DEMYSTIFYING GRAPH NEURAL NETWORK VIA GRAPH FILTER ASSESSMENT

Anonymous authors
Paper under double-blind review

ABSTRACT

Graph Neural Networks (GNNs) have received tremendous attention recently due to their power in handling graph data for different downstream tasks across different application domains. The key of GNN is its graph convolutional filters, and recently various kinds of filters are designed. However, there still lacks in-depth analysis on (1) Whether there exists a best filter that can perform best on all graph data; (2) Which graph properties will influence the optimal choice of graph filter; (3) How to design appropriate filter adaptive to the graph data. In this paper, we focus on addressing the above three questions. We first propose a novel assessment tool to evaluate the effectiveness of graph convolutional filters for a given graph. Using the assessment tool, we find out that there is no single filter as a 'silver bullet' that perform the best on all possible graphs. In addition, different graph structure properties will influence the optimal graph convolutional filter's design choice. Based on these findings, we develop Adaptive Filter Graph Neural Network (AFGNN), a simple but powerful model that can adaptively learn task-specific filter. For a given graph, it leverages graph filter assessment as regularization and learns to combine from a set of base filters. Experiments on both synthetic and real-world benchmark datasets demonstrate that our proposed model can indeed learn an appropriate filter and perform well on graph tasks.

1 INTRODUCTION

Graph Neural Network (GNN) is a powerful tool for representation learning on graph data, which has received increasing attention over the past several years. GNN can obtain informative node representations for a graph of arbitrary size and attributes, and has shown great effectiveness in graph-related down-stream applications, such as semi-supervised node classification (Kipf & Welling, 2017), graph classification (Wu et al., 2019b), recommendation systems (Ying et al., 2018), and knowledge graphs (Schlichtkrull et al., 2018).

As GNN's superior performance in graph-related tasks, a question is naturally raised: what makes GNN so powerful? Noticing that GNN adopts the concept of the convolution operation into graph domain. To obtain the representation of a specific node, GNN aggregates its neighbors' representations with a convolutional filter. For a task that is correlated with graph topology, the convolutional filter can help GNNs get better task-specific representation (Xu et al., 2019). Therefore, it is the filter that makes GNN that powerful, and thus the key of GNN is to design proper graph convolutional filters that can best leverage the graph topology for a given graph data.

Recently we've seen many GNN architecture designs (Kipf & Welling, 2017; Velickovic et al., 2018; Xu et al., 2019; Wu et al., 2019a; Maehara, 2019), with their own graph filter design. However, none of them have properly answered the following fundamental questions of GNN: *(1) Is there a best filter that works for all graphs? (2) If not, what are the properties of graph structure that will influence the performance of graph convolutional filters? (3) Can we design an algorithm to adaptively find the appropriate filter for a given graph?*

In this paper, Inspired by studies in Linear Discriminant Analysis (LDA), we propose the *Graph Filter Discriminant Score*(GFD), which measures the power of a graph convolutional filter in discriminating node representations of different classes on a specific graph. We analyzed all the existing GNN's filters with this assessment method to answer those three aforementioned questions. We found out that no single filter designs can achieve optimal results on all possible graphs. In other words, for different graph structures, we should design different graph convolutional filters to achieve optimal performance. We then analyze experimentally and theoretically how the properties of graph structures influence the optimal choice of graph convolutional filters.

Based on all of our findings, we propose the Adaptive Filter Graph Neural Network (AF-GNN), which can adaptively learn a proper model for the given graph. We use the *Graph Filter Discriminant Score* as a regularization to guide the network to learn a good combination of the task-specific filter from a set of bases. We show that the proposed Adaptive Filter can better capture graph topology and separate features on both real-world benchmark datasets and synthetic datasets.

We highlight our main contributions as follows:

- We propose an assessment tool to analyze the effectiveness of graph convolutional filters. Using this tool, we find out that no best filter can work for all graphs, and the optimal choice of graph convolutional filter depends on graph data.
- We propose Adaptive Filter Graph Neural Network (AF-GNN) that can adaptively learn a proper filter for a specific graph using the Graph Filter Discriminant Score as guidance.
- We demonstrate that the proposed model can find better filters and achieve better performance than existing GNNs, on both real-word and newly created benchmark datasets.

2 PRELIMINARIES

Semi-Supervised Node Classification. Let $\boldsymbol{Y}$ be the class assignment vector for all the nodes in $\mathbb{V}$, where C indicates the total number of classes and $\boldsymbol{Y}_v \in \{1, \cdots, C\}$ indicates the class that node v belongs to. The goal of semi-supervised node classification is to learn a mapping function $f : \mathbb{V} \to \{1, \cdots, C\}$ using the labeled nodes, and predict the class labels for the unlabeled ones, i.e., $\hat{\boldsymbol{Y}}_v = f(v)$, by leveraging both node features $\boldsymbol{X}$ and graph structure $\boldsymbol{A}$.

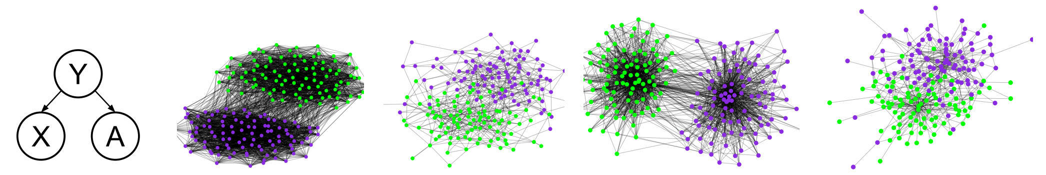
Graph Data Generator. Intuitively, semi-supervised node classification requires both node features ($\boldsymbol{X}$) and graph structure ($\boldsymbol{A}$) to be correlated to the intrinsic node labels ($\boldsymbol{Y}$) to some extent. To systematically analyze the performance of different GNN filters, we'd like to test its performance under different graph data with different properties, i.e., different $\boldsymbol{X}$, $\boldsymbol{A}$, $\boldsymbol{Y}$. However, there are not so much graph data in real-world to support a thorough analysis. Thus, we study some simulated data instead.

