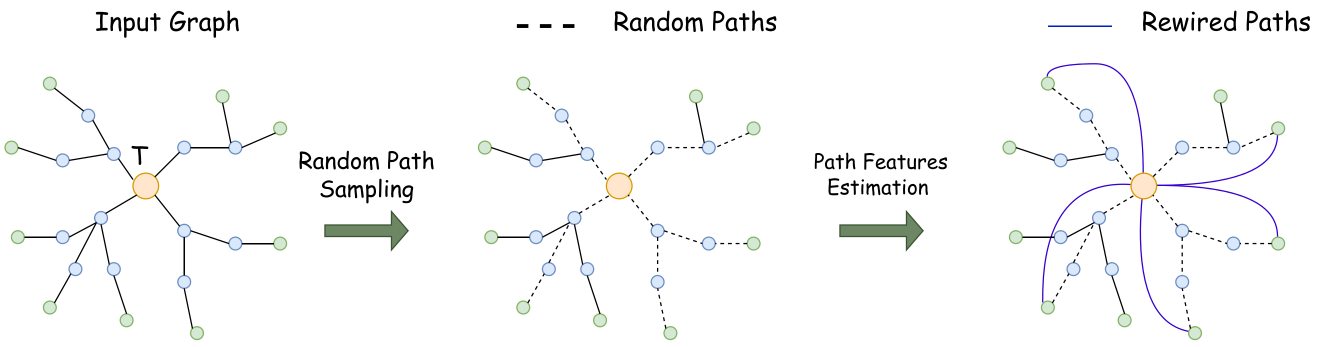

Fig. 1. The overview of RPE-GNN is represented. The center node T whose embedding would be evaluated via 3-hop neighborhood aggregation. Initially, random paths are generated gradually by layer-wise node sampling. The selected paths are marked with dotted lines. The path features are evaluated alongside in a dynamic way. Solid blue lines denote the estimated path features. The graph is rewired by adding edges to replace the sampled paths. The edges are incorporated with the respective path features. The newly added edges act as direct connections between the target node and its multi-hop neighbors. The neighborhood aggregation is performed on the newly rewired graph.

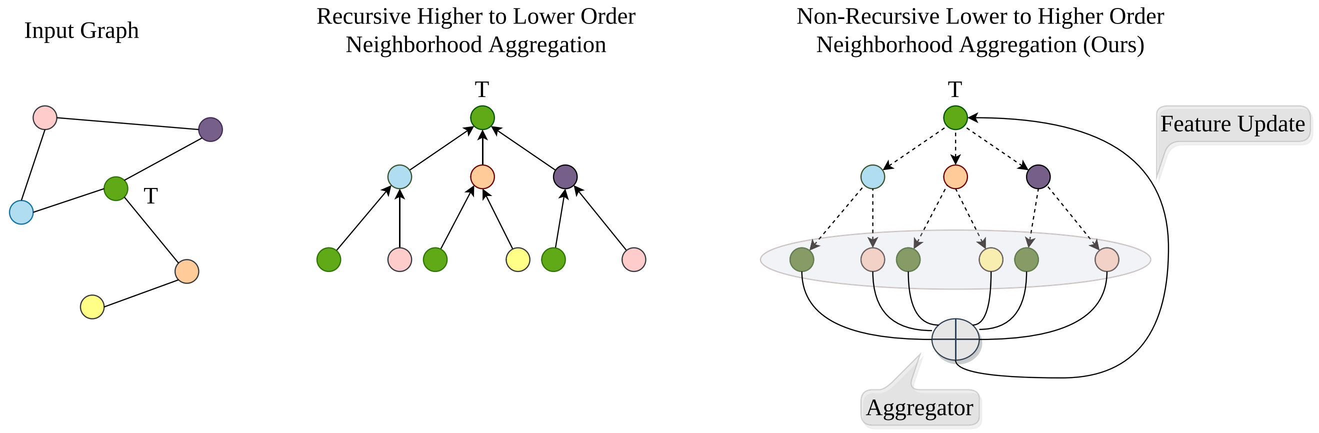
Input Graph
T
Recursive Higher to Lower Order Neighborhood Aggregation
T
Non-Recursive Lower to Higher Order Neighborhood Aggregation (Ours)
T
Feature Update
Aggregator

Fig. 2. Comparative study between the proposed aggregation method and existing aggregation methods is demonstrated. The 2-hop neighborhood aggregation is performed on the node T of the input graph(leftmost). In the case of the recursive aggregation(middle one), the aggregation is performed from higher to lower-order neighborhoods (bold lines with arrows). In this case, the information flow from the leaf nodes to the root node of the computation graph. On the other hand, the non-recursive aggregation method(rightmost) aggregates from lower to higher-order neighborhoods. The non-recursive method initially estimates the path features, and at the final stage, the aggregation is performed with the distant neighbors by employing those path features. Here, the information flows from the root node to the leaf nodes of the computation graph(dotted lines with arrows).

Motivated by those facts, we attempt to solve the issue by introducing a novel architecture. The architecture enables a message-passing strategy which is entirely different from existing message passing frameworks. The proposed approach consists of two steps: initially, the fixed-length random paths are sampled for a source node to explore its multi-hop neighbors. In the second stage, the path features are estimated by using some pre-defined rule. The sampled paths can be assumed as the connectors or edges between source nodes and randomly selected neighboring nodes. The graph is rewired by using those newly constructed edges, and edges are equipped with the features as corresponding estimated path features. After that, the message aggregation is performed on the source node by considering the newly rewired graph. The overview of the proposed algorithm is presented in Fig. 1.

When the number of random paths is increased the computational burden also enhances. To avoid the overhead we propose a dynamic path feature computation technique. The dynamic method allows the model not to store the predetermined random paths as a pre-processing step. The paths are gradually traced out while propagating through the hidden layers. The path features are also estimated alongside. The approach also explores better graph topology.

The aggregation strategy of RPE-GNN is also unique from other aggregation strategies. The existing aggregation techniques updates node features in every hidden layer, whereas we update node features only once which is after the complete estimation of the path features. Specifically, in our case, the hidden layers are utilized to compute path features rather than used for aggregation. In this way, our proposed approach can avoid redundant calculation of neighbors' features, effectively tackling the over-smoothing issue. Finally, we can summarize our contributions as follows,

- Unlike [5], [13], [14], [21] we design a novel strategy that aggregates from lower to higher-order neighborhoods in a non-recursive manner.
- Our approach learns node representations by estimating the features of the randomly explored paths in the graph,

which is described in Section III. Therefore, we termed it as Randomized Path Explorer-Graph Neural Networks or *RPE-GNN*.

- We design an efficient technique where paths are traced out gradually while propagating through the hidden layers without having prior knowledge of the predetermined paths.
- In Section III-C we show that our method simulates a biased random walk and the over-smoothing issue can be prevented by controlling the bias of the walk.
- We perform diversified experiments to showcase the efficacy of our framework in Section IV, and it outperforms existing state-of-the-art approaches.

