Journal of Computational Science 62 (2022) 101695

Contents lists available at ScienceDirect

Journal of Computational Science

journal homepage: www.elsevier.com/locate/jocs

Simplifying approach to node classification in Graph Neural Networks

Sunil Kumar Maurya [a,b,*], Xin Liu [b], Tsuyoshi Murata [a,b]

[a] Department of Computer Science, Tokyo Institute of Technology, Tokyo, Japan
[b] Artificial Intelligence Research Center, AIST, Tokyo, Japan

ARTICLE INFO

Keywords:
Graph Neural Networks
Node classification
Feature selection

ABSTRACT

Graph Neural Networks (GNNs) have become one of the indispensable tools to learn from graph-structured data, and their usefulness has been shown in wide variety of tasks. In recent years, there have been tremendous improvements in architecture design, resulting in better performance on various prediction tasks. In general, these neural architectures combine node feature aggregation and feature transformation using learnable weight matrix in the same layer. This makes it challenging to analyze the importance of node features aggregated from various hops and the expressiveness of the neural network layers. As different graph datasets show varying levels of homophily and heterophily in features and class label distribution, it becomes essential to understand which features are important for the prediction tasks without any prior information. In this work, we decouple the node feature aggregation step and depth of graph neural network, and empirically analyze how different aggregated features play a role in prediction performance. We show that not all features generated via aggregation steps are useful, and often using these less informative features can be detrimental to the performance of the GNN model. Through our experiments, we show that learning certain subsets of these features can lead to better performance on wide variety of datasets. Based on our observations, we introduce several key design strategies for graph neural networks. More specifically, we propose to use softmax as a regularizer and "soft-selector" of features aggregated from neighbors at different hop distances; and L2-Normalization over GNN layers. Combining these techniques, we present a simple and shallow model, Feature Selection Graph Neural Network (FSGNN), and show empirically that the proposed model achieves comparable or even higher accuracy than state-of-the-art GNN models in nine benchmark datasets for the node classification task, with remarkable improvements up to 51.1%. Source code available at https://github.com/sunilkmaurya/FSGNN/.

1. Introduction

Rapid growth of computational technologies and their applications has enabled us to gather data in a wide range of fields. The availability of such data has opened a lot of opportunities for analysis in both academic and commercial applications. Often the collected data can be represented as graphs encoding the relationships between entities. For example, online social networks can be represented as graphs where users are denoted as nodes and friendship status between them as edges. Usually, users sharing friendships also share many common attributes like communities, interests, political opinions, etc. Thus, analysis of graph-structured data can often uncover hidden attributes or missing information. One of the emerging tools for such analysis is neural networks that can learn on large amounts of data and can provide useful and interesting predictions about the properties of the data.

Graph Neural Networks (GNNs) are neural network models commonly used on graph-structured data. Based on user-defined relationships in the data, a graph is constructed, and nodes/edges are assigned attributes (or features) in the form of real-valued vectors. Using these feature vectors as input, GNNs perform feature aggregation step where each node collects feature information from its neighbors based on the graph. Utilizing the feature aggregation step, GNNs learn over similarity/dissimilarity of node features and use this information for various prediction tasks. With rapid development of the field and easy accessibility of computation and data, GNNs have been used to solve a variety of problems like node classification [1–5], link prediction [6–8], graph classification [9,10], prediction of molecular properties [11,12], node ranking [13,14] and natural language processing [15].

One of the popular applications of GNN is the node classification task, where a GNN model learns to predict unknown labels of the nodes based on similarity or dissimilarity to the nodes for which labels are

* Corresponding author at: Department of Computer Science, Tokyo Institute of Technology, Tokyo, Japan.
E-mail addresses: skmaurya@net.c.titech.ac.jp (S.K. Maurya), xin.liu@aist.go.jp (X. Liu), murata@c.titech.ac.jp (T. Murata).

https://doi.org/10.1016/j.jocs.2022.101695
Received 7 November 2021; Received in revised form 9 April 2022; Accepted 28 April 2022
Available online 11 May 2022
1877-7503/© 2022 Elsevier B.V. All rights reserved.

known. Early GNN models such as Graph Convolutional Network (GCN) [1] use simple feature aggregation schemes. Since then, researchers have incorporated many techniques from deep learning literature and proposed numerous GNN variants [16] to address various shortcomings in GNN model training and to improve the prediction capabilities. Some of the techniques used in these variants include neighbor sampling [17,18], attention mechanism [2], use of Personalized PageRank matrix instead of adjacency matrix [19], leveraging proximity in feature space [20] and simplified model design [21]. Also, there has been an increasing trend in making the models deeper by stacking more layers and using the residual connections to improve the expressiveness of the model [4,22]. For example, GCNII [4] incorporates up to 64 layers with scaled residual weights. However, most of these models by design are more suitable for homophily datasets, where nodes that are linked to each other are more likely to belong to the same class. As a result, these GNNs may not perform well with heterophily datasets, which are more likely to have nodes with different labels connected together. This problem was highlighted by Zhu et al. [23] and the authors proposed node's ego-embedding and neighbor-embedding separation to improve performance on heterophily datasets. Other recent works approach this problem in different manner e.g, [24] utilizes belief propagation, [25] uses adaptive gating on edges etc.

With increasing complexity in GNN architecture design, it becomes challenging to analyze GNN models and determine which components contribute more to the model's performance over a specific prediction task. In this work, we study the problem of node classification using GNN and approach it using feature importance and feature selection perspective. With results from extensive experiments, we show that input features often contain noisy components, and minimizing their effects during learning can improve prediction performance even with a simple neural network model. This also highlights that with simple feature-centric approach, use of complex and deep GNN models is often unnecessary. Based on our analysis, we propose a simple GNN model to improve prediction accuracy in the node classification task.

Our Contributions

- We run extensive experiments on multiple node classification benchmark datasets and show that aggregated features generated in GNNs have noisy components. In addition, feature selection is an essential requirement for higher prediction accuracy.
- We experimentally confirm the feature generation requirements for homophily and heterophily graphs.
- We propose a simple 2-layered GNN model, "FSGNN", which incorporates a "soft-selection" mechanism to learn the importance of features during training of the model.
- Our proposed model empirically outperforms other state-of-art (SOTA) GNN models (both shallow and deep) and achieves up to 51.1% higher node classification accuracy.

The rest of the paper is organized as follows: Section 2 outlines node classification task and formulation of graph neural networks. In Section 3, we explore the importance of features by running extensive experiments and present our observations. Based on our experimental observations, in Section 4, we propose our GNN model FSGNN. In Section 5, we briefly introduce relevant GNN literature. Section 6 contains the experimental details and comparison with other GNN models. In Section 7, we present our results and empirically analyze our proposed design strategies and their effect on the model's performance. Section 8 summarizes the paper.

2. Preliminaries

2.1. Mathematical notations

Let $G = (V, E)$ be an undirected graph with n nodes and m edges. For numerical calculations, graph is represented as adjacency matrix denoted by $A \in \{0,1\}^{n\times n}$ with each element $A_{ij} = 1$ if there exists an edge between node v_i and v_j, otherwise $A_{ij} = 0$. When self-loops are added to the graph then, resultant adjacency matrix is denoted as $\tilde{A} = A + I$. Diagonal degree matrix of A and $\tilde{A}$ are denoted as D and $\tilde{D}$ respectively. Each node is associated with a d-dimensional feature vector and the feature matrix for all nodes is represented as $X \in \mathbb{R}^{n\times d}$. In our discussion, we will interchangeably refer to feature matrix as features of the nodes.

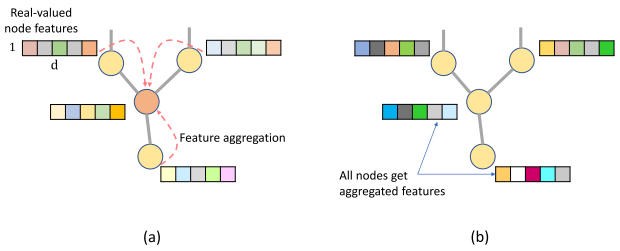

Fig. 1. (a) Each node has a d-dimensional feature vector and aggregates the features from its neighbors. (b) After aggregation step, each node gets an updated aggregated feature vector with neighborhood information.

2.2. Node classification

The rapid advancement of technologies has enabled us to capture and store a tremendous amount of data. However, often this data is noisy and incomplete. For example, in social networks, users may not provide all profile related information like their geographical location, hobbies, interests, political inclination etc. These data attributes can be considered as labels associated with the users [26]. The labels for users can be described in many ways, for example, labels based on gender, interests, hobbies, locations, etc. By predicting these labels, online platforms can help users by suggesting them new friends with similar profiles or recommending them new products through advertisement systems. Considering a social network as a graph with nodes as users and friendship between users as edges, only a subset of nodes are labeled. Node classification is the method to classify/group all nodes based on their likely labels.

Definition 1. For given graph $G = (V, E)$ with subset of nodes $V \subset V$ that are labeled, and $V = V\ V$ is the set of unlabeled nodes. Let Y be the set of m possible labels, and $Y = \{y_1, y_2, \ldots, y\}$ be the known labels of nodes of V. Node classification is the task to infer labels Y for all nodes in the graph.

One approach to solve this problem is to utilize human experts to label the nodes. However, this approach does not scale with large-sized graphs with hundreds of thousands to millions of nodes. Hence, researchers have approached this problem using computational based learning approaches. These approaches take advantage of the fact that nodes with the same labels would often share common attributes.