A simple graphical model of generator logic is illustrated in Fig. 1(a). We generate the graph data by generating both nodes attributes $\boldsymbol{X}$ and graph structure $\boldsymbol{A}$ conditioned on label $\boldsymbol{Y}$. We now describe the generation of $\boldsymbol{Y}$, $\boldsymbol{X}|\boldsymbol{Y}$ and $\boldsymbol{A}|\boldsymbol{Y}$, respectively.

Generating $\boldsymbol{Y}$: Each node is randomly assigned with a class label. We assume each class c is associated with n_c nodes which are indexed based on their class groups.

Generating $\boldsymbol{X}|\boldsymbol{Y}$: We assume the node feature are sampled from a distribution determined by its corresponding label. For example, we can sample node features of class c by a Gaussian distributions with the parameters conditioned on class c: $\boldsymbol{X}^{(c)} \sim \mathcal{N}(\boldsymbol{\mu}^{(c)}, \boldsymbol{\Sigma}^{(c)})$. In our experiment, we choose different distributions as $\boldsymbol{X}$, such as circles, Gaussian, etc.

Generating $\boldsymbol{A}|\boldsymbol{Y}$: We follow the most classic class-aware graph generator, i.e. stochastic block model (SBM, Holland et al. (1983)), to generate graph structure conditional on class labels. SBM has several simple assumptions that (1) edges are generated via Bernoulli distributions independently and (2) the parameter of Bernoulli distribution is determined by the classes of the corresponding pair of nodes v_i and v_j, i.e., $A_{ij}|\boldsymbol{Y}_i, \boldsymbol{Y}_j \sim Ber(p_{\boldsymbol{Y}_i \boldsymbol{Y}_j})$, where $p_{\boldsymbol{Y}_i \boldsymbol{Y}_j}$ is a parameter determined by the two corresponding classes. In a simple two-class case, $p = p_{11} = p_{22}$ denotes the probability that linked pair belongs to the same class, while $q = p_{12} = p_{21}$ denotes the probability that linked pair belongs to different classes. We call $\frac{p+q}{2}$ the *"density of graph"*, which controls the overall connectivity, and we call $|p/q|$ the *"density gap"*, which controls how closely the graph generated by SBM correlates with labels. We assume $p \geq q$ in all the following sections. Degree Corrected SBM (DCSBM, Karrer & Newman (2011)), which is a variation of SBM, can use another parameter γ to control the *"power-law coefficient"* of degree distribution among nodes. Figure 1(b-e) demonstrates examples of synthetic graphs generated by SBM and DCSBM with different graph structure properties.



(a) Dependency of $\boldsymbol{Y}$, $\boldsymbol{X}$ and $\boldsymbol{A}$ (b) SBM: $p = 0.3$, $q = 0.05$ (c) SBM: $p = 0.1$, $q = 0.02$ (d) DCSBM: $p = 0.3$, $q = 0.05, \gamma = -0.9$ (e) DCSBM: $p = 0.1$, $q = 0.02, \gamma = -0.6$

Figure 1: (a) shows dependency between $\boldsymbol{Y}$, $\boldsymbol{X}$ and $\boldsymbol{A}$. (b) and (c) are dense and sparse graph generated by SBM, which have uniform degree distribution. (d) and (e) are dense and sparse graph generated by DCSBM, which have power-law degree distribution.

Graph Convolutional Filters. By examining various GNN designs, we find most of the GNN operators can fit into a unified framework, i.e., for the l-th layer:

$$\boldsymbol{H}^{(l)} = \sigma(\mathcal{F}(\mathcal{G})\boldsymbol{H}^{(l-1)}\boldsymbol{W}) \tag{1}$$

which describes the three-step process including: (1) a graph convolutional operation (can also be regarded as feature propagation or feature smoothing) denoted as $\mathcal{F}(\mathcal{G})$, (2) a linear transformation denoted by multiplying $\boldsymbol{W}$, and (3) a non-linear transformation denoted by $\sigma(\cdot)$. Clearly, the graph convolutional operation $\mathcal{F}(\mathcal{G})\boldsymbol{H}^{(l-1)}$ is the key step that helps GNNs to obtain performance gain. Thus, to design a good GNN, a powerful graph convolutional filter $\mathcal{F}(\mathcal{G})$ is very crucial.

From this perspective, we analyze the effectiveness of graph filters for existing GNNs in the following. The work of GCN (Kipf & Welling, 2017) first adopts the convolutional operation on graphs and the filter they used is $\mathcal{F}(\mathcal{G}) = \tilde{\boldsymbol{D}}^{-1/2}\tilde{\boldsymbol{A}}\tilde{\boldsymbol{D}}^{-1/2}$. Here, $\tilde{\boldsymbol{A}} = \boldsymbol{A} + \boldsymbol{I}$ is the self-augmented adjacency matrix, and $\tilde{\boldsymbol{D}} = \text{diag}(\tilde{\boldsymbol{d}}_1, ..., \tilde{\boldsymbol{d}}_n)$ is the corresponding degree matrix, where $\tilde{\boldsymbol{d}}_i = \sum_{j=1}^{n} \tilde{\boldsymbol{A}}_{ij}$.

Some studies (Wu et al., 2019a; Maehara, 2019)) use a filter $\mathcal{F}(\mathcal{G}) = (\tilde{\boldsymbol{D}}^{-1/2}\tilde{\boldsymbol{A}}\tilde{\boldsymbol{D}}^{-1/2})^k$ that has similar form of GCN's filter, but with a higher pre-defined exponent k. This would help a node to obtain information from its further neighbors without redundant computation cost. Several other studies propose to use sampling to speed up GNN training (Chen et al., 2018b; Hamilton et al., 2017; Chen et al., 2018a)), which can be considered as a sampled and sparser version of GCN's filter.