II. Related Works

Multiple significant works offer resilience to the over-smoothing issue. Models like DeepGCN [20], JKNet [21] involves residual connections across the hidden layers to tackle over-smoothing. Meanwhile, GCNII [26] utilizes the fraction of the initial features and identity mapping with the weight matrices to update the node representations. DropEdge [24], and AdaEdge [25] reduce the rate of over-smoothing by learning to modify the underlying graph topology. Alternatively, normalization-based techniques like PairNorm [22], and DGN [23] introduces additional normalization layers to distinguish the node features in deeper architectures. On the other hand, Simplifying Graph Convolution (SGC) [27] manages to linearize GCN. APPNP [18] employs a costlier PageRank [28] matrix to perform higher order neighborhood aggregation while [19] relies on graph diffusion to learn the same. GPR-GNN [29] jointly optimize node features and topological information irrespective of the node homophily or node heterophily where Ortho-GConv [30] relies on orthogonal feature transformation to minimize the effect of over-smoothing.

NRGCN [31] proposes a non-recursive aggregation strategy that learns node embedding by extracting the information from the hops of the current set of neighbors. However, the work is not intended to tackle over-smoothing. On the other hand, GeniePath [32] explores the receptive paths of the graph in both breadth and width direction of the respective neighborhoods where SPAGAN [33] aggregates features from the higher-order neighbors by incorporating path-based attention where shortest paths are considered. Another significant work GraphSAINT [34] samples different subgraphs and construct the graph convolution layers to learn node features. RWGNN [35] designs a differentiable random walk kernel for learning graph representations. The over-smoothing problem is also possible in the hyper-graphs which is being discussed in [36]. Chen et al. [37] discusses the theoretical concepts of over-smoothing in terms of Dirichlet energy. Another approach, Deeply Supervised GNN (DSGNN) [38] learned representations from each layer and also contribute to the layer-wise loss functions. On the other hand Zhou et al. [39] designed Dirichlet energy-constrained graph neural networks, which control over-smoothing in deep models.

III. Proposed Method

A. Preliminaries

A graph $G = (V, E)$ consists of a set of n nodes V and a set of edges $E \subseteq (V \times V)$. The connections between nodes are represented by an adjacency matrix $A = [a_{ij}]_{n\times n}$ where $a_{ij} \in \{0, 1\}$. Evidently, $a_{ij} = 1$ indicates that the ith and jth nodes are connected by an edge, while $a_{ij} = 0$ denotes otherwise. The feature matrix of G is denoted by $X \in \mathcal{R}^{n\times f}$ where the ith row x_i denotes the f-dimensional feature representation of the ith node. The degree matrix D is a diagonal matrix where the ith element in the diagonal corresponds to the degree of the ith node. The graph Laplacian is defined as $L = D - A$. The symmetrically normalized graph Laplacian is represented as $N = D^{-\frac{1}{2}} L D^{-\frac{1}{2}}$.

B. Randomized Path Exploration

The core idea of our approach is randomly exploring multiple paths in the graph to aggregate multi-hop information. Simultaneously, we also learn features of the corresponding paths via propagating through the hidden layers. The path features are eventually used to evaluate the node embeddings. The node representations are updated only after the estimation of the path features.

Assume t is the node whose embedding would be evaluated using the k-hop distant neighbors. Consider $N_t^k = \{n_1, n_2, \ldots, n_l\}$ as the set containing all k-hop neighbors. The complete embedding method will be described in a top-down fashion to evaluate the embedding of t.

Path Feature Estimation: A path in the graph comprises a sequence of nodes contained in the graph. Assume a k-length path starting from t is denoted as $P_t = \{t = n_0, n_1 \cdots n_k = d\}$. A feature of a path is the sum of the features estimated over the individual edges of the path. The features of a single edge of a path can be estimated as below:

$$\mathbf{M}_{pq} = \sigma((x_p - x_q)W), \tag{1}$$

where p and q are the tail and head of the edge. $\mathbf{M}_{pq}$ is the feature of the corresponding edge e_{pq} and $W \in \mathbb{R}^{f\times f'}$ is the trainable weight matrix where f' denotes dimension of newly transformed space. The weight W is shared across all edges in the graph, which makes our architecture parameter efficient. $\sigma(.)$ represents the non-linear activation function. x_p and x_q are the features associated with the pth and qth node respectively.

As previously mentioned, a path's feature is evaluated as the sum of features of the individual edges of the respective path. Then the path P_t will have a message as shown below:

$$\mathbf{M}_P = \phi\left(\sum_{i=0}^{k-1} \mathbf{M}_{n_i n_{i+1}}\right), \tag{2}$$

where $\mathbf{M}_{n_i n_{i+1}}$ is the features of the edge $e_{n_i n_{i+1}}$ and $\phi(.)$ is a trainable function. Specifically, the feature of a path is the sole representation within the graph. The learned features of the paths contain information among the non-adjacent nodes.

Aggregation: The estimated path features will be employed to update the node embeddings. The path features encode the

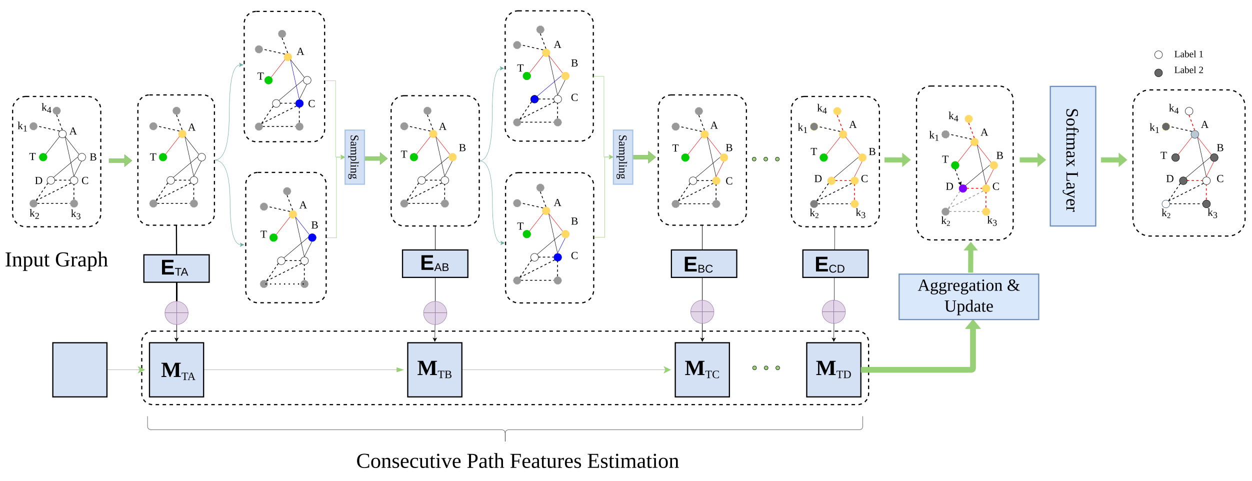

Fig. 3. The workflow of our proposed aggregation scheme is demonstrated. The node T (in green) is the target node whose embedding would be evaluated. The bold (black) lines denote edges between pairs of nodes, where the dotted lines (black) in the input graph show the existence of multiple intermediate nodes between any pairs of nodes. At each step, a neighbor (marked blue) of the current node (marked yellow) is randomly selected, and the features of the newly selected edge (bold red lines) are calculated. The current edge features are added with the total estimated features. The procedure is continued until the required length of path features is evaluated. The edges with the red color denote the complete traced out path by the random walk. The remaining grey nodes are denoted as unvisited nodes. After the path feature estimation, the aggregation step is performed with the distant neighbor (marked purple), and features of T are updated. The path features are denoted by the dotted line with an arrow directly connecting from T to D. After passing through the softmax layer, the final nodes are classified into the desired number of classes.