Learning on graph data often requires transforming the data into appropriate formats. Graph is represented as an adjacency matrix or a laplacian matrix. Attribute information pertaining to the nodes or edges is often stored in a real-valued vector representation. For example, for a node representing a user in a social network, the attributes from the user's profile are encoded in numeric form. Each index of the vector represents an individual attribute. Each node has a d-dimensional vector representation, also known as node feature vector, encoding all node attribute information (Fig. 1). $X \in \mathbb{R}^{n\times d}$ is node feature matrix containing feature vectors of all nodes. For the node classification problem, A and X are main input for learning algorithms.

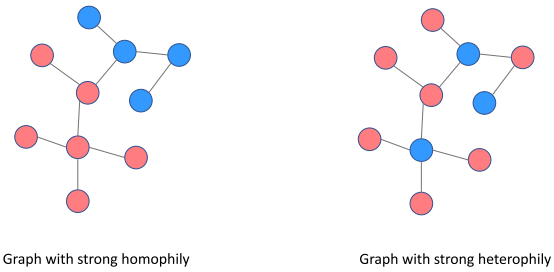

Fig. 2. Graph with strong homophily have nodes with similar attributes or labels connected together. In case of graphs with strong heterophily, nodes with dissimilar attributes or labels are connected to each other.

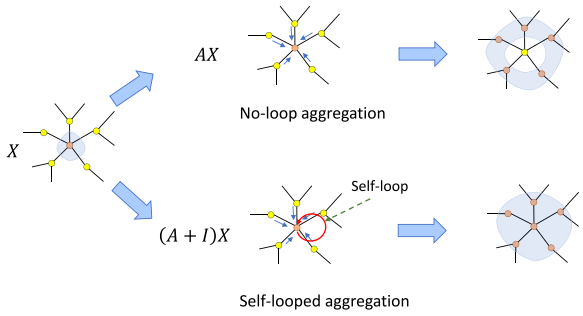

Fig. 3. No-loop (top) and Self-loop (bottom) aggregation of node features. No-loop aggregation step does not combine the node's features to neighbors' as they are dissimilar (heterophily). Self-looped aggregation step combines the node's features with neighbors' as they share common attributes (homophily).

2.3. Homophily and heterophily

Homophily or heterophily in a graph describes similarity or dissimilarity of attributes of connected nodes [23,27]. A graph is said to have high homophily when most edges connect nodes with similar attributes or same labels. Conversely, a graph has high heterophily if most edges connect nodes with dissimilar attributes or different labels (Fig. 2). One example of homophily is social networks where users with common attributes like geography, interests, political affiliations etc. are friends with each other. An example of heterophily in graphs is some dating networks where a user connects to other users of the opposite gender.

One way to measure homophily/heterophily in graph is through edge homophily ratio ($\hbar$) [23,24]. It is defined as a fraction of the number of edges with both nodes having same labels to the total number of edges in the graph.

$$\hbar = \frac{|\{(v_i, v_j) : (v_i, v_j) \in E \wedge y_i = y_j\}|}{|E|} \tag{1}$$

Graphs with strong homophily has high edge homophily ratio, $\hbar \to 1$, while graphs with strong heterophily has small edge homophily ratio, $\hbar \to 0$. Real-world datasets often show varying level of edge homophily ratio. Table 2 shows homophily ratio of some standard graph datasets.

As the node classification task requires identifying similarities between node attributes, the study of homophily and heterophily properties in the data helps to guide the design of the GNNs. In the next section, we provide more details on the basic architecture design of the GNN model.

2.4. Graph neural networks

Graph Neural Networks (GNNs) are a class of neural network models specifically designed to learn on graph-structured data. GNNs learn to map high dimensional graph data to low dimensional node representations, which are then used for predictions on various downstream tasks. Nodes with similar input features are embedded closer to each other in the embedding vector space, while dissimilar nodes have a greater distance between them (Fig. 4).

2.4.1. Hop-feature generation

One of the primary operations of GNN is feature aggregation. In the feature aggregation step, each node combines the features of its neighbors (Fig. 1). The number of aggregation steps corresponds to the number of hops a neighbor node is from the source node for feature aggregation. For example, three aggregation steps correspond to the node aggregating features of neighboring nodes lying at up to 3-hop distance from the node in the graph. Sharing of feature information among neighbors helps to define common neighborhood properties, which in turn helps GNN models to learn and predict effectively.

Feature aggregation step is calculated by matrix multiplication of adjacency matrix with node feature matrix. For K-step aggregation, aggregated features are calculated as $A^k X \in \mathbb{R}^{n \times d}, k \in (1 \cdots K)$. Hence, with multiple aggregation steps, each node gets an additional feature information vector corresponding to the neighborhood at each hop.

2.4.2. Layer design

GNNs leverage feature propagation mechanism [11] to aggregate neighborhood information of a node and use non-linear transformation with trainable weight matrix to get the final embeddings for the nodes. Conventionally, a simple GNN layer is defined as

$$H^{(i+1)} = \sigma(\tilde{A}_{sym} H^{(i)} W^{(i)}) \tag{2}$$

where $\tilde{A}_{sym} = \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}}$ is a symmetric normalized adjacency matrix with added self-loops. $H^{(i)}$ represents features from the previous layer, $W^{(i)}$ denotes the learnable weight matrix, and σ is a non-linear activation function, which is usually ReLU in most implementations of GNNs. However, this formulation is suitable for homophily datasets as features are cumulatively aggregated, i.e. node's own features are added together with neighbor's features. The cumulative aggregation of node's self-features with that of neighbors reinforces the signal corresponding to the label and helps to improve accuracy of the predictions. On the other hand, in the case of heterophily, nodes are assumed to have dissimilar features and labels to their neighbors. For heterophily datasets, H2GCN [23] proposes to separate features of neighbors from node's own features, thus avoiding aggregation of dissimilar features. So we use the following formulation for the GNN layer,

$$H^{(i+1)} = \sigma(A_{sym} H^{(i)} W^{(i)}) \tag{3}$$

where $A_{sym} = D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$ is symmetric normalized adjacency matrix without added self-loops. Fig. 3 shows difference between no-loop and self-looped aggregation. To combine features from multiple hops, concatenation operator can be used before the final layer.

One of the popular example of GNN is Graph Convolutional Network (GCN) [1]. GCN is a two-layered GNN model with layer design based on Eq. (2). Formulation of GCN model is defined as,

$$Z = \tilde{A}_{sym} \sigma(\tilde{A}_{sym} X W^{(0)}) W^{(1)} \tag{4}$$

GCN aggregates the features of two hops of neighbors and outputs Z. Softmax function is applied row-wise, and cross-entropy error is calculated over all labeled training examples. The gradients of loss are back-propagated through the GNN layers. Once trained, the model can be used to predict labels of the nodes in the test set.

2.5. Node classification with GNNs

Node classification is an extensively studied graph based semi-supervised learning problem. It encompasses training the GNN to predict unknown labels of nodes based on the features and neighborhood

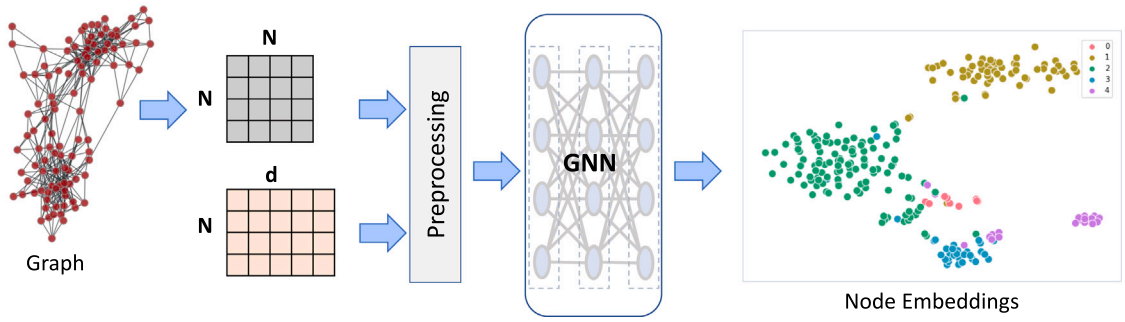

Fig. 4. Figure shows a simple representation of graph neural network. Graph is represented as adjacency matrix and along with node feature matrix is set as input to GNN model. Preprocessing includes normalization of the adjacency matrix and the node feature matrix. GNN models learns to map input node features to node representations in embedding space. Colors of the nodes denote their class labels.

structure of the nodes. GNN model is considered as a function $f(X, A)$ conditioned on node features X and adjacency matrix A. After training the model, nodes with similar features and proximity in graph structure are embedded closer to each other in vector space forming clusters as shown in Fig. 4. Then, unknown labels of the nodes can be predicted based on distance from labeled nodes in vector or embedding space.

For clarity, we provide an example of node classification using Cora, a real-world benchmark dataset for node classification.

Example: *Cora is a citation graph for machine learning papers. It has 2708 nodes and 5429 edges.*

Nodes V: Each academic paper is denoted as a node.

Edges E: An edge is defined when a paper cites another paper.

Node Labels Y: The papers are divided into seven classes based on the research field of these papers. Labels for nodes in the training set are visible to the GNN model. Once GNN is trained, predicted labels of the test set are compared with real labels.

Node Features X: Node feature vector is bag of word representation of 1433 distinct words [23,28]. Each index contains 1 or 0 depending upon the presence or absence of the word in the paper. Papers in the same field may share common words related to the area. Hence, the nodes representing them will have similar feature vectors.