Another set of GNNs consider using a learnable graph convolutional filter. For example, Xu et al. (2019) and Chiang et al. (2019) both propose to use $\mathcal{F}(\mathcal{G}) = \boldsymbol{A} + \epsilon \boldsymbol{I}$ where ϵ is a learnable parameter to augment self-loop skip connection. Graph Attention Network (Velickovic et al., 2018)) propose to assign attention to different nodes in a neighborhood as weight of graph convolutional filter. Their graph filter applied on a feature matrix $\boldsymbol{X}$ can be considered as: $\forall i, j$, $\mathcal{F}(\mathcal{G})_{ij} = \frac{exp(\sigma(\boldsymbol{\alpha}[\boldsymbol{W}\boldsymbol{X}_i||\boldsymbol{W}\boldsymbol{X}_j]))}{\sum_{k\in\mathcal{N}_i} exp(\sigma(\boldsymbol{\alpha}[\boldsymbol{W}\boldsymbol{X}_i||\boldsymbol{W}\boldsymbol{X}_k]))}\boldsymbol{A}_{ij}$, where $\mathcal{N}_i$ is the neighborhood of node i, α is a learnable weight vector, and $||$ indicates concatenation.

3 Assessment Tool for Analyzing Graph Convolutional Filters

In this section, we introduce a novel assessment tool for analyzing graph convolutional filters. We first review Fisher score, which is widely used in Linear Discriminant Analysis to quantify the linear separability of two sets of features. Based on it, we propose *Graph Filter Discriminant Score* to evaluate graph convolutional filter on how well it can separate nodes in different classes.

3.1 The Assessment Tool: Graph Filter Discriminant Score

Fisher Score. When coming to traditional non-graph data, Fisher Score (Fisher, 1936) is widely used to assess the linear separability. Given two classes of features $\boldsymbol{X}^{(i)}$ and $\boldsymbol{X}^{(j)}$, Fisher Score is defined as the ratio of the variance between the classes (inter-class distance) to the variance within the classes (inner-class distance) under the best linear projection $\boldsymbol{w}$ of original feature:

$$J(\boldsymbol{X}^{(i)}, \boldsymbol{X}^{(j)}) = \max_{\boldsymbol{w} \in \mathbb{R}^d} \frac{(\boldsymbol{w}^\top(\boldsymbol{\mu}^{(i)} - \boldsymbol{\mu}^{(j)}))^2}{\boldsymbol{w}^\top(\boldsymbol{\Sigma}^{(i)} + \boldsymbol{\Sigma}^{(j)})\boldsymbol{w}} \tag{2}$$

where $\boldsymbol{\mu}^{(i)}$ and $\boldsymbol{\mu}^{(j)}$ denotes the mean vector of $\boldsymbol{X}^{(i)}$ and $\boldsymbol{X}^{(j)}$, and $\boldsymbol{\Sigma}^{(i)}$ and $\boldsymbol{\Sigma}^{(j)}$ denotes the variance of $\boldsymbol{X}^{(i)}$ and $\boldsymbol{X}^{(j)}$. $\boldsymbol{w}$ denotes the linear projection vector. We can understand it as a rotation of coordinate system, and the $\max_{\boldsymbol{w}}$ operation is to find the best direction in which these two class of nodes are most separable. With such procedure, the Fisher score is invariant to any linear transformation of feature, which can eliminate the influence of scale problem. Notice that, the numerator of J indicates inter-class distance and denominator of J indicates inner-class distance. Thus, larger J indicates these two classes of nodes are more linearly separable. Note that for given feature, we can directly get the closed form solution of optimal $\boldsymbol{w}$, with which Fisher Score could be deformed as: $J(\boldsymbol{X}^{(i)}, \boldsymbol{X}^{(j)}) = (\boldsymbol{\mu}^{(i)} - \boldsymbol{\mu}^{(j)})^\top(\boldsymbol{\Sigma}^{(i)} + \boldsymbol{\Sigma}^{(j)})^{-1}(\boldsymbol{\mu}^{(i)} - \boldsymbol{\mu}^{(j)})$. The detailed proof is provided in appendix A.2.

Graph Filter Discriminant Score. As mentioned before, the key component that empowers GNNs is the graph convolutional filter $\mathcal{F}(\mathcal{G})$. Intuitively, an effective filter should make the representation of nodes in different class more separable. Therefore, we propose to use Fisher Score of the node representation before and after applying graph convolutional filter as a metric to evaluate this filter.

For each pair of classes (i, j), we compare the difference of their Fisher Score of initial representations $J(\boldsymbol{X}^{(i)}, \boldsymbol{X}^{(j)})$ and their Fisher Score of representation after filter $J\Big(\big(\mathcal{F}(\mathcal{G})\boldsymbol{X}\big)^{(i)}, \big(\mathcal{F}(\mathcal{G})\boldsymbol{X}\big)^{(j)}\Big)$. We define such difference as "Graph Filter Discriminant Score" for the filter $\mathcal{F}(\mathcal{G})$ as follows.

$$GFD\Big(\mathcal{F}(\mathcal{G})\Big) = \sum_{i \neq j} \frac{n_i + n_j}{\sum_{k \neq t}(n_k + n_t)} \bigg(J\Big(\big(\mathcal{F}(\mathcal{G})\boldsymbol{X}\big)^{(i)}, \big(\mathcal{F}(\mathcal{G})\boldsymbol{X}\big)^{(j)}\Big) - J\big(\boldsymbol{X}^{(i)}, \boldsymbol{X}^{(j)}\big)\bigg) \tag{3}$$

where n_i and n_j is the node size within i and j class, (k, t) denotes all possible class pairs, and $\frac{n_i + n_j}{\sum_{k \neq t}(n_k + n_t)}$ serves as normalization for classes with different node numbers. Intuitively, the higher this GFD score, the more effective is this filter to increase the separability of node feature.