node sequences of the respective paths between the target nodes and the multi-hop neighbors. One can redefine path features as the features of the connecting edges between the pair of non-adjacent nodes. Our approach gathers information from the sequence of edges to generate complete path features rather than passing information to distant neighbors like in graph diffusion [19]. Consider the set of all possible k-length paths starting from t is denoted as $\mathcal{P}$. Therefore, the neighborhood aggregation is formulated as the following:

$$\tilde{x_t} = x_t + \frac{1}{N(\mathcal{P})} \sum_{p \in P, l \in N_t^k} \phi(\mathbf{M}_p) \odot x_l, \tag{3}$$

where $\tilde{x_t}$ denotes updated node embedding, $\mathbf{M}_p$ is the estimated features of the path $p \in \mathcal{P}$, x_l is the feature of the $n_l \in N_t^k$ which is the tail node of the path p. $\odot$ denotes the element-wise multiplication between the vectors. $N(\mathcal{P})$ is the number of elements of the set $\mathcal{P}$. The (3) aggregates information from the multi-hop node l, which is also a tail node of path p. The equation also acts as the message propagation rule of our proposed approach.

Dynamic Feature Computation: In (3) the set $\mathcal{P}$ contains all k length paths begin from node t. The number of paths will increase exponentially when k increases, and this will generate a massive computational cost. To prevent the issue, we devise a technique that computes path features dynamically by defining a recurrence relation. Assume a k-length path p whose node sequences are represented as $\{t = n_0, n_1, \ldots, n_k = d\}$. The feature of the path $\mathbf{M}_p$ or $\mathbf{M}_{n_0 n_k}$ is estimated dynamically in the following way:

$$\mathbf{M}^{(l+1)}_{n_0 n_{l+1}} = \mathbf{M}^{(l)}_{n_0 n_l} + \mathbf{M}^{(l+1)}_{n_l n_{l+1}}, \tag{4}$$

where l lies in $[1, (k-1)]$ and $\mathbf{M}^{(l+1)}_{n_0 n_{l+1}}$ denotes estimated path features node n_0 to n_{l+1} at the $(l+1)$th layer. Similarly, $\mathbf{M}^{(l)}_{n_0 n_l}$ indicates the computed features from n_0 to n_l at the lth layer. Again the features of the $(l+1)$th edge is $\mathbf{M}^{(l+1)}_{n_l n_{l+1}}$ which is estimated by using (1) as $\mathbf{M}^{(l+1)}_{n_l n_{l+1}} = \sigma((x_{n_l} - x_{n_{l+1}})W^{(l)})$ where $W^{(l)}$ denotes the trainable weight matrix of the lth layer.

The recurrence relation reduces the massive computational overhead during path feature estimation. Prior knowledge of the paths is not required as they would be gradually traced out while propagating through the hidden layers. Therefore, the features of the k-length paths are estimated by consecutively adding the features of k edges. As depicted in Fig. 3, at lth layer, we have already computed up to l-length features of a k-length path where $l \leq k-1$. Then at the $(l+1)$th layer we will only estimate the features of the $(l+1)$th edge of the corresponding path. The features of the $(l+1)$th edge will be added with the previously computed l-length features to evaluate the $(l+1)$-length path features. Hence, the feature computation is implemented by considering only the immediate estimated total features and the current edge features. Therefore, this dynamic approach prevents the algorithm from the unnecessary requirements of the predetermined random paths. The node features are updated only after evaluating the desired length of path features. No update is performed between any two hidden layers like the other prevalent aggregation strategies such as GCN, GAT, GraphSage, JKNet, GCNII, etc. This approach helps the model avoid redundant feature computation during the aggregation procedure.

C. Relation With Random Walk Statistics

In this section, we will establish a connection between RPE-GNN and random walk on the graph. Our proposed approach inherently simulates a biased random walk in the graph. To aggregate features from k-hop neighborhood we perform a k-length random walk in the graph. Suppose X_t denotes the state of the walk at the timestamp t and u, v are two connected nodes in the graph. Now, we can define the probability rule of the walk in the following way.

$$Pr[X_{t+1} = v | X_t = u] = \begin{cases} 1-p, & \forall u \neq v, v \in N(u). \\ p, & \text{otherwise.} \end{cases} \quad (5)$$

where $Pr[A]$ denotes the probability of occurring of event A, $0 < p < 1$ which controls the bias of the random walk. The $N(u)$ denotes the set of neighbors of the node u. In general, the transition matrix of the random walk is AD^{-1}. But following the lazy random walk [40] style, we can formulate the parameterized transition matrix as

$$W_p = pI + (1-p)AD^{-1}. \quad (6)$$

Now let us introduce the following lemma.

Lemma 3.1: If (λ, e) denotes the eigenvalue and eigenvector pair of the normalized Laplacian N respectively, then the corresponding eigenvalue and eigenvector pair of W_p is $(1-(1-p)\lambda, D^{\frac{1}{2}}e)$.

The proof is available in the Appendix A. The over-smoothing occurs due to the convergence of the biased random walk to the stationary distribution [26]. Suppose at time-stamp t the state transition probability vector is denoted as s_t^p, which depends upon the parameter p. The dependence of s_t^p on p can be deduced by proposing the following theorem.

Theorem 3.2: For any p_1, p_2 such that $0 < p_2 \leq p_1 < 1$, then the following inequality will hold

$$\left(s_t^{p_1} - \lim_{t\to\infty} s_t^{p_1}\right) \geq \left(s_t^{p_2} - \lim_{t\to\infty} s_t^{p_2}\right). \quad (7)$$

The proof is discussed in the Appendix A.B. From the above theorem, we can conclude that if p tends to 1, which will eventually slow the convergence of state transition probabilities to the stationary distribution. If $p \to 1$, then from (5) it can be stated that the probability of moving from the current node to its neighbors will become lesser than the probability of staying in the current node. This fact indicates that after infinitely many steps of the random walk, it is better to stay at the current vertex than to move to its neighbors. At that moment, exploring more paths does not improve the model's performance. The fact is also experimentally demonstrated in the Section IV-F.

D. Connection With Existing Approaches

RPE-GNN has some significant properties of the propagation rule over the other existing approaches such as GCN [5], GAT [13], and GraphSage-GCN [14]. Assume x and d are respectively the features and the degree of any node v, then the updated feature $\tilde{x}$ of GCN can be estimated as:

$$\tilde{x} = \frac{x}{d+1} + \sum_{j \in N(v)} \frac{1}{\sqrt{d+1)(d_j+1)}} x_j, \quad (8)$$

where x_j is the features of jth neighbor of node v and $N(v)$ is the set of neighbors of the same. In GCN the aggregated features are symmetrically normalized by the degree of the neighbors of the node v. Similarly, RPE-GNN normalizes the same with the total number of random paths. The random paths may be assumed as the edges incorporated with the estimated path features connecting the source node to its multi-hop neighbors. Therefore, our method tackles the aggregation of the non-adjacent neighbors through the connecting paths.