In order to train a GNN model, the dataset is split into training, validation, and test set. Labels of nodes in the training set are accessible to the GNN model during training. Validation set is used to tune the hyper-parameters of the model. Once the model is trained, labels of nodes in the test set are predicted using the model. Predicted labels are compared with the real labels of the nodes and the accuracy of prediction is calculated. Fig. 8 (Appendix D) shows the embedding plots of original node features and learned node embeddings after training a GNN model on the Cora dataset.

2.6. Feature generation in GNNs

As discussed in Section 2.4, we can generate different features for the nodes capturing homophilic and heterophilic properties in the graph by modifying the feature aggregation step. The number of different features generated is dependent on the number of hops of aggregation over neighbors. In addition, node features can also be generated based on proximity in feature space or based on some other arbitrary criterion. Feature generation steps can differ among GNNs based on their architecture. Many GNN models have feature aggregation and representation learning combined in single layer [1,4,19], while in other models features can be precomputed beforehand [21,29].

2.7. Challenges with current GNNs

Following the progress in the deep learning field, there have been many recent advances in the design of GNNs. Recent works have proposed to make models deeper, using residual connections, attention layers, scaling output of GNN layers and residual connections etc. However, these improvements also bring challenges of increased complexity in GNN models. The new GNN models requires more number of hyper-parameters that need to be tuned. Moreover, it also becomes difficult to analyze which components of the model are contributing to the performance in prediction tasks. These challenges make it harder for any practitioner to decide which GNN model or architecture to use out of many when approaching a new problem or task.

In the next section, we reevaluate the problem of node classification and try to simplify the approach to GNN model design. We explore the problem from the perspective of feature selection and demonstrate that we can reduce the complexity of the GNN model by using an appropriate feature selection strategy.

3. Feature selection in graph neural networks

On any given graph-structured data, a set of features can be generated for the nodes (e.g. using Eqs. (1) & (2)). The number of features depends on the problem in hand, properties of the dataset, design choice of practitioner etc.

We assume a function,

$$g(X, A, K) \mapsto \{X_1, X_2, \ldots, X_l\}$$

The function takes $X \in \mathbb{R}^{n \times d}$ as node features matrix, A as an adjacency matrix, K as the power of the adjacency matrix or number of hops to propagate features and outputs a set of l node features.

With the increase in value of K, the node aggregates features of other nodes that lie multiple hops away from it. Features of such nodes may not be similar or correlated with that of the source node. Thus, some aggregated features may be uninformative for learning over a task. Finding the best value of K can require some experimental exploration, and often practitioner chooses K based on the task and dataset.

In the node classification task, for given label distribution, only a subset of these features are relevant to predict the label of the node. For example, a feature X_i is relevant to class C_i, if X_i and C_i are highly correlated [30–32]. Irrelevant or noisy features may not correlate with target labels but can still affect the learning process.

In this section, we explore the importance of features generated by the aggregation step at different hops. We run a series of extensive experiments to study how different features affect predictions for graph neural networks in the node classification task. Using these experiments, we aim to answer the following three questions:

Q.1 *How useful are individual features generated from multi-step aggregation in graph neural networks?*

Q.2 *What is the effect of training the model over all the features, and what are the effects of different aggregator schemes?*

Q.3 *What is the impact of adding or removing features on the model's performance?*

Exploring these questions provides a deeper understanding of how GNN models can be designed to have better prediction capabilities.

3.1. Experiment setting

3.1.1. Model design

Conventionally, GNN models have feature propagation and transformation combined into a single layer, and the layers are stacked together. This step makes it difficult to distinguish the importance of the features and the role of MLP, and it becomes harder to analyze their impact on the prediction results of the model. For our experiments, we decouple the feature generation step and representation learning over features separately.

We use 2-layer neural network for our experiments. For the model design, instead of a single input channel, we propose to have all these features as input in parallel. Each feature is mapped to a separate linear layer. Hence the linear transformations are uniquely learned for all input features. In addition, we use L2-normalization to row-wise normalize the output of the linear layer. L2-normalization scales the node embedding vectors to lie on the "unit sphere". `ReLU` and `Dropout` are used for non-linear transformation of hidden features and regularization respectively. In the case of a single input feature matrix, hidden features are mapped to the final layer, and in the case of multiple input features, all hidden features of the node are aggregated and mapped to the final layer. We use two different aggregator schemes: `sum` and `concatenation`, and compare the results.

3.1.2. Input features

In our experiments, we consider node features of up to 3-hop neighbors (commonly used setting) in the graph. As we analyze both heterophily and homophily properties in a graph, we calculate both self-looped and no-loop features of the nodes. Hence, including node's own features, our feature set has total of 7 ($1+2\times3$) different features for the node $X_{feat} = \{X, AX, (A+I)X, A^2X, (A+I)^2X, A^3X, (A+I)^3X\}$. To explore the answers to the three questions defined earlier, we design three experiment settings: `Single_feature`, `All_feature` and `Sub_feature`. In `Single_feature` setting, we use only one out of seven features to train the model, and results are compared among all features. In this way, we analyze how informative each feature is for the label prediction of the nodes. In `All_feature` setting, we train the model on all features together and ascertain the model's performance. In addition, we use two aggregation schemes, i.e. sum and concatenation of hidden node features as they are commonly used in GNN models. Please note that `All_feature` with concatenate setting is similar to the SIGN model [29] as the model uses simple concatenation of features. In `Sub_feature` setting, we train the model on all possible combinations of the features in X_{feat} excluding the subsets already used in `Single_feature` and `All_feature`. Examples of such subset are $\{X, A^2X, (A+I)^3X\}$ and $\{X, AX\}$. Hence in this setting, we train the model on 119 different subsets of input features. Using all combinations of feature matrices, we perform an explicit selection of feature matrices. We report the result of the subset with the best accuracy. We use both aggregator schemes in this case too.

3.1.3. Datasets and hyperparameters

We run experiments on nine different datasets with varying homophily and heterophily properties. More details on datasets and preprocessing are provided in Section 6. For each input feature setting, we perform a search over 54 hyperparameter combinations of learning rate, weight decay and dropout.

3.2. Analysis of results

Table 1 shows the mean node classification accuracy with different input features for hidden dimension size of 64. We also include current state-of-the-art (SOTA) results for a comparison with our results. We find many interesting observations as follows:

Observation 1. We find that each hop features contribute differently to the prediction performance of the model. Some hop features are more informative than the others. For homophily datasets: Cora, Citeseer, and Pubmed self-looped features have higher node classification accuracy. In many recent publications that have considered heterophily, there is often more emphasis on Texas, Wisconsin and Cornell as good heterophily datasets, and Squirrel and Chameleon are considered to have low-quality node features as heterophily datasets [23]. However, we observe that for Wisconsin, Texas, Cornell and Actor, the best features are node's own features, and features of the neighbors are not informative enough. In case of Squirrel and Chameleon, we achieve best performance with node's first hop no-loop features. In these two datasets, node's own features and self-looped features have a low correlation with node's labels. Hence, they are, in fact, very good representation of heterophily datasets. These observations highlight the importance of using both self-looped and no-looped adjacency matrices in GNNs for better generalization over homophily and heterophily datasets respectively.

Based on our above observations, we postulate that *the problem of homophily and heterophily in graphs is a feature generation problem*. With appropriate feature generation measures, GNNs can learn on different types of graph datasets.

Observation 2. In `All_feature` setting, we train the model on all features with both concatenation and sum as aggregation operation. Our first observation is improved performance compared to `Single_feature`, which is natural as all features in combination provide more information. Comparing the aggregator schemes, for many datasets: Chameleon, Wisconsin, Texas, Cornell, and Squirrel sum operation has significantly lower accuracy compared to concatenation operation even with higher dimension embeddings (Please refer to Table 5 for additional results with d = 128 & 256). In this setting, we find concatenation operation overall provides better accuracy values compared with sum operation.

Observation 3. In `Sub_feature` setting, we find significant improvements in performance of the model on all datasets compared to both `Single_feature` setting and `All_feature` settings. This observation implies that among all features, there are subsets of features that are more informative than others for better prediction performance. In addition, other less informative features, if present in the input, can act as noise and may lead to worse performance of the model. This leads to the idea that feature selection is an important aspect of the design of graph neural networks. By reducing the effect of less informative/noisy features, higher prediction accuracy can be achieved even with a simple two-layered neural network. Over-smoothing problem in GNNs is considered to be due to node features becoming less informative because of feature averaging over many hops. However, with a feature selection mechanism, these uninformative features can be ignored. With these observations, we postulate that *graph learning and over-smoothing mitigation is a feature selection problem.* With a good feature selection strategy, GNN model can provide good prediction accuracy and eliminate the over-smoothing problem.

In addition, we find another interesting observation that in `Sub_feature` setting, the difference in the performance of concatenation and sum aggregation operation is reduced and is not as significant as observed with `All_feature` setting.

4. Proposed architecture

As discussed in Section 3.2 that feature selection is important to improve prediction capability of the model. However, using `Sub_Feature` is not feasible for real-world applications as the number of input feature combinations increase exponentially with an increase in the number of hops, making it computationally expensive. Nevertheless, a GNN model can be designed to approximate feature selection strategy. When all input features are provided, the model should be able learn to assign higher weights to more relevant and informative features and actively reject features that are not useful. To construct such a model, we propose to weight input features with

Table 1
Mean Classification Accuracy on fully-supervised node classification task on 2-layered MLP with hidden dimension size as 64. CAT and SUM refers to concatenation and sum aggregation operation respectively.