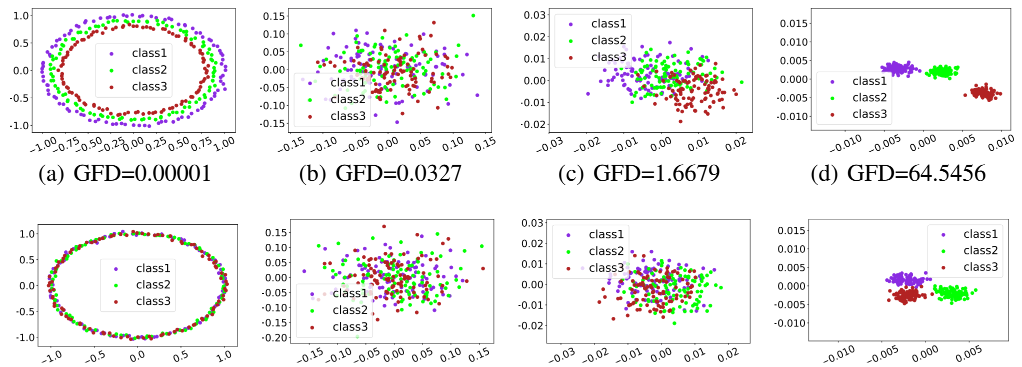
One might doubt whether the Fisher score is only used for evaluating linear separability, as in real-world tasks we are facing many non-linear separable data. We claim the reasonability of such measure by showing that the graph convolutional can actually help non-linearly separable data to be linearly separable. As is shown in Figure 2(a)(b), if we use a proper filter, the convolutional operation can turn three-circle distributions, which are apparently non-linearly separable, into three linearly separable clusters. Moreover, as is shown in Figure 2(c)(d), even the original features of different classes are sampled from the same distribution, the graph convolutional filter can still separate them well. This phenomenon shows that if the graph structure ($\mathcal{G}$) is correlated with the task ($\boldsymbol{Y}$), a proper filter alone is powerful enough to empower GNNs with non-linearity, without any non-linear activation. This phenomenon is also supported by the promising result of SGC (Wu et al., 2019a) which removes all the non-linear activations. Therefore, we claim that the proposed Filter Effective Score is reasonable to measure a given filter's effectiveness.

3.2 Assessing Existing Graph Convolutional Filters

With the help of the assessment tool, we now examine existing filters and try to answer the two fundamental questions: *(1) Is there a best filter that works for all graphs? (2) If not, what are the properties of graph structure that will influence the performance of graph convolutional filters?*

The GFD Score we introduced in the above section can work for any filter on any given graph. From Table 3, we can see that most of the current GNN falls into the following filter family: $\{(\hat{\boldsymbol{A}})^k\}$, where the base $\hat{\boldsymbol{A}}$ is a normalized adjacency matrix, and k is the order of the filter. Noted that there are some other variant of GNN filters that does not fall into this family, but the analyzing procedure is similar. Therefore, without lose of generality, we focus on analyzing this filter family.

The main two components of this filter family is the normalization strategy ($\hat{\boldsymbol{A}}$) and the order (k). For simplicity, we studies the roles of this two components separately, use our assessing tool to show whether there exist a best choice. If not, what are the factors that will influence the choice.



(a) GFD=0.00001 (b) GFD=0.0327 (c) GFD=1.6679 (d) GFD=64.5456

(e) GFD=0.00004 (f) GFD=0.0069 (g) GFD=0.4281 (h) GFD=17.8018

Figure 2: Each row corresponds to a graph. The i-th column of subgraphs corresponds to feature distribution of $(\tilde{\boldsymbol{D}}^{-1/2}\tilde{\boldsymbol{A}}\tilde{\boldsymbol{D}}^{-1/2})^{i-1}\boldsymbol{X}$. The GFD Score is provided in subgraph's title. Both graphs includes 3 classes of nodes with same size and has structure generated by SBM with $p = 0.6$ and $q = 0.03$. The first graph's feature follows circle distribution with radius $= 1, 0.9, 0.8$ for each class and Gaussian noise $= 0.02$. The second graph's feature follows circle distribution with radius $= 1$ and Gaussian noise $= 0.02$ for the three classes;

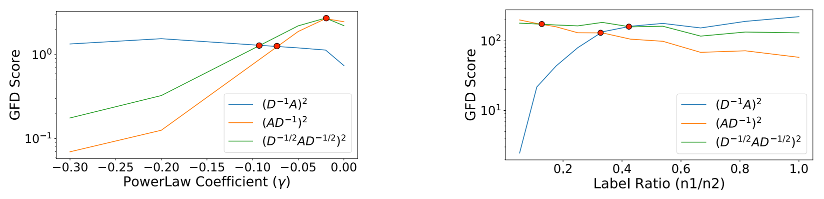


Figure 3: How powerlaw coefficient and label ratio influence the optimal choice of filter's normalization strategy. Detailed Parameters of graph generator are provided in appendix A.3.

Through the analysis, we choose SBM and DCSBM introduced previously to generate synthetic graphs. We enumerate the hyper-parameters to generate graphs with different properties, including powerlaw coefficient (γ), label ratio ($\frac{n_1}{n_2}$), Density ($\frac{p+q}{2}$) and Density gap ($|p - q|$). As these properties are significant for real-world graphs, our generated synthetic graphs can cover a large range of possible graph properties, and are representative for analyzing different filters.

Analyzing Filter's Normalization. We consider the three normalization strategy, including row normalization $\boldsymbol{D}^{-1}\boldsymbol{A}$, column normalization $\boldsymbol{A}\boldsymbol{D}^{-1}$, and symmetric normalization $\boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2}$. We try different graph structure and calculate these three filters' GFD score. As is shown in Figure 3, *none of the normalization strategy can be the best one for all graphs.*[1] Here we give a empirical explanation to this phenomenon. Note that, with a same order, each filter has a same receptive field, and different normalization strategy only influences how to assign weight to the neighboring nodes. The row normalization strategy simply takes the mean of a node's neighbors' feature. Clearly, this would help to keep every node's new representations in a same range. On the contrary, the other two normalization strategy, especially column normalization, might keep a larger representation for high-degree nodes. Applying column-normalized adjacency matrix as base of the graph convolutional filter is similar to PageRank algorithm. While a node propagates its feature to neighbors, this normalization strategy take into account the number of its neighbors. Thus, column normalization could be more helpful when the connectivity pattern is important. Symmetric normalization combines the properties from both row and column normalization. Thus even in the case that the other two doesn't perform well, it can still maintain a promising result.