On the other hand, the feature aggregation of GAT is formulated as:

$$\tilde{x} = x + \sum_{j \in N(v)} \frac{a_{ij}}{d_j} x_j, \quad (9)$$

where a_{ij} is the attention coefficient between ith and jth node. The attention coefficient a_{ij} is multiplied by the neighbor's features where the identical attention value weights each dimension. But RPE-GNN weights each dimension of the same by the distinct values. This implies our approach can provide feature-level attention weights.

Another architecture called GraphSage-GCN has the following propagation formula.

$$\tilde{x}_i = x_i + \frac{1}{d_i} \sum_j x_j. \quad (10)$$

In this case, the neighborhood features are also simply averaged by the node degree like GCN where we average the weighted features by the number of randomly selected paths. GraphSage-GCN cannot provide feature-level attention whereas RPE-GNN enables feature-level attention which is learned from the random paths. The message propagation technique's highlighted features help prevent over-smoothing in the deeper GNN architectures.

The common objective between RPE-GNN and Graph Diffusion Convolution (GDC) [19] is to gather multi-hop neighborhood information to update the node features. However, GDC and RPE-GNN have fundamental differences while aggregating features from long-range dependency. Applying GDC, initially, we make the graph denser because the higher powers of the transition matrix lead to interaction with the higher-order neighbors. Then a dense graph is sparsified by removing the edges whose weights are below a pre-defined threshold value. In contrast, RPE-GNN is a novel architecture, which rewires the graphs by randomly exploring multi-hop neighbors via sampling random paths. RPE-GNN does not require any threshold value to rewire the graph and enjoys complete freedom to select any random neighbors.

IV. Experiments and Results

We perform diversified experiments to showcase the efficacy of our model. The experiments are done on the open benchmark graph datasets.

A. Details of Datasets

The experiments are carried out on four different categories of datasets.

Citation datasets: We select Cora [41], Citeseer [41], and Pubmed [41] as the citation networks where the nodes represent documents, and the edges act as citations between the documents. Each document belongs to one academic topic. The features associated with each node correspond to the bag-of-words representation of the respective document.

Co-authorship datasets: Coauthor CS [42] and Coauthor Physics [42] are two co-authorship networks. Nodes represent authors, and edges exist between them if they coauthored a paper. The features of the nodes represent the keywords related to the

TABLE I
DATASET STATISTICS

Dataset	Nodes	Edges	Classes	Features
Cora	2708	5429	7	1433
Citeseer	3327	4732	6	3703
Pubmed	19,717	44,338	3	500
Coauthor CS	18333	81894	15	6805
Coauthor Physics	34493	495924	5	8415
Amazon Photo	13752	491722	10	767
Amazon Computers	7650	238162	8	745
Chameleon	2277	36101	5	2325
Wisconsin	251	499	5	1703
Cornell	183	295	5	1703

author's paper. The label of each node denotes the field of study of the corresponding author.

Co-purchase datasets: Amazon Computers [42], and Amazon Photo [42] are two co-purchase networks where each node denotes products, and an edge exists if two products are bought frequently. Node features denote the bag-of-words representation of the product reviews. Node labels indicate the product category.

Wikipedia networks: Chameleon [43] is the Wikipedia network where nodes denote web pages from the Wikipedia pages, and edges exist if edges have mutual links between them. Node features are the bag-of-words of the nouns on the page. The node labels signify one of the five classes of average monthly web page traffic.

WebKB Datasets: We choose Wisconsin and Cornell [43] as our webkb datasets where nodes represent web pages, and edges are the hyperlinks between the web pages. The node features are the bag-of-words representation of the web pages. These datasets are heterophilic graphs where connected nodes are more likely to be in different classes.

Table I summarizes the details of the datasets.

The code is implemented by using Pytorch-Geometric [44] framework and is available at https://github.com/gnn-codes/rpe-gnn

B. Semi-Supervised Node Classification

Dataset Settings: For semi-supervised node classification, we use standard train-validation-test split of the three datasets Cora, Citeseer, and Pubmed as stated in [5]. For training 20 labeled samples are selected from each of the classes. For validation and testing, we select 500 nodes and 1000 nodes, respectively. For Coauthor CS, Coauthor Physics, Amazon Computers, and Amazon Photo, 20 samples are randomly selected from each class for training, 30 samples from each class are randomly selected for validation, and the rest of the samples are chosen for testing as described in [45].

Discussions: The performance of our model can be evident by comparing it with the other SOTA approaches. We consider GCN [5], GAT [13], SGC [27], Jumping Knowledge Networks [21], and APPNP [18] as our contenders. Table II provides necessary details regarding the performances of RPE-GNN. The method outperforms all results on three Cora, Citeseer, and Pubmed. The optimal layer numbers are written in the parenthesis. The baselines like GCN, GAT, and JKNet utilize

TABLE II
MEAN CLASSIFICATION ACCURACY(%) OF THE MODEL COMPARING WITH DIFFERENT STATE-OF-THE-ART APPROACHES

Method	Cora	Citeseer	Pubmed
GCN	81.1	70.8	79.0
GAT	83.0	71.5	79.0
SGC	81.7	71.3	78.9
JKNet	81.1	69.8	78.1
APPNP	83.1	71.8	80.1
RPE-GNN(Ours)	**83.3±0.02**(8)	**72.1±0.01**(16)	**80.8±0.05**(8)

TABLE III
MEAN CLASSIFICATION ACCURACY(%) OF THE RPE-GNN COMPARING WITH DIFFERENT STATE-OF-THE-ART APPROACHES WHERE GRAPH DIFFUSION IS EMPLOYED AS PRE-PROCESSING

Method	Cora	Citeseer	Pubmed
GDC-GCN	82.93	71.95	79.62
GDC-GAT	81.6	70.25	77.78
GDC-JKNet	82.78	71.87	79.95
GDC-APPNP	83.23	71.93	79.78
RPE-GNN	**83.3±0.02**	**72.1±0.01**	**80.8±0.05**
GDC-RPE-GNN	81.6 ±0.33	68.7 ±0.2	80.6 ±0.60

recursive message-passing frameworks. However, we employ non-recursive aggregation without performing redundant feature computations, and still, we achieve better performances over the datasets.

We also consider to pre-process data with graph diffusion convolution(GDC) [19] to evaluate the performance of RPE-GNN. We consider the variants of GNNs like GCN, GAT, GraphSAGE, and APPNP with the graph diffusion based pre-processing to compare the performances with the GDC-based RPE-GNN. The results are presented in Table III. From the results, it can be observed that applying graph diffusion does not improve model performance. Graph diffusion modifies the graph connections under some conditions, but RPE-GNN rewires the graph with long-distant nodes via sampling random paths.

To compare performances of RPE-GNN on co-authorship and co-purchase networks we include GCN [5], GAT [13], GraphSage [14], MoNet [46], and DAGNN [45] as our contenders. RPE-GNN outperforms all methods in the four datasets, as mentioned earlier. All said competitors utilize recursive aggregation in contrast to our aggregation, which is non-recursive and still achieves the best performances, which is demonstrated in Table IV. Coauthor CS and Coauthor Physics achieve the best performance when network depth is 2 and 8, respectively, whereas Amazon Computers and Amazon Photo accomplish the same when network depth is 8 and 32, respectively.