Dataset	Single_feature							All_feature		Sub_feature		SOTA
	X	AX	(A + I)X	A^2X	$(A + I)^2X$	A^3X	$(A + I)^3X$	CAT	SUM	CAT	SUM	
Cora	73.40	79.55	84.28	83.86	85.47	83.58	85.41	87.68	87.5	**88.10**	88.04	88.49 [4,33]
Citeseer	71.66	69.10	73.53	72.38	74.07	70.55	73.92	77.08	77.09	**77.52**	77.43	77.99 [34]
Pubmed	87.79	81.77	88.27	84.70	88.06	83.06	86.63	89.75	89.55	**89.88**	89.83	90.30 [4]
Chameleon	46.05	77.74	71.22	76.07	71.77	75.26	71.62	75.61	72.25	**78.59**	78.55	66.47 [33]
Wisconsin	87.45	63.13	58.03	62.54	52.94	60.00	51.76	85.09	79.8	87.84	**88.62**	86.98 [35]
Texas	85.40	66.21	61.35	67.29	58.64	62.43	58.10	84.32	78.91	88.64	**88.91**	86.49 [33]
Cornell	85.94	58.64	63.51	58.64	61.62	58.91	60.27	81.89	72.25	86.21	**86.75**	82.16 [23]
Squirrel	30.24	73.18	63.79	71.28	63.37	64.42	62.82	73.02	64.68	**74.16**	73.12	49.03 [33]
Actor	35.32	25.47	29.22	25.38	27.95	25.27	26.43	35.15	35.39	35.63	**35.67**	36.53 [35]

a single scalar value that is multiplied to each input feature matrix. We impose a constraint on these scalar values by the softmax function as follows. Let α_i be the scalar value for the ith feature matrix, then α_i scales the magnitude of the features as $\alpha_i X_i W_i^{(0)}$. Softmax function is used in deep learning as a non-linear normalizer, and its output is often practically interpreted as probabilities. Before training, the scalar values corresponding to each feature matrix are initialized with equal values, and softmax is applied on these values. The resultant normalized values α_i are then multiplied with the input features, and the concatenation operator is applied. Considering L number of input feature matrices X_l, $l \in \{1 .. L\}$, the formulation can be described as,

$$H^{(1)} = \Big\|_{l=1}^{L} \alpha_l X_l W_l^{(0)} \tag{5}$$

where $\sum_{l=1}^{L} \alpha_l = 1$ and $\|$ denotes concatenation operation.

While training, the scalar values of relevant features corresponding to the labels increase towards 1 while others decrease towards 0. The features that are not useful and represent more noise than signal have their magnitudes reduced with a corresponding decrease in their scalar values. Since we are not using a binary selection of features, we term this selection procedure as "soft-selection" of features.

The formulation discussed above can be understood in two ways. As GNNs have been represented with a polynomial filter,

$$g_\theta(P) = \sum_{k=0}^{K-1} \theta_k P^k \tag{6}$$

where $\theta \in \mathbb{R}^K$ is a vector of polynomial coefficients and P can be adjacency matrix [1,4,33], laplacian matrix [36] or PageRank based matrix [37]. As the polynomial coefficients are scalar parameters then our scheme can be considered as applying regularization on these parameters using the softmax function. The other way to look is to simply consider it as a weighting scheme. The input features can be arbitrarily chosen, and instead of a scalar weighting scheme, a more sophisticated scheme can be used.

For practical implementation since all weights are initialized as equal, they are all set to 1. After normalizing with softmax function, the individual scalar values become equal to $1/L$. During training, these values change, denoting the importance of the features. As the scalar values affect the magnitude of the features, they also affect the gradients propagated back to the linear layer, which transforms the input features. Hence it is important to have a unique weight matrix for each input feature matrix.

Feature selection graph neural network

Combining the model designs formulated earlier, we propose a simple and shallow (2-layered) graph GNN model called Feature Selection Graph Neural Network (FSGNN). Fig. 5 shows the diagrammatic representation of our model. Input features are precomputed using A_{sym} and $\tilde{A}_{sym}$ and transformed using a linear layer unique to each feature matrix. L2-normalization is applied on the output activations of the first layer and weighted with scalar weights regularized by the softmax function. Output features are then concatenated and non-linearly transformed using ReLU and mapped to the second linear layer. Cross-entropy loss is calculated with output logits of the second layer. The model can be represented as,

$$Z = \sigma\left(\text{CONCAT}_l(\alpha_l X_l W_l^{(0)})\right) W^{(1)} \tag{7}$$

where $\text{CONCAT}_l, \forall l \in \{1 .. L\}$, X_l are input features and σ is ReLU activation function.

5. Related work

GNNs have emerged as an indispensable tool to learn graph-centric data. Many prediction tasks like node classification, link prediction, graph classification, etc. [1,38] introduced a simple end-to-end training framework using approximations of spectral graph convolutions. Since then, there has been efforts in the research community to improve the performance of GNNs, and a variety of techniques have been introduced. Earlier GNN frameworks utilized a fixed propagation scheme along all edges, which is not always scalable for larger graphs. GraphSAGE [17] and FastGCN [18] introduce neighbor sampling approaches in graph neural networks. GAT [2] introduces the use of the attention mechanism to provide weights to features that are aggregated from the neighbors. APPNP [19], JK [39], Geom-GCN [34], SimP-GCN [20], and CPGNN [24] aim to improve the feature propagation scheme within layers of the model. More recently, researchers are proposing to make GNN models deeper [4,40,41]. However, deeper models suffer from over-smoothing, where after stacking many GNN layers, features of the node become indistinguishable from each other, and there is a drop in the performance of the model. DropEdge [22] proposes to drop a certain number of edges to reduce the speed of convergence of over-smoothing and relieves the information loss. GCNII [4] use residual connections and identity mapping in GNN layers to enable deeper networks. RevGNN [40] uses deep reversible architectures and [41] uses noise regularization to train deep GNN models.

Researchers find that traditional GNNs work well in homophily graphs but fail to generalize to heterophily graphs. Several models propose to handle heterophily properties in graph data: H2GCN [23], CPGNN [24], TDGNN [5], Geom-GCN [34], and GPRGNN [33]. However, a recent work [42] reveals that GCNs can achieve strong performance on heterophily graphs under certain conditions.

Similar to our work, the idea of decoupling feature generation and representation learning has been adopted by several existing works, such as SGC [21] and SIGN [29]. However, these models are equivalent to `Single_feature` and `All_feature` setting, thus suffer from similar drawbacks. Many GNN models exhibit similarity of weighting the features, however, many of them have fixed weighting scheme like JK-Net [39], APPNP [19], and GCNII [4]. In case of models with adaptable weighting scheme like ChebyNet [38] and GPR-GNN [33], learned weights do not show feature selection pattern due to lack of explicit regularization. Moreover, these GNN models do not use no-loop features by design, thus limiting their learning capability on heterophily datasets.

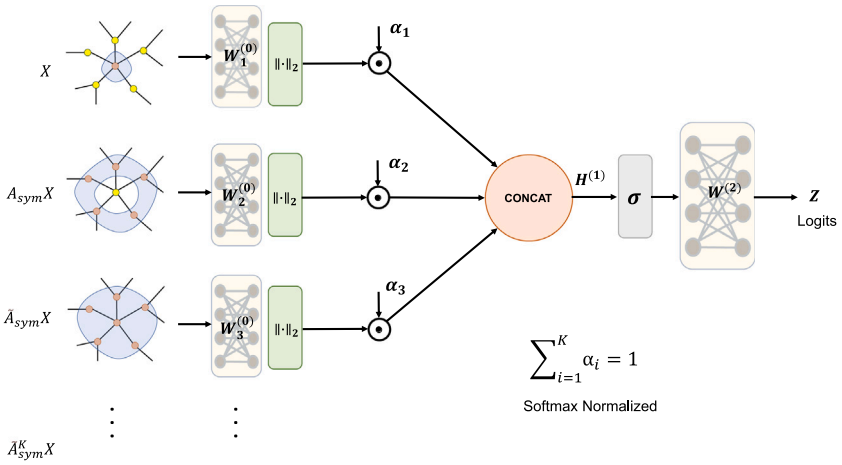

Fig. 5. Figure shows model diagram of FSGNN. Input features are generated based on powers of A and $\bar{A}$.

Table 2
Statistics of the node classification datasets.

Datasets	Hom. Ratio	Nodes	Edges	Features	Classes
Cora	0.81	2,708	5,429	1433	7
Citeseer	0.74	3,327	4,732	3703	6
Pubmed	0.80	19,717	44,338	500	3
Chameleon	0.23	2,277	36,101	2325	4
Wisconsin	0.21	251	499	1703	5
Texas	0.11	183	309	1703	5
Cornell	0.30	183	295	1703	5
Squirrel	0.22	5,201	198,353	2089	5
Actor	0.22	7,600	26,659	932	5

6. Experiments

In this section, we evaluate the empirical performance of our proposed model on real-world datasets on the node classification task and compare with other graph neural network models.

6.1. Datasets

For fully-supervised node classification tasks, we perform experiments on nine datasets commonly used in graph neural networks literature. Details of the datasets are presented in Table 2. Homophily ratio [23] denotes the fraction of edges which connects two nodes of the same label. Cora, Citeseer, and Pubmed [28] are citation networks based datasets and in general, are considered as homophily datasets. Graphs in Wisconsin, Cornell, Texas [34] represent links between webpages, Actor [43] represent actor co-occurrence in Wikipedia pages, Chameleon and Squirrel [44] represent the web pages in Wikipedia discussing corresponding topics. These datasets are considered as heterophily datasets. To provide a fair comparison on experimental results, we use publicly available data splits taken from [34].[1] These splits have been frequently used by researchers for experiments in their publications. The results of comparison methods presented in this paper are also based on this split.