Since there is no a single 'silver bullet', we are curious about what graph properties will influence the choice of optimal normalization strategy. We find that power-law coefficient γ is a very important

[1]We also provide examples in appendix A.3 to illustrates that each normalization and order will outperform the others in some specific graph data, and none of a single normalization or order can be the best choice.

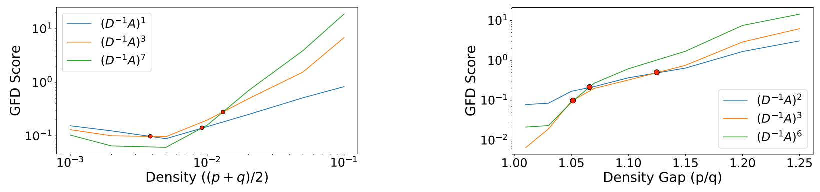


Figure 4: How density and density Gap influence the optimal choice of filter's order. Detailed Parameters of graph generator are provided in appendix A.3.

factor that influence the choice of normalization strategy. As is shown in Figure 3, when power-law coefficient γ decrease (which indicates the graph's extent of power-law), row normalization tends to have better performance. This is because row normalization helps to keep node representations in same range, so that can avoid high degree nodes have larger representation, therefore prevent nodes with similar degree get closer to each other and complicate the classification.

We also find that the label ratio ($\frac{n_1}{n_2}$) matters. As is shown in Figure 3, when the sizes of each class become more imbalanced, column normalization tends to work better. This is because column normalization better leverage degree property and can help nodes in large-size class obtain larger representations, and thus the representations for nodes within different classes would become more separable. Density gap has similar impact on the choice of normalization strategy, since it also controls the component of information received by each node in a similar manner with label ratio.

Analyzing Filter's Order. We then consider what would be the best order for filters. With a high-order filter, a node can obtain information from its further neighbor, thus the amount of its information source during the feature propagation process increased. But do we always need more information under any circumstances? The answer is no. Still, we find that, *for different graphs, the order that obtains best performance would be different*[1].

Since there's no best order for all the cases, we explore the factors that can influence the choice of order. We find the density of graph and Density gap between two classes have big impact. As is shown in figure 4, when the density or density gap increase, the filter with higher order tends to be better choice. We provide an intuitive explanation for this phenomenon as follow. The feature propagation scheme is based on the assumption that nodes in same class have closer connection. On one hand, when the density increases, the connections between nodes are closer. Therefore, high-order filters can help gather richer information and thus makes the representations for node in same class smoother. On the other hand, when the density gap increase, the size of node within same class becomes larger, which will benefit high-order filter.

4 Learning to Find the Optimal Graph Convolutional Filter

Based on previous analysis, we now answer the last question: *Can we design an algorithm to adaptively find the appropriate filter for a given graph?* We develop a simple but powerful model Adaptive Filter Graph Neural Network (AFGNN). For a given graph, AFGNN can learn to combine an effective filter from a set of filter bases, guided by GDF Score.

Adaptive Filter Graph Neural Network (AFGNN). For simplicity, we only consider to to find the optimal filter within a limited family of graph convolutional filters: $\mathbb{F}(\mathcal{G}) = \{\boldsymbol{I}, \tilde{\boldsymbol{D}}^{-1/2}\tilde{\boldsymbol{A}}\tilde{\boldsymbol{D}}^{-1/2}, \cdots, (\tilde{\boldsymbol{D}}^{-1/2}\tilde{\boldsymbol{A}}\tilde{\boldsymbol{D}}^{-1/2})^k, \tilde{\boldsymbol{D}}^{-1}\tilde{\boldsymbol{A}}, \cdots, (\tilde{\boldsymbol{D}}^{-1}\tilde{\boldsymbol{A}})^k, \tilde{\boldsymbol{A}}\tilde{\boldsymbol{D}}^{-1}, \cdots, (\tilde{\boldsymbol{A}}\tilde{\boldsymbol{D}}^{-1})^k\}$, where k is the maximum order. Notice that, we also include identity matrix, which serve as a skip-connection, to maintain original feature representation. Based on our previous analysis, for graphs that are not that correlated to tasks (i.e., small density gap in SBM), identity matrix will outperform all the other convolutional filters. We denote the above $3k+1$ filters as $\mathcal{F}_1^{base}(\mathcal{G}), \cdots, \mathcal{F}_{3k+1}^{base}(\mathcal{G})$, the l-th layer AFGNN is defined as a learnable linear combination of these filter bases:

$$\mathcal{F}_{AFGNN}(\mathcal{G})^{(l)} = \sum_{i=1}^{3k+1} \alpha_i^{(l)} \mathcal{F}_i^{base}(\mathcal{G}), \text{ where } \alpha_i^{(l)} = \frac{\exp(\psi_i^{(l)})}{\sum_{k=1}^{L} \exp(\psi_k^{(l)})} \tag{4}$$

where $\alpha^{(l)}$ is softmax-normalized vector gotten from $\psi^{(l)}$ to combine base filters. Comparing to GNNs with fixed filters such as GCN and SGC, our proposed AFGNN can adaptively learn a filter based on any given graph. Since we've shown none of any filter can be the best for all the graph, an adaptive filter has more capacity to learn better representation. Comparing to other GNNs with learnable filters such as GAT, AFGNN needs less computation cost and can obtain similar or even better performance on most existing benchmark and our synthetic datasets (as shown in experiment section). In the future, we can also consider adding complex filter as GAT into our filter bases.