C. Analysis of Deep Models

We conduct a detailed study on the performance of RPE-GNN for the different network depths. We select three citation networks Cora, Citeseer, and Pubmed, along with two heterophilic datasets Wisconsin and Cornell. The number of hidden layers is chosen from the set $\{2, 4, 6, 8, 10, 12, 14, 16\}$. Fig. 4 illustrates the variation of test accuracy when the network depth is increased gradually. For citation datasets, the model performance

TABLE IV
MEAN CLASSIFICATION ACCURACY(%) IS PRESENTED BY COMPARING WITH THE STATE-OF-THE-ART ARCHITECTURES

Method	Coauthor CS	Coauthor Physics	Amazon Computers	Amazon Photo
GCN	91.1 $\pm$0.5	92.8 $\pm$1.0	82.6 $\pm$2.4	91.2 $\pm$1.2
GAT	90.5 $\pm$0.6	92.5 $\pm$0.9	78.0 $\pm$19.0	85.7 $\pm$20.3
MoNet	90.8 $\pm$0.6	92.5 $\pm$0.9	83.5 $\pm$2.2	91.2 $\pm$1.3
GraphSAGE-mean	91.3 $\pm$2.8	93.0 $\pm$0.8	82.4 $\pm$1.8	91.4 $\pm$1.3
GraphSAGE-maxpool	85.0 $\pm$1.1	90.3 $\pm$1.2	N/A	90.4 $\pm$1.3
GraphSAGE-meanpool	89.6 $\pm$0.9	92.6 $\pm$1.0	79.9 $\pm$2.3	90.7 $\pm$1.6
DAGNN	92.8 $\pm$0.9	94.0 $\pm$0.6	84.5 $\pm$1.2	92.0 $\pm$0.8
RPE-GNN(Ours)	**93.18 $\pm$0.02**	**94.87 $\pm$0.01**	**84.78 $\pm$0.05**	**93.14 $\pm$0.01**

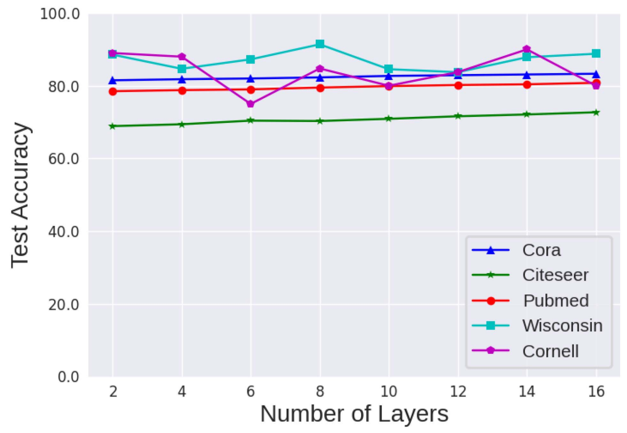

Fig. 4. The performance of RPE-GNN is observed by varying the network depths in the model. RPE-GNN achieves the best performance on Cora, Citeseer, and Pubmed at 8th, 16th, and 8th layers, respectively. RPE-GNN shows best results on Wisconsin and Cornell at 8th and 14th layers. Better test accuracy may be achieved when multiple layers are stacked together.

gradually improves throughout the experiment while the model depth increases. On the other side, the best performance of Wisconsin and Cornell is achieved in the deeper models. Therefore, the study justifies that going deeper into the GNN models is sometimes helpful to achieve optimal model performance. Thus, we can conclude that the proposed non-recursive aggregation strategy may mitigate the over-smoothing issue in the deeper architectures.

D. Full-Supervised Node Classification

Besides semi-supervised settings, we also evaluate RPE-GNN on fully-supervised node classification tasks.

Dataset Settings: In this case, we include the web data network Chameleon [43] along with the previous three datasets, Cora, Citeseer, and Pubmed. As per [26], the datasets are randomly split into $60\%, 20\%, 20\%$ for train, validation, and testing, respectively, for a fair comparison.

Discussions: In addition to previously mentioned baselines we further include Geom-GCN [43], APPNP [18] and SIGN [47] as our new contenders. We present the obtained results in Table V. The best results are mentioned with the optimal layer number within parentheses. RPE-GNN shows almost 1% improvement on Cora and Citeseer datasets compared to all approaches.

TABLE V
MEAN CLASSIFICATION ACCURACY(%) OF THE MODEL UNDER FULLY-SUPERVISED CONDITION

Method	Cora	Citeseer	Pubmed	Chameleon
GCN	85.77	73.68	88.13	28.18
GAT	86.37	74.32	87.62	42.93
Geom-GCN-I	85.19	77.99	90.05	60.31
Geom-GCN-P	84.93	75.14	88.09	60.90
Geom-GCN-S	85.27	74.71	84.75	59.96
APPNP	87.87	76.53	89.40	54.3
JKNet	85.25	75.85	88.94	60.07
JKNet(Drop)	87.46	75.96	89.45	62.08
Incep(Drop)	86.86	76.83	89.18	61.71
GCNII	88.49	77.08	89.57	60.61
GCNII*	88.01	77.13	**90.30**	62.48
RPE-GNN(Ours)	**89.15**(8)	**78.39**(8)	89.43(8)	**71.48**(16)

On the Pubmed dataset, the algorithm produces competitive results ($\leq 1\%$) compared to other approaches. On Pubmed still, RPE-GNN outperforms all other mentioned approaches except GCNII and GCNII*. For Chameleon, we achieve almost 9% improvements over SOTA. The results suggest that model performance can still be improved without having redundant feature estimation.

E. Architecture & Training

The architecture of RPE-GNN consists of a set of hidden layers, followed by an aggregation layer and, finally, a softmax layer. A k-layered RPE-GNN can perform k-hop neighborhood aggregation. Each hidden layer consists of a trainable weight matrix shared across all the edges for that layer and is followed by batch normalization layers, non-linear activation function ReLU and dropout layers.

The model is trained by Adam [48] optimizer with learning rate 0.2, and other parameters are set to defaults. The weight decay is set to 0.0005. We use 0.5 dropout to prevent potential overfitting. The model is trained for a maximum number of 200 steps, and the results are reported after averaging 10 steps during the test phase. The dimension of the hidden layers is 32. Cross-Entropy loss is used to optimize the model parameters. We also introduce a sampling strategy to generate random paths, which are broadly discussed in Appendix B.A.

F. Effect of Sampling of Random Paths

The effect on the model performance with a different number of sampled paths in the graph is demonstrated in Fig. 5. The experiment suggests that the test accuracy of the model significantly improves when the number of sampled paths is increased. The higher number of random paths enhances the scope of exploring the graph topology. Another observation is the test accuracy may not improve beyond an optimal number of random paths. The optimal number at which test accuracy becomes non-increasing depends on the dataset itself. One should note that for different depths of the model, the required number of random paths is different to achieve the best performance. In fact, for shallow architecture, the optimal number of paths is less than the requirement for the same in the case of deeper architecture.