6.2. Preprocessing

We follow the same preprocessing steps used by [4,34]. Other models to which we compare our results also follow the same set of procedures. Initial node features are row-normalized. To account for both homophily and heterophily, we use the adjacency matrix and adjacency matrix with added self-loops for feature transformation. Both matrices are symmetrically normalized. For efficient computation, adjacency matrices are stored and used as sparse matrices.

6.3. Settings and baselines

For a fully-supervised node classification task, each dataset is split evenly for each class into 48%, 32%, and 20% for training, validation, and testing [23,34]. We report the performance as mean classification accuracy over 10 random splits.

We fix the embedding size to 64 and set the initial learnable scalar parameter with respect to each hop to 1. Thus, the initial scalar value α_i is set to $1/L$. Hyper-parameter settings of the model for best performance are found by performing a grid-search over a range of hyper-parameters. We train the model under two input settings. In first setting, we follow the conventional classification of the datasets as homophily datasets and heterophily datasets. For homophily datasets, we use input features as node's self feature and self-looped aggregated features. For heterophily datasets, we use self-features and no-loop aggregated features. In the second setting, we use all features to train the model.

We compare our model to 10 different baselines and use the published results as the best performance of these models. GCNII [4] and H2GCN [23] have proposed multiple variants of their model. We have chosen the variant with the best performance on most datasets. GPRGNN uses random splits in their published results. For fair comparison, we ran their publicly available code on our standard splits while keeping other settings same. To get the best results, we performed hyperparameter search as mentioned in their code repository.

7. Results

7.1. Node classification results

Table 3 shows the comparison of the mean classification accuracy of our model and other popular GNN models. In general, traditional GNN models like GCN and GAT have higher performance on homophily datasets, however, they perform poorly on heterophily datasets. More recent models like H2GCN, WRGAT and GPRGNN perform relatively better on both homophily and heterophily datasets.

On heterophily datasets, our model shows significant improvements especially 51.1% on Squirrel and 18.8% on Chameleon dataset. Similarly, on Wisconsin, Texas, and Cornell, improvements are 1.6%, 1.2%, and 6.9%, respectively. On homophily datasets, we observe that different models perform best on different datasets. Our model still has consistent and comparable performance to SOTA.

[1] https://github.com/graphdml-uiuc-jlu/geom-gcn.

Table 3
Mean classification accuracy on fully-supervised node classification task. Results for GCN, GAT, GraphSAGE, Cheby+JK, MixHop and H2GCN-1 are taken from [23]. For GEOM-GCN, GCNII and WRGAT results are taken from the respective article. Best performance for each dataset is marked as bold and second best performance is underlined for comparison.

		Cora	Citeseer	Pubmed	Chameleon	Wisconsin	Texas	Cornell	Squirrel	Actor	Mean Acc.
GCN		87.28 ± 1.26	76.68 ± 1.64	87.38 ± 0.66	59.82 ± 2.58	59.80 ± 6.99	59.46 ± 5.25	57.03 ± 4.67	36.89 ± 1.34	30.26 ± 0.79	61.62
GAT		82.68 ± 1.80	75.46 ± 1.72	84.68 ± 0.44	54.69 ± 1.95	55.29 ± 8.71	58.38 ± 4.45	58.92 ± 3.32	30.62 ± 2.11	26.28 ± 1.73	58.55
GraphSAGE		86.90 ± 1.04	76.04 ± 1.30	88.45 ± 0.50	58.73 ± 1.68	81.18 ± 5.56	82.43 ± 6.14	75.95 ± 5.01	41.61 ± 0.74	34.23 ± 0.99	69.50
Cheby+JK		85.49 ± 1.27	74.98 ± 1.18	89.07 ± 0.30	63.79 ± 2.27	82.55 ± 4.57	78.38 ± 6.37	74.59 ± 7.87	45.03 ± 1.73	35.14 ± 1.37	69.89
MixHop		87.61 ± 0.85	76.26 ± 1.33	85.31 ± 0.61	60.50 ± 2.53	75.88 ± 4.90	77.84 ± 7.73	73.51 ± 6.34	43.80 ± 1.48	32.22 ± 2.34	68.10
GEOM-GCN		85.27	**77.99**	90.05	60.90	64.12	67.57	60.81	38.14	31.63	64.05
GCNII		88.01 ± 1.33	77.13 ± 1.38	**90.30 ± 0.37**	62.48 ± 2.74	81.57 ± 4.98	77.84 ± 5.64	76.49 ± 4.37	N/A	N/A	–
H2GCN-1		86.92 ± 1.37	77.07 ± 1.64	89.40 ± 0.34	57.11 ± 1.58	86.67 ± 4.69	84.86 ± 6.77	82.16 ± 4.80	36.42 ± 1.89	35.86 ± 1.03	70.71
WRGAT		88.20 ± 2.26	76.81 ± 1.89	88.52 ± 0.92	65.24 ± 0.87	86.98 ± 3.78	83.62 ± 5.50	81.62 ± 3.90	48.85 ± 0.78	**36.53 ± 0.77**	72.93
GPRGNN		**88.49 ± 0.95**	77.08 ± 1.63	88.99 ± 0.40	66.47 ± 2.47	85.88 ± 3.70	86.49 ± 4.83	81.89 ± 6.17	49.03 ± 1.28	36.04 ± 0.96	73.37
FSGNN (Homo/Hetero)	**3-hop**	87.61 ± 1.62	77.17 ± 1.48	89.70 ± 0.44	78.93 ± 1.03	88.24 ± 3.40	**87.57 ± 4.71**	87.30 ± 5.93	73.86 ± 1.81	35.38 ± 0.81	78.42
	8-hop	88.23 ± 1.17	77.35 ± 1.17	89.78 ± 0.38	**78.95 ± 0.86**	87.65 ± 3.51	**87.57 ± 4.86**	87.30 ± 4.53	73.94 ± 2.02	35.62 ± 0.87	**78.49**
FSGNN (All)	**3-hop**	87.73 ± 1.36	77.19 ± 1.35	89.73 ± 0.39	78.14 ± 1.25	**88.43 ± 3.22**	87.30 ± 5.55	87.03 ± 5.77	73.48 ± 2.13	35.67 ± 0.69	78.30
	8-hop	87.93 ± 1.00	77.40 ± 1.93	89.75 ± 0.39	78.27 ± 1.28	87.84 ± 3.37	87.30 ± 5.28	**87.84 ± 6.19**	**74.10 ± 1.89**	35.75 ± 0.96	78.46

Table 4
Ablation study over 1080 different hyperparameter settings.

	Cora	Citeseer	Pubmed	Chameleon	Wisconsin	Texas	Cornell	Squirrel	Actor
Proposed	83.68 ± 2.22	74.48 ± 1.44	**89.24 ± 0.27**	**72.48 ± 4.16**	81.48 ± 5.62	**78.80 ± 5.88**	**78.09 ± 2.22**	**63.57 ± 6.83**	33.54 ± 1.21
Without soft-selection	**87.07 ± 0.26**	**76.45 ± 0.27**	89.09 ± 0.39	72.27 ± 1.34	78.03 ± 6.55	76.28 ± 6.72	74.32 ± 6.54	61.73 ± 4.15	34.15 ± 0.64
Common weight ($W^{(0)}$)	83.19 ± 1.41	72.15 ± 1.02	88.96 ± 0.28	68.24 ± 6.03	70.56 ± 10.94	68.45 ± 7.65	68.18 ± 9.13	56.63 ± 8.54	32.73 ± 1.48
Without L2-normalization	77.12 ± 3.49	71.40 ± 10.01	87.72 ± 0.77	53.06 ± 6.18	**82.60 ± 2.68**	76.33 ± 3.87	76.18 ± 3.43	32.60 ± 6.38	**36.66 ± 0.55**

7.2. Comparison with `Sub_feature`

In our work, we aim to maintain performance as close as possible to `Sub_feature` (hidden dimension, d = 64) in Table 1. On five datasets: Cora, Chameleon, Cornell, Squirrel and Actor, our model performs as well as `Sub_feature`, however for other datasets, performance is comparable, albeit a bit lower. During the training, our model starts with all input features and learns to identify relevant features and reduce the effect of irrelevant features. However, it is difficult to completely reduce the effect of noisy/irrelevant features without an explicit forgetting scheme. We consider the development of such a scheme to completely remove the impact of noisy features in GNN training as the future direction of our work.

7.3. Ablation studies

In this section, we consider the effect of various proposed design strategies in Section 3.1.1 on the performance of the model. In general, graph neural networks are sensitive to the hyperparameters used in training and require some amount of tuning to get the best performance. Since each dataset may have a different set of best hyperparameters, it can be difficult to judge design decisions based just on best performance of the model with single hyperparameter setting. To provide a comprehensive evaluation, we compare the average accuracy of the model over 1080 combinations of the hyperparameters. The hyperparameters we tune are learning rate and weight decay of layers and dropout value applied as regularization between layers. We use four different settings: (1) Proposed model setting, (2) without softmax regularization on scalar weight parameters, (3) shared linear transformation layer on input features, and (4) without L2-normalization on input feature activations. Table 4 shows the average of classification accuracy values under various settings.