Training Loss. To train this AFGNN model, we can simply optimize the whole model via any downstream tasks, i.e., node classification. However, most of the semi-supervised node classification datasets only contain very limited training data. The enlarged filter space will make the model prone to over-fitting. Thus, we design to add the GFD Score as a regularization term into the training loss:

$$\mathcal{L} = \mathcal{L}_{CE} + \lambda \mathcal{L}_{GFD}, \text{ where } \mathcal{L}_{GFD} = -\sum_{l}^{L} \mathcal{F}_{AFGNN}(\mathcal{G})^{(l)}(H^{(l-1)}) \tag{5}$$

where $\mathcal{L}_{CE}$ is the cross-entropy loss of the node classification, and $\mathcal{L}_{GFD}$ is defined as the cumulative negation of GFD Score for the learned adaptive filter $\mathcal{F}_{AFGNN}(\mathcal{G})^{(l)}$ at each layer with respect to its input representation $H^{(l-1)}$. Noted that the GFD score assume a fixed data representation, and is invariant to linear transformation, thus it's not reasonable to use it for optimizing parameters in previous layers or the linear weights. Thus, we only use GFD to optimize the filter choice, i.e., ψ.

With a different choice of the regularization term λ, we can categorize our model into:
AFGNN$_0$: With $\lambda = 0$, the model is only trained by classification loss without any regularization, which might be prone to over-fitting when data is not sufficient.
AFGNN$_1$: With $\lambda = 1$, the model is trained by both classification loss and regularization loss, and the learned filter is a linear combination of base filters.
AFGNN$_\infty$: With $\lambda = \infty$, we implement it iteratively: after the model is optimized with $\mathcal{L}_{CE}$, we directly find the optimal combination $\alpha^{(l)}$ of base filter with highest GFD score for each layer l, which is equivalent to applying an infinite regularization. Noted that $H^{(0)}$ is the input feature, we can simply pre-compute the best filter for the first layer and keep it fixed.

5 EXPERIMENTS

In this section, we evaluate whether AFGNN can learn an effective filter and perform well for node classification.

Dataset We first evaluate AFGNN on three widely used benchmark datasets: Cora, Citeseer, and Pubmed (Sen et al., 2008). Based on our findings in section 3.2, we also generate two synthetic benchmarks called SmallGap and SmallRatio. SmallGap corresponds to the case in which the density gap of the graph is close to 1, which means the graph structure is not that correlates to the task, thus $\boldsymbol{I}$ is even better than all the other filters. SmallRatio corresponds to the case in which the label ratio is small, i.e. the size of one class is clearly smaller than the other, and column normalization $\boldsymbol{AD}^{-1}$ is the best normalization[2].

Baselines and Settings. We compare against 5 baselines, including GCN, GIN, SGC, GFNN, and GAT. To make fair comparisons, for all the baseline GNNs, we set the number of layers (or orders) to be 2, and tune the parameters including learning rate, weight decay, and number of epochs[3].

For all the benchmark datasets, we follow the same setting of (Kipf & Welling, 2017). For the synthetic dataset, we conduct 5-fold cross-validation, randomly split the nodes into 5 groups of the same size, taking one group as the training set, one as the validation set and the remaining three as the test set. Each time we pick the model with highest validation accuracy and record its test accuracy. For each dataset, we run the experiment 10 times and computes mean and standard deviation of recorded test accuracy.

Classification Performance. As is shown in Tabel 1, our proposed AFGNN$_P$ model can consistently achieve competitive test accuracy. On Cora and Citeseer, only GAT outperform our proposed

[2]The statistics of dataset and parameters used to generate synthetic dataset are provided in Appendix A.6
[3]The details about baseline code and hyperparameters settings are provided in Appendix A.7

Dataset	GCN	GIN	SGC	GFNN	GAT	**AFGNN$_0$**	**AFGNN$_1$**	**AFGNN$_\infty$**
Cora	80.85±0.43	76.37±0.75	81.14±0.05	80.42±0.70	**82.90± 0.01**	61.22±1.61	81.16±0.48	**81.40 ±0.03**
Citeseer	71.19±0.60	67.85±0.52	71.91±0.01	71.15±0.55	**72.20±0.07**	60.58±0.91	71.32±0.85	**71.80±0.01**
Pubmed	79.08±0.23	74.23±1.76	78.50±0.00	79.12±0.23	78.50±0.01	72.58±2.84	78.51±0.14	**79.20±0.01**
SmallGap	82.78±0.20	76.83±0.87	74.53±0.94	83.38±0.30	85.26±0.07	90.85±3.24	99.91±0.04	**99.95±0.01**
SmallRatio	87.79±1.05	77.82±3.40	87.14±0.19	83.75±0.20	82.10±0.01	74.45±4.81	85.69±3.69	**93.80±1.11**

Table 1: Test accuracy of different models on both benchmark and synthetic datasets.

Graph Filters	Cora	Citeseer	Pubmed	SmallGap	SmallRatio
$\boldsymbol{I}$	13.15	25.05	11.75	**34.13**	3.62
$\tilde{\boldsymbol{D}}^{-1}\tilde{\boldsymbol{A}}$	33.68	48.24	22.05	0.97	6.25
$(\tilde{\boldsymbol{D}}^{-1}\tilde{\boldsymbol{A}})^2$	**58.48**	**67.53**	**37.70**	18.17	10.50
$\tilde{\boldsymbol{A}}\tilde{\boldsymbol{D}}^{-1}$	28.35	43.99	12.44	0.97	10.67
$(\tilde{\boldsymbol{A}}\tilde{\boldsymbol{D}}^{-1})^2$	49.64	62.16	22.74	16.27	**80.37**
$\tilde{\boldsymbol{D}}^{-1/2}\tilde{\boldsymbol{A}}\tilde{\boldsymbol{D}}^{-1/2}$	31.87	47.20	20.09	0.97	8.51
$(\tilde{\boldsymbol{D}}^{-1/2}\tilde{\boldsymbol{A}}\tilde{\boldsymbol{D}}^{-1/2})^2$	54.76	65.50	34.61	17.58	49.69
AFGNN$_0$	19.72	34.84	12.02	34.29	8.50
AFGNN$_1$	58.28	**67.83**	**38.61**	34.59	77.81
AFGNN$_\infty$	**58.48**	67.68	**38.61**	**34.68**	**80.37**