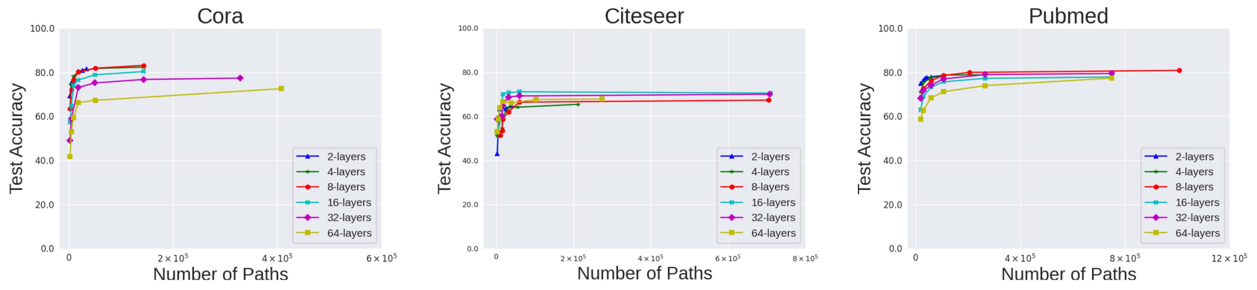

Fig. 5. Test accuracy of RPE-GNN is observed with varying the number of random paths for different depths of the architecture. Test accuracy increases when the number of sampled paths is increased. The model with more depth needs more random paths to produce optimal results.

V. Conclusion and Future Works

We propose RPE-GNN, an architecture that aggregates node features from lower to higher-order neighborhoods in a non-recursive way by considering path features of randomly explored paths in the graph. We design an efficient technique to estimate path features without having prior knowledge of the predetermined random paths. We have theoretically shown that RPE-GNN simulates a biased random walk in the graph. We have demonstrated RPE-GNN is capable of minimizing the effect of over-smoothing in the deeper GNN models through rigorous experiments. As a future work, the method corresponding to path features estimation can be further improved for better model performance. Also, our work can be extended to define a new field of study regarding the non-recursive aggregation strategies for graph learning tasks.

Appendix A

A. *Proof of Lemma 3.1*

Proof: If λ and e are the eigenvalue and the eigenvector of N respectively, then we have

$$Ne = \lambda e \tag{11}$$

By algebraic manipulation, the W_p can be rewritten as

$$W_p = \left(I - (1-p)D^{\frac{1}{2}}ND^{-\frac{1}{2}}\right) \tag{12}$$

Now, consider the following equation

$$\begin{aligned} W_pD^{\frac{1}{2}}e &= \left(I - (1-p)D^{\frac{1}{2}}ND^{-\frac{1}{2}}\right)D^{\frac{1}{2}}e \\ &= D^{\frac{1}{2}}e - (1-p)D^{\frac{1}{2}}Ne \\ &= D^{\frac{1}{2}}e - (1-p)D^{\frac{1}{2}}\lambda e[\text{from } (11)] \\ &= (1-(1-p)\lambda)D^{\frac{1}{2}}e \end{aligned} \tag{13}$$

Hence, the claim is proved. □

B. *Proof of Theorem 3.2*

Proof: The transition matrix of the biased random walk is derived as below

$$W_p = p.I + (1-p)AD^{-1}. \tag{14}$$

We know $L = D - A$ as the graph Laplacian. The symmetrically normalized Laplacian is represented as:

$$\begin{aligned} N &= D^{-\frac{1}{2}}LD^{-\frac{1}{2}} \\ &= D^{-\frac{1}{2}}(D-A)D^{-\frac{1}{2}} \\ &= I - D^{-\frac{1}{2}}AD^{-\frac{1}{2}} \end{aligned} \tag{15}$$

Reformulating the (14) by using the (15) we get

$$\begin{aligned} W_p &= p.I + (1-p)D^{\frac{1}{1}}D^{-\frac{1}{2}}AD^{-\frac{1}{2}}D^{-\frac{1}{2}} \\ &= p.I + (1-p)D^{\frac{1}{2}}(I-N)D^{-\frac{1}{2}} \\ &= p.I + (1-p)\left(I - D^{\frac{1}{2}}ND^{-\frac{1}{2}}\right) \\ &= I - (1-p)D^{\frac{1}{2}}ND^{-\frac{1}{2}} \end{aligned} \tag{16}$$

Suppose the state transition probabilities at time t is s_t them at time $t+1$ the state transition probabilities will be

$$s_{t+1} = W_p s_t \tag{17}$$

Substituting $t = 0$ we get $s_1 = W_p s_0$. As W_p is diagonalizable, then a set of eigenvectors forms a basis. Therefore, any vector can be expressed as

$$s_0 = \sum_{i=1}^{n} \alpha_i D^{\frac{1}{2}} e_i \tag{18}$$

where e_i is the ith eigen vectors and n is the number of dimension. Now, we can rewrite

$$\begin{aligned} s_1 &= \sum_{i=0}^{n} \alpha_i W_p D^{\frac{1}{2}} e_i \\ &= \sum_{i=0}^{n} \alpha_i (1-(1-p)\lambda_i) D^{\frac{1}{2}} e_i \end{aligned} \tag{19}$$

Iterating t times the expression will be

$$s_t = \sum_{i=0}^{n} \alpha_i (1-(1-p)\lambda_i)^t D^{\frac{1}{2}} e_i \tag{20}$$

It is a well-known fact that eigenvalues of N lie between $[0, 2]$. Therefore, it is easy to observe that the eigenvalues of W_p lie between $[0, 1]$. Assume $\lambda_1 = 1, \lambda_j \leq 1 \forall j \in [2, n]$. Finally, the stationary distribution of the random walk is

$$\begin{aligned} \lim_{t\to\infty} s_t &= \lim_{t\to\infty} \sum_{i=0}^{n} \alpha_i (1-(1-p)\lambda_i)^t D^{\frac{1}{2}} e_i \\ &= \sum_{i=0}^{n} \alpha_i \lim_{t\to\infty} (1-(1-p)\lambda_i)^t D^{\frac{1}{2}} e_i \\ &= \alpha_1 D^{\frac{1}{2}} e_1 \end{aligned} \tag{21}$$

It may observe that the eigenvalues of W_p other than 1 all lies between $[0, 1)$. Therefore, the rest of the terms vanish as t tends to infinity. Moreover, the expression s_t also depends on the parameter p so that we can rewrite it as s_t^p. WLOG for a fixed λ the following inequality will hold

$$(1-(1-p_1)\lambda) \geq (1-(1-p_2)\lambda), \tag{22}$$

where $p_1 \geq p_2$. For $t > 0$ the above inequality can be written as

$$\begin{aligned}
&(1-(1-p_1)\lambda)^t \geq (1-(1-p_2)\lambda)^t \\
&\Rightarrow \sum_{i=0}^{n}(1-(1-p_1)\lambda_i)^t \geq \sum_{i=0}^{n}(1-(1-p_2)\lambda_i)^t, \\
&\Rightarrow \sum_{i=0}^{n}\alpha_i(1-(1-p_1)\lambda_i)^t D^{\frac{1}{2}}e_i \\
&\geq \sum_{i=0}^{n}\alpha_i(1-(1-p_2)\lambda_i)^t D^{\frac{1}{2}}e_i, \\
&\Rightarrow s_t^{p_1} \geq s_t^{p_2} \\
&\Rightarrow (s_t^{p_1} - \epsilon) \geq (s_t^{p_2} - \epsilon) \\
&\Rightarrow \left(s_t^{p_1} - \lim_{t\to\infty} s_t^{p_1}\right) \geq \left(s_t^{p_2} - \lim_{t\to\infty} s_t^{p_2}\right)
\end{aligned}$$

From the (21), it can be deduced that the limiting value of the state transition probabilities is independent of p. Therefore, we can have $\lim_{t\to\infty} s_t^{p_1} = \lim_{t\to\infty} s_t^{p_2} = \epsilon$ for any p_1, p_2 where ϵ denotes the vector of the stationary distribution of the random walk. Hence, the theorem is proved. $\square$

Appendix B

This section provides the details of the hyper-parameters used to reproduce the results.