For most datasets, our proposed design schemes lead to better average accuracy. Cora and Citeseer show better average performance without softmax regularization; however, the peak performance is marginally less with regularization. Even though Wisconsin shows higher average accuracy without normalization, however, the best performance on the dataset was achieved with the normalization layer. We found that Actor was the only dataset where accuracy was reduced with the addition of the normalization layer. Without the normalization layer, our model achieves 37.63% accuracy. However, to maintain consistency, we do not include it in the main results. These variations also highlight that a single set of design choices may not apply to all datasets/tasks and some level of exploration is required.

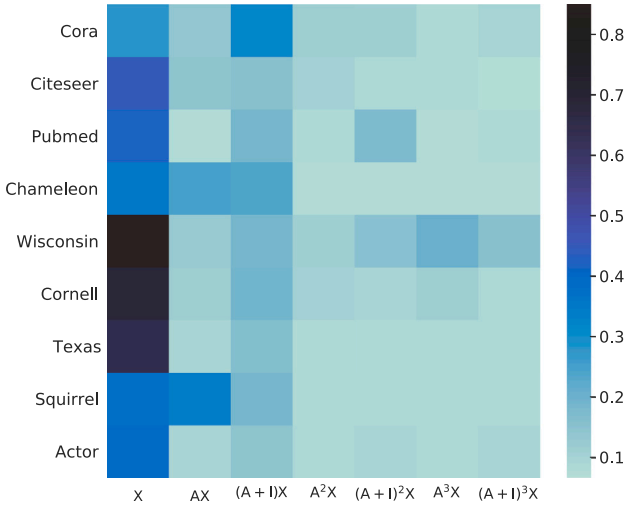

Fig. 6. Heatmap of average of learned soft-selection scalar for all datasets.

It is interesting to note that performance on almost all datasets is sensitive to the choice of the hyperparameters for training the model as there is a wide gap between best and average performance. One exception is Pubmed, where the model's performance is relatively unperturbed under various hyperparameter combinations.

7.4. Soft-selection parameter analysis

We analyze the learned soft-selection parameters on average over different model hyperparameter combinations. Fig. 6 shows average weights learned for different hop features for all datasets. For homophily datasets, it is easy to see that self-looped features are given more importance. Among heterophily datasets, Wisconsin, Cornell, Texas, and Actor have the most weights on node's self features. In these datasets, graph structure plays a limited role in the performance accuracy of the model. For Chameleon and Squirrel datasets, we observed

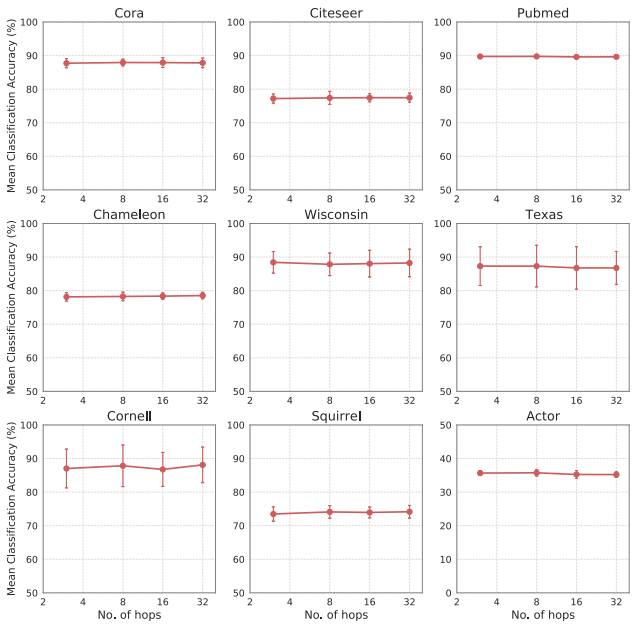

Fig. 7. Figure shows the effect on classification accuracy of FSGNN with increase in the number of hops of feature aggregation. x-axis is in logarithmic scale.

that the node's own features and first-hop features (without self-loop) were more useful for classification than any other features.

7.5. Over-smoothing analysis

Many GNN models suffer from the over-smoothing problem when the number of hops for feature aggregation is increased. In Section 3.2, we discussed how feature selection can be helpful to overcome over-smoothing problem. In this section, we evaluate the change in model's performance with increase in the hops for aggregation. We run additional experiments with hop values set to 16, and 32 with all features as input as described in Section 6. Fig. 7 shows the performance of the model for hop setting of 3,8,16 and 32. We observe that there is little variation in the performance of the model on various datasets and the model does not suffer from over-smoothing. This result is intuitive as aggregated features from higher hops are not very useful, and the model can learn to place low weights on them. For few datasets, we were able to achieve higher accuracy values: Citeseer (77.46%), Cornell (88.11%) and Squirrel (74.15%).

8. Conclusion

In this work, we explore the importance of feature selection in GNN training. We run extensive experiments to investigate GNN model design requirements for homophily and heterophily datasets and how feature selection can lead to higher prediction accuracy on benchmark datasets. Our experimental observations provide a definite confirmation that feature selection is a good direction for the exploration of GNN architectures. Based on our experimental observations, we propose a novel GNN model called FSGNN. Using extensive experiments, we show that simple 2-layered FSGNN outperforms the current state-of-the-art GNN models with complex designs on the node classification task. Analysis of the learned parameters provides us the crucial information of feature importance. In addition, we show that even shallow models can learn and provide high prediction accuracy, and with our model over-smoothing phenomenon can be easily avoided.

CRediT authorship contribution statement

Sunil Kumar Maurya: Conception and design of study, Acquisition of data, Analysis and/or interpretation of data, Writing – original draft, Writing – review & editing. **Xin Liu:** Conception and design of study, Analysis and/or interpretation of data, Writing – original draft, Writing – review & editing. **Tsuyoshi Murata:** Conception and design of study, Writing – original draft.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was supported by JSPS Grant-in-Aid for Scientific Research, Japan (Grant Number 21K12042, 17H01785), JST CREST, Japan (Grant Number JPMJCR1687), and the New Energy and Industrial Technology Development Organization, Japan (Grant Number JPNP20006).

All authors approved the version of the manuscript to be published.

Appendix A. Extended experiment results

A.1. Node classification on higher dimensions

Table 5 shows the mean node classification accuracy under `Single_Feature`, `All_Feature` and `Sub_Feature` settings with hidden dimensions set to 64, 128 and 256. We observe that just with increase in hidden dimensions of 2-layered MLP, we approach classification accuracy values similar or higher than state-of-the-art more complex and/or deeper GNN models.

A.2. Experiments with no non-linear activation

As we observe in Table 5, with `Sub_Feature` scheme enabling feature selection, a simple 2-layered MLP already trains to very high accuracy. However, we would like to further understand the requirements of GNN complexity for the given datasets. In this section, we compare the effect of non-linear activation ReLU in 2-layered MLP model under `Sub_Feature` setting with hidden dimensions set to 256. We remove the ReLU unit between the two layers and keep other settings same for learning rate, weight decay and dropout.

Table 6 shows the accuracy comparisons of the models with and without non-linear activation between layers. Comparing both settings, we find two interesting observations. First, for Cora, Texas and Cornell datasets, we see further improvement in accuracy values. Second, for other datasets while accuracy values have decreased (expectedly), the difference is not significant except for Chameleon and Squirrel dataset.

With these results, we infer that for these datasets, simple graph convolution operation over node features combined with hop-feature selection provides sufficient information. Thus enabling a simple 2-layered model to perform well on the node classification task.

Appendix B. FSGNN scalability

In this paper, we have analyzed importance of feature selection by separating feature aggregation process with neural network component.

Table 5
Mean Classification Accuracy on fully-supervised node classification task with hidden dimensions set to 64, 128 & 256.