Table 2: Graph Filter Discriminant Score of different filter on both benchmark and synthetic datasets

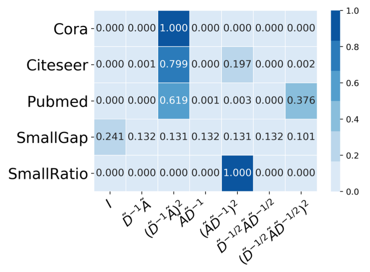


Figure 5: Base filter combination Learned by AFGNN$_\infty$

model, which takes a longer time to train and converge. While on Pubmed and the other two synthetic benchmark, AFGNN$_P$ can achieve the best results among all the baseline models.

We further compare our AFGNN$_0$, AFGNN$_1$, AFGNN$_\infty$ to examine the role of GFD regularization. The AFGNN$_0$ performs quite bad on all the datasets, implying that the larger searching space of filter without regularization is prone to over-fitting, while AFGNN$_1$ and AFGNN$_\infty$ performs much better. Among these two, AGFNN$_\infty$ performs even better, which indicates the GFD Score is indeed a very powerful tool for assessing the effectiveness of the filter.

Graph Filter Discriminant Analysis. We are also interested to see whether the proposed method can indeed pick the best combination of the filter from the base filter family. To do so, we calculate the GFD Score of the first-layer filter learned by AFGNN$_0$, AFGNN$_1$, AFGNN$_\infty$ and the seven base filters on the test set for each dataset. Tabel 2 shows the GFD score and Figure 5 shows the filter combination learned by AFGNN$_\infty$, we can see that our proposed method can indeed learn a combined filter on all the datasets. Specifically, in all the benchmark dataset, the best base filter is all $(\tilde{\boldsymbol{D}}^{-1}\tilde{\boldsymbol{A}})^2$, and our proposed adaptive filter not only can pick out the best base filter but also learn a better combination. For the two synthetic datasets, where $\boldsymbol{I}$ and $(\tilde{\boldsymbol{A}}\tilde{\boldsymbol{D}}^{-1})^2$ are the best filters, our algorithm can also learn to pick them out. All the analysis show that the proposed GFD regularization can help find an appropriate filter for a given dataset.

6 Conclusion

Understanding the graph convolutional filters in GNNs is very important, it can help to answer the question whether a GNN will work on a given graph, and can provide important guidance for GNN design. In our paper, we focus on semi-supervised node classification task. We first propose Graph Filter Discriminant Score as an assessment tool for graph convolutional filter evaluation, and then apply this GFD Score to analyze a family of existing filters as case study, and find out there's no filter can always be the best one for all graphs. We develop a simple but powerful GNN model: Adapative Filter Graph Neural Network, which can learns to combine a family of filters and obtain a task-specific powerful filter. We also propose to add the neigative GFD Score as a regularization component to the objective function, it can act as a guidance to learn a more effective filter. Experiment shows our approach outperforms many existing GNNs on both benchmark and synthetic graphs.

REFERENCES

Jianfei Chen, Jun Zhu, and Le Song. Stochastic training of graph convolutional networks with variance reduction. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, pp. 941–949, 2018a. URL `http://proceedings.mlr.press/v80/chen18p.html`.

Jie Chen, Tengfei Ma, and Cao Xiao. Fastgcn: Fast learning with graph convolutional networks via importance sampling. *CoRR*, abs/1801.10247, 2018b. URL `http://arxiv.org/abs/1801.10247`.

Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019, Anchorage, AK, USA, August 4-8, 2019.*, pp. 257–266, 2019.

Ronald A Fisher. The use of multiple measurements in taxonomic problems. *Annals of eugenics*, 7 (2):179–188, 1936.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pp. 1024–1034, 2017.

Paul W Holland, Kathryn Blackmond Laskey, and Samuel Leinhardt. Stochastic blockmodels: First steps. *Social networks*, 5(2):109–137, 1983.

Brian Karrer and Mark EJ Newman. Stochastic blockmodels and community structure in networks. *Physical review E*, 83(1):016107, 2011.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017. URL `https://openreview.net/forum?id=SJU4ayYgl`.

Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

Takanori Maehara. Revisiting graph neural networks: All we have is low-pass filters. *arXiv preprint arXiv:1905.09550*, 2019.

Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *European Semantic Web Conference*, pp. 593–607. Springer, 2018.

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

Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*, 2018. URL `https://openreview.net/forum?id=rJXMpikCZ`.

Felix Wu, Tianyi Zhang, Amauri Holanda de Souza Jr, Christopher Fifty, Tao Yu, and Kilian Q Weinberger. Simplifying graph convolutional networks. *arXiv preprint arXiv:1902.07153*, 2019a.

Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S Yu. A comprehensive survey on graph neural networks. *arXiv preprint arXiv:1901.00596*, 2019b.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019. URL `https://openreview.net/forum?id=ryGs6iA5Km`.

Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 974–983. ACM, 2018.

GNNs	Graph Convolutional Filters
GCN (Kipf & Welling, 2017)	$\mathcal{F}(\mathcal{G}) = \tilde{\boldsymbol{D}}^{-1/2}\tilde{\boldsymbol{A}}\tilde{\boldsymbol{D}}^{-1/2}$
SGC (Wu et al., 2019a)	$\mathcal{F}(\mathcal{G}) = (\tilde{\boldsymbol{D}}^{-1/2}\tilde{\boldsymbol{A}}\tilde{\boldsymbol{D}}^{-1/2})^k$
GFNN (Maehara, 2019)	$\mathcal{F}(\mathcal{G}) = (\tilde{\boldsymbol{D}}^{-1/2}\tilde{\boldsymbol{A}}\tilde{\boldsymbol{D}}^{-1/2})^k$
GIN (Xu et al., 2019)	$\mathcal{F}(\mathcal{G}) = \boldsymbol{A} + \epsilon\boldsymbol{I}$
GAT (Velickovic et al., 2018)	$\mathcal{F}(\mathcal{G}) = \boldsymbol{Q}$, where $\boldsymbol{Q}$ is parametric attention function of $\boldsymbol{X}$ and $\boldsymbol{A}$

Table 3: A Summary of Graph Filters of Existing GNNs.