A. Sampling Strategy

We adopt a simple and effective layer-wise sampling technique that prevents increasing the exponential number of random paths. At the beginning of each layer, we initialize a parameter that decides the maximum of how many neighbors of a node can be selected. For every layer, the value of the parameter remains fixed for each node in the graph. The parameter is increased to explore more graph topology and is decreased to restrict the number of random paths.

We also introduce two random variables for layer-wise sampling, namely n_{sample} and l_{range}. The first one decides the maximum number of neighbors that will be selected for every node in the graph. This value will be different for every layer. The second one indicates the number of layers up to which the node-wise sampling will be performed. After that layer, the sampling will be 1, i.e., only one neighbor will be randomly selected. So the default value of n_{sample} is 1.

The hyper-parameters of the semi-supervised experiments are summarized in Table VI. Table VII reports the hyper-parameters of the fully-supervised experiments. The 'layers' denotes the network depth where the algorithm performs best on the corresponding dataset. Here L_2 represents the weight decay of the optimizer, and 'lr' denotes the learning rate used during the training.

TABLE VI
HYPER-PARAMETERS FOR SEMI-SUPERVISED NODE CLASSIFICATION EXPERIMENTS

Dataset	Hyper-parameters
Cora	layers: 8, l_{range}: 4, n_{sample}: 3, lr: 0.2, dropout: 0.50, L_2: 0.0005, hidden: 32
Citeseer	layers: 16, l_{range}: 4, n_{sample}: 4, lr: 0.2, dropout: 0.50, L_2: 0.0005, hidden: 32
Pubmed	layers: 8, l_{range}: 3, n_{sample}: 4, lr: 0.2, dropout: 0.50, L_2: 0.0005, hidden: 32
Coauthor CS	layers: 2, l_{range}: 2, n_{sample}: 4, lr: 0.2, dropout: 0.50, L_2: 0.0005, hidden: 32
Coauthor Physics	layers: 8, l_{range}: 2, n_{sample}: 4, lr: 0.2, dropout: 0.50, L_2: 0.0005, hidden: 32
Amazon Computers	layers: 8, l_{range}: 2, n_{sample}: 4, lr: 0.2, dropout: 0.50, L_2: 0.0005, hidden: 32
Amazon Photo	layers: 32, l_{range}: 3, n_{sample}: 4, lr: 0.2, dropout: 0.50, L_2: 0.0005, hidden: 32

TABLE VII
HYPER-PARAMETERS FOR FULLY-SUPERVISED NODE CLASSIFICATION EXPERIMENTS

Dataset	Hyper-parameters
Cora	layers: 8, l_{range}: 3, n_{sample}: 3, lr: 0.2, dropout: 0.50, L_2: 0.00005, hidden: 64
Citeseer	layers: 8, l_{range}: 2, n_{sample}: 3, lr: 0.2, dropout: 0.50, L_2: 0.00005, hidden: 64
Pubmed	layers: 8, l_{range}: 3, n_{sample}: 3, lr: 0.2, dropout: 0.50, L_2: 0.00005, hidden: 64
Chameleon	layers: 16, l_{range}: 3, n_{sample}: 3, lr: 0.02, dropout: 0.50, L_2: 0.00005, hidden: 64

References

[1] J. Masci, E. Rodolà, D. Boscaini, M. M. Bronstein, and H. Li, "Geometric deep learning," in *Proc. SIGGRAPH Asia Courses*, 2016, pp. 1–50.

[2] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, "Geometric deep learning: Going beyond euclidean data," *IEEE Signal Process. Mag.*, vol. 34, no. 4, pp. 18–42, Jul. 2017.

[3] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Trans. Neural Netw.*, vol. 20, no. 1, pp. 61–80, Jan. 2009.

[4] Z. Yang, W. Cohen, and R. Salakhudinov, "Revisiting semi-supervised learning with graph embeddings," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 40–48.

[5] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *Proc. Int. Conf. Learn. Representations (ICLR)*, 2017.

[6] A. Bojchevski, O. Shchur, D. Zügner, and S. Günnemann, "NetGAN: Generating graphs via random walks," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 610–619.

[7] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2016, pp. 855–864.

[8] D. Duvenaud et al., "Convolutional networks on graphs for learning molecular fingerprints," in *Proc. 28th Int. Conf. Neural Inf. Process. Syst.*, 2015, pp. 2224–2232.

[9] M. Zhang, Z. Cui, M. Neumann, and Y. Chen, "An end-to-end deep learning architecture for graph classification," in *Proc. 32nd AAAI Conf. Artif. Intell.*, 2018, pp. 4438–4445.

[10] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 1263–1272.