Dataset	#dimensions	Single_Feature							All_feature		Sub_feature		SOTA
		X	AX	(A + I)X	A^2X	$(A+I)^2X$	A^3X	$(A+I)^3X$	CAT	SUM	CAT	SUM	
Cora	d = 64	73.40	79.55	84.28	83.86	85.47	83.58	85.41	87.68	87.5	88.10	88.04	88.49 [4,33]
	d = 128	73.84	79.93	84.56	85.85	86.94	85.05	86.23	87.76	87.92	88.19	**88.43**	
	d = 256	74.06	82.25	85.97	86.27	87.54	85.67	86.86	87.7	87.68	88.09	88.41	
Citeseer	d = 64	71.66	69.10	73.53	72.38	74.07	70.55	73.92	77.08	77.09	77.52	77.43	77.99 [34]
	d = 128	71.94	70.74	73.96	73.70	74.58	71.42	74.28	77.35	77.04	77.70	77.63	
	d = 256	72.54	71.80	76.67	75.02	76.53	72.77	75.36	77.35	77.11	**77.86**	77.74	
Pubmed	d = 64	87.79	81.77	88.27	84.70	88.06	83.06	86.63	89.75	89.55	89.88	89.83	90.30 [4]
	d = 128	87.93	81.90	88.26	84.84	88.09	83.01	86.66	89.82	89.58	89.92	89.86	
	d = 256	88.01	81.89	88.31	84.86	88.08	83.02	86.74	89.81	89.64	**89.97**	89.89	
Chameleon	d = 64	46.05	77.74	71.22	76.07	71.77	75.26	71.62	75.61	72.25	78.59	78.55	66.47 [33]
	d = 128	46.07	77.74	71.40	76.11	71.42	76.07	71.86	75.76	71.4	78.99	77.98	
	d = 256	46.09	77.63	71.25	76.77	71.07	76.2	72.58	76.77	70.81	**79.01**	77.63	
Wisconsin	d = 64	87.45	63.13	58.03	62.54	52.94	60.00	51.76	85.09	79.8	87.84	88.62	86.98 [35]
	d = 128	88.03	62.54	57.84	62.15	52.35	58.82	51.17	85.29	82.94	88.43	88.04	
	d = 256	88.03	62.54	58.03	61.96	51.76	57.84	50.78	87.45	83.92	89.02	**89.22**	
Texas	d = 64	85.40	66.21	61.35	67.29	58.64	62.43	58.10	84.32	78.91	88.64	88.91	86.49 [33]
	d = 128	86.21	67.02	61.62	67.56	58.64	61.62	57.83	84.32	78.91	88.38	88.11	
	d = 256	85.94	67.83	61.08	67.29	58.91	61.35	58.10	86.48	82.92	**88.65**	**88.65**	
Cornell	d = 64	85.94	58.64	63.51	58.64	61.62	58.91	60.27	81.89	72.25	86.21	86.75	82.16 [23]
	d = 128	86.21	58.10	63.78	58.64	60.54	58.91	60.27	84.05	74.86	87.56	87.57	
	d = 256	87.83	58.64	65.40	58.64	61.08	58.91	60.54	85.13	77.29	**88.11**	87.57	
Squirrel	d = 64	30.24	73.18	63.79	71.28	63.37	64.42	62.82	73.02	64.68	74.16	73.12	49.03 [33]
	d = 128	30.30	72.83	63.66	71.49	64.43	64.49	63.59	72.55	62.50	73.87	72.78	
	d = 256	30.66	72.54	63.28	71.91	65.36	65.24	63.77	72.63	59.88	**74.49**	72.76	
Actor	d = 64	35.32	25.47	29.22	25.38	27.95	25.27	26.43	35.15	35.39	35.63	35.67	36.53 [35]
	d = 128	35.75	25.38	29.26	25.25	27.71	25.26	26.21	35.94	35.57	35.96	36.05	
	d = 256	36.08	25.41	29.28	25.23	27.53	25.29	26.15	36.10	35.60	36.22	**36.31**	

Table 6
Mean node classification accuracy under `Sub_Feature` setting with and without using ReLU activation and hidden dimensions set to 256.

Dataset	Sub_Feature (With ReLU)		Sub_Feature (No ReLU)		SOTA
	CAT	SUM	CAT	SUM	
Cora	88.09	88.41	88.49	**88.51**	88.49
Citeseer	**77.86**	77.74	77.72	77.81	77.99
Pubmed	**89.97**	89.89	89.52	89.54	90.30
Chameleon	**79.01**	77.63	76.95	76.69	66.47
Wisconsin	89.02	**89.22**	88.63	88.82	86.98
Texas	88.65	88.65	**89.73**	88.38	86.49
Cornell	88.11	87.57	**88.38**	87.84	82.16
Squirrel	**74.49**	72.76	70.45	69.94	49.03
Actor	36.22	**36.31**	36.11	35.8	36.53

Table 7
Mean classification accuracy on ogbn-100M dataset. Best performance is marked bold and second best performance is underlined.

Method	Accuracy
SGC	63.29 ± 0.19
Node2Vec	58.07 ± 0.28
SIGN	65.11 ± 0.14
SAGN	66.75 ± 0.84
GAMLP	<u>67.71 ± 0.02</u>
FSGNN	**68.07 ± 0.06**

This setting provides an additional benefit of making the model scalable. Many GNN models by design are not scalable for large graph datasets with millions of nodes as they are limited by the amount of GPU memory available in the system. To demonstrate the scalability of our model, we run experiments on `ogbn-papers100M` dataset [45,46], which is the largest public graph learning benchmark with about 111 million nodes, 1.6 billion edges and 172 node label classes. Similar to our previous experimental settings, we generate a set of features with A and $\tilde{A}$ for 4-hop aggregation. The dimension of the hidden layer is set to 1280 and γ is set to L = 9 (equal to number of input features) to provide a stable training. The model is trained batchwise with input features for 10 random initializations, and we report mean accuracy.

We compare the accuracy of our model with SGC [21], Node2Vec [47], SIGN [29], SAGN (base-model) [48] and GAMLP (base-model) [49]. Similar to our method, input features can be precomputed in many of these models, thus making them scalable for larger datasets. Once features are computed, the model can be trained with small input batches of node features on the GPU. Many other GNN models cannot be trained for larger graphs as the feature generation, and model training are combined.

Table 7 shows the mean node classification accuracy of our model along with published results of other methods taken from [29,46,48,49]. As seen from the results, FSGNN outperforms all other models. We would like to point out that classification accuracy of our model in addition to other models may be increased further by utilizing additional augmentation schemes or training measures [48–52]. However, as this is out of scope of our paper, hence we do not discuss it here.

Appendix C. Implementation details of FSGNN

For reproducibility of experimental results, we provide the details of our experiment setup and hyperparameters of the model.

We use PyTorch 1.6.0 as deep learning framework on Python 3.8. Model training is done on Nvidia V100 GPU with 16 GB graphics memory and CUDA version 10.2.89.

For node classification results (3), we do grid search for learning rate and weight decay of the layers and dropout between the layers. Hyperparameters are set for first layer $fc1$, second layer $fc2$ and scalar weight parameter sca. ReLU is used as non-linear activation and Adam is used as the optimizer. Table 8 shows details of hyperparameter search space. Tables 9 and 10 show the best hyperparameters for the model in 3-hop and 8-hop configuration respectively. Patience value 100 is used for all datasets.

For experiments on ogbn-papers100M dataset, based on the data from earlier experiments, we manually tuned the hyperparameters to

Fig. 8. Figure shows t-SNE embedding plots of the node features of Cora dataset. Each node belongs to one of the seven classes represented by different colors. (a) Plot of initial node features, (b) Plot of node embeddings learned after training FSGNN model on the dataset.

Table 8
Hyperparameter search space.

Hyperparameter	Values
WD_{sca}	0.0, 0.0001, 0.001, 0.01, 0.1
LR_{sca}	0.04, 0.02, 0.01, 0.005
WD_{fc1}	0.0, 0.0001, 0.001
WD_{fc2}	0.0, 0.0001, 0.001
LR_{fc}	0.01, 0.005
Dropout	0.5, 0.6, 0.7

Table 9
Hyperparameters of the 3-hop model.

Datasets	WD_{sca}	LR_{sca}	WD_{fc1}	WD_{fc2}	LR_{fc}	*Dropout*
Cora	0.1	0.01	0.001	0.0001	0.01	0.6
Citeseer	0.0001	0.005	0.001	0.0	0.01	0.5
Pubmed	0.01	0.005	0.0001	0.0001	0.01	0.7
Chameleon	0.1	0.005	0.0	0.0	0.005	0.5
Wisconsin	0.0001	0.01	0.001	0.0001	0.01	0.5
Texas	0.001	0.01	0.001	0.0	0.01	0.7
Cornell	0.0	0.01	0.001	0.001	0.01	0.5
Squirrel	0.1	0.04	0.0	0.001	0.01	0.7
Actor	0.0	0.04	0.001	0.0001	0.01	0.7

Table 10
Hyperparameters of the 8-hop model.

Datasets	WD_{sca}	LR_{sca}	WD_{fc1}	WD_{fc2}	LR_{fc}	*Dropout*
Cora	0.1	0.02	0.001	0.0001	0.01	0.6
Citeseer	0.0001	0.01	0.001	0.0001	0.01	0.5
Pubmed	0.01	0.02	0.0001	0.0	0.005	0.7
Chameleon	0.1	0.01	0.0	0.0	0.005	0.5
Wisconsin	0.001	0.02	0.001	0.0001	0.01	0.5
Texas	0.01	0.01	0.001	0.0	0.01	0.7
Cornell	0.0	0.01	0.001	0.0001	0.01	0.5
Squirrel	0.1	0.02	0.0	0.0001	0.01	0.5
Actor	0.0001	0.04	0.001	0.0001	0.01	0.7

Table 11
Hyperparameters for the ogbn-paper100M dataset.

Hyperparameters	Values
WD1, WD2, WD3, WDsca	1e−05, 1e−06, 9e−06, 0.1
LR1, LR2, LR3, LR_sca	5e−05, 2e−04, 2e−05, 1e−04
Dropout1, Dropout2	0.5, 0.6

get the best accuracy result. Batch size of 4096 was used for training data and maximum training epoch was set to 1500. Table 11 shows the relevant hyperparameters for the model. Patience value 300 was used in model training.

Appendix D. Cora node embedding comparison: Original vs. learned with GNN

Cora dataset has 2708 nodes with 7 classes. Fig. 8 shows plots of embeddings with initial node features and node features learned after training FSGNN model. For initial node features (on left), we observe that distance between intra-class nodes and inter-class nodes is not significant. On the other hand, for learned node features, the inter-class distance is significantly higher and nodes of same class form higher quality clusters. Thus, the model can predict unknown labels of the nodes with higher accuracy.