[11] G. A. Pavlopoulos et al., “Using graph theory to analyze biological networks,” *Biodata Mining*, vol. 4, no. 1, pp. 1–27, 2011.
[12] M. Koutrouli, E. Karatzas, D. Paez-Espino, and G. A. Pavlopoulos, “A guide to conquer the biological network era using graph theory,” *Front. Bioeng. Biotechnol.*, vol. 8, 2020, Art. no. 34.
[13] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, “Graph attention networks,” in *Proc. Int. Conf. Learn. Representations (ICLR)*, 2018. [Online]. Available: https://openreview.net/forum?id=rJXMpikCZ
[14] W. L. Hamilton, R. Ying, and J. Leskovec, “Inductive representation learning on large graphs,” in *Proc. 31st Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 1025–1035.
[15] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.
[16] Q. Li, Z. Han, and X.-M. Wu, “Deeper insights into graph convolutional networks for semi-supervised learning,” in *Proc. 32nd AAAI Conf. Artif. Intell.*, 2018, pp. 3538–3545.
[17] K. Oono and T. Suzuki, “Graph neural networks exponentially lose expressive power for node classification,” in *Proc. Int. Conf. Learn. Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=S1ldO2EFPr
[18] J. Klicpera, A. Bojchevski, and S. Günnemann, “Predict then propagate: Graph neural networks meet personalized pagerank,” in *Proc. Int. Conf. Learn. Representations*, 2018.
[19] J. Klicpera, S. Weißenberger, and S. Günnemann, “Diffusion improves graph learning,” in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2019, vol. 32, pp. 13354–13366.
[20] G. Li, M. Muller, A. Thabet, and B. Ghanem, “DeepGCNs: Can GCNs go as deep as CNNS?,” in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2019, pp. 9267–9276.
[21] K. Xu, C. Li, Y. Tian, T. Sonobe, K.-i. Kawarabayashi, and S. Jegelka, “Representation learning on graphs with jumping knowledge networks,” in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 5453–5462.
[22] L. Zhao and L. Akoglu, “Pairnorm: Tackling oversmoothing in GNNs,” 2019, *arXiv:1909.12223*.
[23] K. Zhou, X. Huang, Y. Li, D. Zha, R. Chen, and X. Hu, “Towards deeper graph neural networks with differentiable group normalization,” in *Proc. 34th Annu. Conf. Neural Inf. Process. Syst.*, 2020, pp. 1–12.
[24] Y. Rong, W. Huang, T. Xu, and J. Huang, “DROPEDGE: Towards deep graph convolutional networks on node classification,” in *Proc. Int. Conf. Learn. Representations*, 2019.
[25] D. Chen, Y. Lin, W. Li, P. Li, J. Zhou, and X. Sun, “Measuring and relieving the over-smoothing problem for graph neural networks from the topological view,” in *Proc. AAAI Conf. Artif. Intell.*, 2020, vol. 34, pp. 3438–3445.
[26] M. Chen, Z. Wei, Z. Huang, B. Ding, and Y. Li, “Simple and deep graph convolutional networks,” in *Proc. Int. Conf. Mach. Learn.*, 2020, pp. 1725–1735.
[27] F. Wu, A. Souza, T. Zhang, C. Fifty, T. Yu, and K. Weinberger, “Simplifying graph convolutional networks,” in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 6861–6871.
[28] L. Page, S. Brin, R. Motwani, and T. Winograd, “The PageRank citation ranking: Bringing order to the web,” Stanford InfoLab, Tech. Rep., 1999.
[29] E. Chien, J. Peng, P. Li, and O. Milenkovic, “Adaptive universal generalized PageRank graph neural network,” 2020, *arXiv:2006.07988*.
[30] K. Guo, K. Zhou, X. Hu, Y. Li, Y. Chang, and X. Wang, “Orthogonal graph neural networks,” in *Proc. AAAI Conf. Artif. Intell.*, 2022, vol. 36, pp. 3996–4004.
[31] H. Chen, Z. Deng, Y. Xu, and Z. Li, “Non-recursive graph convolutional networks,” in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, 2021, pp. 3205–3209.
[32] Z. Liu et al., “Geniepath: Graph neural networks with adaptive receptive paths,” in *Proc. AAAI Conf. Artif. Intell.*, 2019, vol. 33, pp. 4424–4431.
[33] Y. Yang, X. Wang, M. Song, J. Yuan, and D. Tao, “SPAGAN: Shortest path graph attention network,” 2021, *arXiv:2101.03464*.
[34] H. Zeng, H. Zhou, A. Srivastava, R. Kannan, and V. Prasanna, “GraphSAINT: Graph sampling based inductive learning method,” 2019, *arXiv:1907.04931*.
[35] G. Nikolentzos and M. Vazirgiannis, “Random walk graph neural networks,” in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2020, pp. 16211–16222.
[36] G. Chen and J. Zhang, “Preventing over-smoothing for hypergraph neural networks,” 2022, *arXiv:2203.17159*.
[37] C. Cai and Y. Wang, “A note on over-smoothing for graph neural networks,” 2020, *arXiv:2006.13318*.
[38] C. Yang, R. Wang, S. Yao, S. Liu, and T. Abdelzaher, “Revisiting oversmoothing in deep GCNs,” 2020, *arXiv:2003.13663*.
[39] K. Zhou et al., “Dirichlet energy constrained learning for deep graph neural networks,” in *Proc. Adv. Neural Inf. Process. Syst.*, 2021, vol. 34, pp. 21834–21846.
[40] G. Wang, R. Ying, J. Huang, and J. Leskovec, “Improving graph attention networks with large margin-based constraints,” 2019, *arXiv:1910.11945*.
[41] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad, “Collective classification in network data,” *AI Mag.*, vol. 29, no. 3, pp. 93–93, 2008.
[42] O. Shchur, M. Mumme, A. Bojchevski, and S. Günnemann, “Pitfalls of graph neural network evaluation,” 2018, *arXiv:1811.05868*.
[43] H. Pei, B. Wei, K. C.-C. Chang, Y. Lei, and B. Yang, “Geom-GCN: Geometric graph convolutional networks,” 2020, *arXiv:2002.05287*.
[44] M. Fey and J. E. Lenssen, “Fast graph representation learning with PyTorch geometric,” 2019, *arXiv:1903.02428*.
[45] M. Liu, H. Gao, and S. Ji, “Towards deeper graph neural networks,” in *Proc. 26th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2020, pp. 338–348.
[46] F. Monti, D. Boscaini, J. Masci, E. Rodola, J. Svoboda, and M. M. Bronstein, “Geometric deep learning on graphs and manifolds using mixture model CNNs,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 5115–5124.
[47] F. Frasca, E. Rossi, D. Eynard, B. Chamberlain, M. Bronstein, and F. Monti, “SIGN: Scalable inception graph neural networks,” 2020, *arXiv:2004.11198*.
[48] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” 2014, *arXiv:1412.6980*.

Kushal Bose received the B.Tech. degree in electronics and communication engineering from the Heritage Institute of Technology, Kolkata, India, in 2015, and the M.Tech. degree in computer science in 2019 from the Indian Statistical Institute, Kolkata, where he is currently working towards Ph.D. degree. His research interests include geometric deep learning, graph neural networks, non-euclidean neural networks, and their applications in real-world scenarios.

Swagatam Das (Senior Member, IEEE) received the B.E. Tel. E., M.E. Tel. E (control engineering specialization) and Ph.D. degrees from Jadavpur University, Kolkata, India, in 2003, 2005, and 2009, respectively. He is currently an Associate Professor and the Head with the Electronics and Communication Sciences Unit, Indian Statistical Institute, Kolkata, India. He has authored or coauthored more than 300 research articles in peer-reviewed journals and international conferences and has more than 28,000 Google Scholar citations and an H-index of 79 till. His research interests include evolutionary computing and machine learning. He is also the founding Co-Editor-in-Chief of *Swarm and Evolutionary Computation*, an international journal from Elsevier. He was or is an Associate Editor for the IEEE TRANSACTIONS ON CYBERNETICS, IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, *Pattern Recognition* (Elsevier), *Neurocomputing* (Elsevier), *Information Sciences* (Elsevier), and IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS: SYSTEMS. He is an Editorial Board Member of *Information Fusion* (Elsevier), *Progress in Artificial Intelligence* (Springer), *Applied Soft Computing* (Elsevier), *Engineering Applications of Artificial Intelligence* (Elsevier), and *Artificial Intelligence Review* (Springer). He has been associated with the international program committees and organizing committees of several reputed international conferences including NeurIPS, AAAI, AISTATS, ACM Multimedia, BMVC, IEEE CEC, and GECCO. He was Guest Editors for special issues in journals like IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION and IEEE Transactions on SMC, Part C. He was the recipient of the 2012 Young Engineer Award from the Indian National Academy of Engineering (INAE) and 2015 Thomson Reuters Research Excellence India Citation Award as the highest cited Researcher from India in Engineering and Computer Science category between 2010 to 2014.