References

[1] T.N. Kipf, M. Welling, Semi-supervised classification with graph convolutional networks, ICLR (2017).
[2] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Liò, Y. Bengio, Graph attention networks, ICLR (2018).
[3] S. Abu-El-Haija, B. Perozzi, A. Kapoor, H. Harutyunyan, N. Alipourfard, K. Lerman, G.V. Steeg, A. Galstyan, MixHop: Higher-order graph convolutional architectures via sparsified neighborhood mixing, in: ICML, 2019.
[4] M. Chen, Z. Wei, Z. Huang, B. Ding, Y. Li, Simple and deep graph convolutional networks, ICML (2020).
[5] Y. Wang, T. Derr, Tree decomposed graph neural network, CIKM (2021).
[6] R. Ying, R. He, K. Chen, P. Eksombatchai, W.L. Hamilton, J. Leskovec, Graph convolutional neural networks for web-scale recommender systems, in: KDD '18, 2018.
[7] R. van den Berg, T. Kipf, M. Welling, Graph convolutional matrix completion, 2017, ArXiv abs/1706.02263.
[8] I. Chami, R. Ying, C. Ré, J. Leskovec, Hyperbolic graph convolutional neural networks, NeurIPS (2019).
[9] R. Ying, J. You, C. Morris, X. Ren, W.L. Hamilton, J. Leskovec, Hierarchical graph representation learning with differentiable pooling, in: NIPS'18, 2018, pp. 4805–4815.
[10] M. Zhang, Z. Cui, M. Neumann, Y. Chen, An end-to-end deep learning architecture for graph classification, in: AAAI, 2018.
[11] J. Gilmer, S.S. Schoenholz, P.F. Riley, O. Vinyals, G.E. Dahl, Neural message passing for quantum chemistry, in: Proceedings of the 34th International Conference on Machine Learning - Volume 70, in: ICML'17, 2017.
[12] K. Madhawa, K. Ishiguro, K. Nakago, M. Abe, GraphNVP: AN invertible flow model for generating molecular graphs, 2019, arXiv:1905.11600.
[13] S.K. Maurya, X. Liu, T. Murata, Graph neural networks for fast node ranking approximation, ACM Trans. Knowl. Discov. Data 15 (5) (2021) 78:1–78:32.
[14] C. Fan, L. Zeng, Y. Ding, M. Chen, Y. Sun, Z. Liu, Learning to identify high betweenness centrality nodes from scratch: A novel graph neural network approach, in: Proceedings of the 28th ACM International Conference on Information and Knowledge Management, in: CIKM '19, 2019, pp. 559–568.
[15] D. Marcheggiani, I. Titov, Encoding Sentences with Graph Convolutional Networks for Semantic Role Labeling, ACL, 2017, pp. 1506–1515.
[16] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, P.S. Yu, A comprehensive survey on graph neural networks, 2019, arXiv:1901.00596.
[17] W. Hamilton, Z. Ying, J. Leskovec, Inductive representation learning on large graphs, in: NIPS, 2017, pp. 1024–1034.

[18] J. Chen, T. Ma, C. Xiao, FastGCN: FAst learning with graph convolutional networks via importance sampling, in: ICLR, 2018.
[19] J. Klicpera, A. Bojchevski, S. Günnemann, Predict then Propagate: Combining neural networks with personalized pagerank for classification on graphs, 2018.
[20] W. Jin, T. Derr, Y. Wang, Y. Ma, Z. Liu, J. Tang, Node similarity preserving graph convolutional networks, in: WSDM '21, Association for Computing Machinery, 2021, pp. 148–156, http://dx.doi.org/10.1145/3437963.3441735.
[21] F. Wu, A.H. Souza, T. Zhang, C. Fifty, T. Yu, K.Q. Weinberger, Simplifying graph convolutional networks, in: ICML, 2019.
[22] Y. Rong, W. Huang, T. Xu, J. Huang, DropEdge: Towards deep graph convolutional networks on node classification, in: ICLR, 2020.
[23] J. Zhu, Y. Yan, L. Zhao, M. Heimann, L. Akoglu, D. Koutra, Beyond homophily in graph neural networks: Current limitations and effective designs, NeurIPS 33 (2020).
[24] J. Zhu, R.A. Rossi, A.B. Rao, T. Mai, N. Lipka, N. Ahmed, D. Koutra, Graph neural networks with heterophily, in: AAAI, 2021.
[25] D. Bo, X. Wang, C. Shi, H. Shen, Beyond low-frequency information in graph convolutional networks, in: AAAI, 2021.
[26] S. Bhagat, G. Cormode, S. Muthukrishnan, Node classification in social networks, in: C.C. Aggarwal (Ed.), Social Network Data Analytics, Springer US, Boston, MA, 2011, pp. 115–148.
[27] M. McPherson, L. Smith-Lovin, J. Cook, Birds of a feather: Homophily in social networks, 2001.
[28] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Gallagher, T. Eliassi-Rad, Collective classification in network data, AI Mag. (2008).
[29] F. Frasca, E. Rossi, D. Eynard, B. Chamberlain, M. Bronstein, F. Monti, SIGN: Scalable inception graph neural networks, 2020, arXiv:2004.11198.
[30] J. Tang, S. Alelyani, H. Liu, Feature selection for classification: A review, Data Classif.: Algorithms Appl. (2014) 37–64.
[31] J. Li, K. Cheng, S. Wang, F. Morstatter, R.P. Trevino, J. Tang, H. Liu, Feature selection: A data perspective, ACM Comput. Surv. 50 (6) (2017) 94:1–94:45.
[32] G. Chandrashekar, F. Sahin, A survey on feature selection methods, 40th-year commemorative issue, Comput. Electr. Eng. 40 (1) (2014) 16–28.
[33] E. Chien, J. Peng, P. Li, O. Milenkovic, Adaptive universal generalized PageRank graph neural network, ICLR (2021).
[34] H. Pei, B. Wei, K. Chang, Y. Lei, B. Yang, Geom-GCN: Geometric graph convolutional networks, ICLR (2020).
[35] S. Suresh, V. Budde, J. Neville, P. Li, J. Ma, Breaking the limit of graph neural networks by improving the assortativity of graphs with local mixing patterns, 2021, arXiv:2106.06586.
[36] H. NT, T. Maehara, T. Murata, Stacked graph filter, 2020, arXiv:2011.10988.
[37] D. Berberidis, A.N. Nikolakopoulos, G.B. Giannakis, Adaptive diffusions for scalable learning over graphs, IEEE Trans. Signal Process. 67 (5) (2019) 1307–1321.
[38] M. Defferrard, X. Bresson, P. Vandergheynst, Convolutional neural networks on graphs with fast localized spectral filtering, 2016, arXiv:1606.09375.
[39] K. Xu, C. Li, Y. Tian, T. Sonobe, K.-i. Kawarabayashi, S. Jegelka, Representation Learning on Graphs with Jumping Knowledge Networks, PMLR, 2018.
[40] G. Li, M. Müller, B. Ghanem, V. Koltun, Training graph neural networks with 1000 layers, in: ICML, 2021.
[41] J. Godwin, M. Schaarschmidt, A. Gaunt, A. Sanchez-Gonzalez, Y. Rubanova, P. Veliković, J. Kirkpatrick, P. Battaglia, Very deep graph neural networks via noise regularisation, 2021, arXiv:2106.07971.
[42] Y. Ma, X. Liu, N. Shah, J. Tang, Is homophily a necessity for graph neural networks? 2021, arXiv:2106.06134.
[43] J. Tang, J. Sun, C. Wang, Z. Yang, Social influence analysis in large-scale networks, in: KDD '09, Association for Computing Machinery, 2009, pp. 807–816.
[44] B. Rozemberczki, C. Allen, R. Sarkar, Multi-scale attributed node embedding, 2020, arXiv:1909.13021.
[45] K. Wang, Z. Shen, C. Huang, C.-H. Wu, Y. Dong, A. Kanakia, Microsoft academic graph: When experts are not enough, Quant. Sci. Stud. 1 (1) (2020) 396–413.
[46] W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, J. Leskovec, Open graph benchmark: Datasets for machine learning on graphs, 2021, arXiv:2005.00687.
[47] A. Grover, J. Leskovec, Node2vec: Scalable feature learning for networks, KDD (2016).
[48] C. Sun, H. Gu, J. Hu, Scalable and adaptive graph neural networks with self-label-enhanced training, 2021, arXiv:2104.09376.
[49] W. Zhang, Z. Yin, Z. Sheng, W. Ouyang, X. Li, Y. Tao, Z. Yang, B. Cui, Graph attention multi-layer perceptron, 2021, arXiv:2108.10097.
[50] Q. Li, Z. Han, X.-M. Wu, Deeper insights into graph convolutional networks for semi-supervised learning, 2018, arXiv:1801.07606.
[51] K. Sun, Z. Lin, Z. Zhu, Multi-stage self-supervised learning for graph convolutional networks on graphs with few labels, 2020, arXiv:1902.11038.
[52] Q. Huang, H. He, A. Singh, S.-N. Lim, A.R. Benson, Combining label propagation and simple models out-performs graph neural networks, 2020, arXiv:2010.13993.

Sunil Kumar Maurya is currently Ph.D. candidate at department of Computer Science of Tokyo Institute of Technology, Japan. He received his B.Tech in Electrical Engineering from National Institute of Technology Silchar, India and his M.E. in Artificial Intelligence from Tokyo Institute of Technology, Japan. His main research interests topics are network science, machine learning and deep learning.

Xin Liu is a senior researcher at the Artificial Intelligence Research Center (AIRC) of the National Institute of Advanced Industrial Science and Technology (AIST). He received Ph.D. and M.Sc in Computer Science from Tokyo Institute of Technology and Wuhan University, respectively. His main research interests are graph & network analytics, data mining, machine learning, and deep learning.

Tsuyoshi Murata is a professor of the department of computer science, school of computing, Tokyo Institute of Technology. He has been doing research on artificial intelligence, especially network science, machine learning and Web mining.