A Appendix

A.1 Summary of Graph Filters for Existing GNNs.

Tabel 3 summarized the graph filters for existing GNNs.

A.2 Proof of Proposition 1

Proof According to the conclusions in linear discriminant analysis, the maximum separation occurs when $\boldsymbol{w} \propto (\boldsymbol{\Sigma}^{(i)} + \boldsymbol{\Sigma}^{(j)})^{-1}(\boldsymbol{\mu}^{(i)} - \boldsymbol{\mu}^{(j)})$. Notice that, when we want to apply this fisher linear discriminant score in our problem, the linear transformation part in our classifier (and also the linear transformation part in GNN) will help to find the best $\boldsymbol{w}$. Thus, we can directly plug the optimum solution $\boldsymbol{w}^* = c(\boldsymbol{\Sigma}^{(i)} + \boldsymbol{\Sigma}^{(j)})^{-1}(\boldsymbol{\mu}^{(i)} - \boldsymbol{\mu}^{(j)})$ into this formula, here c is a scalar. Then, we'll have:

$$\begin{aligned}
J(\boldsymbol{X}^{(i)}, \boldsymbol{X}^{(j)}) &= \max_{\boldsymbol{w}\in\mathbb{R}^d} \frac{(\boldsymbol{w}^\top(\boldsymbol{\mu}^{(i)} - \boldsymbol{\mu}^{(j)}))^2}{\boldsymbol{w}^\top(\boldsymbol{\Sigma}^{(i)} + \boldsymbol{\Sigma}^{(j)})\boldsymbol{w}} \\
&= \frac{\left((c(\boldsymbol{\Sigma}^{(i)} + \boldsymbol{\Sigma}^{(j)})^{-1}(\boldsymbol{\mu}^{(i)} - \boldsymbol{\mu}^{(j)}))^\top(\boldsymbol{\mu}^{(i)} - \boldsymbol{\mu}^{(j)})\right)^2}{(c(\boldsymbol{\Sigma}^{(i)} + \boldsymbol{\Sigma}^{(j)})^{-1}(\boldsymbol{\mu}^{(i)} - \boldsymbol{\mu}^{(j)}))^\top(\boldsymbol{\Sigma}^{(i)} + \boldsymbol{\Sigma}^{(j)})(c(\boldsymbol{\Sigma}^{(i)} + \boldsymbol{\Sigma}^{(j)})^{-1}(\boldsymbol{\mu}^{(i)} - \boldsymbol{\mu}^{(j)}))} \\
&= \frac{\left((\boldsymbol{\mu}^{(i)} - \boldsymbol{\mu}^{(j)})^\top(\boldsymbol{\Sigma}^{(i)} + \boldsymbol{\Sigma}^{(j)})^{-1}(\boldsymbol{\mu}^{(i)} - \boldsymbol{\mu}^{(j)})\right)^2}{(\boldsymbol{\mu}^{(i)} - \boldsymbol{\mu}^{(j)})^\top(\boldsymbol{\Sigma}^{(i)} + \boldsymbol{\Sigma}^{(j)})^{-1}(\boldsymbol{\mu}^{(i)} - \boldsymbol{\mu}^{(j)})} \\
&= (\boldsymbol{\mu}^{(i)} - \boldsymbol{\mu}^{(j)})^\top(\boldsymbol{\Sigma}^{(i)} + \boldsymbol{\Sigma}^{(j)})^{-1}(\boldsymbol{\mu}^{(i)} - \boldsymbol{\mu}^{(j)})
\end{aligned} \tag{6}$$

Thus we completed the proof. ■

Comment. When each dimension of feature is independently with each other, i.e. $(\boldsymbol{\Sigma}^{(i)} + \boldsymbol{\Sigma}^{(j)})^{-1}$ is diagonal matrix, we can further deform the above formula as:

$$(\boldsymbol{\mu}^{(i)} - \boldsymbol{\mu}^{(j)})^\top(\boldsymbol{\Sigma}^{(i)} + \boldsymbol{\Sigma}^{(j)})^{-1}(\boldsymbol{\mu}^{(i)} - \boldsymbol{\mu}^{(j)}) = \sum_{t=1}^{d} \frac{(\boldsymbol{\mu}_t^{(i)} - \boldsymbol{\mu}_t^{(j)})^2}{\boldsymbol{\Sigma}_{tt}^{(i)} + \boldsymbol{\Sigma}_{tt}^{(j)}}$$

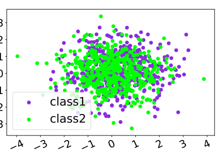
A.3 No Best Filter for All Graphs

A.3.1 Examples of "No Best Normalization Strategy for All"

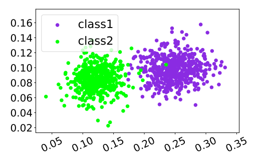
Figure 6 provides two examples to show there is no best normalization strategy for all graphs. For both examples, we fix the order of filter to be 2.

The first row shows a case in which row normalization is better than the other two. The corresponding graph contains 2 classes of nodes with size 500. The graph structure is generated by DCSBM with $p = 0.3$, $q = 0.05$, power-law coefficient $\gamma = -0.9$. The features for two classes satisfy multivariate distribution with an identity co-variance matrix, and with mean (0.2,0.2) and (0,0) respectively. In this example, we can clearly see that with other two normalization strategy, some high-degree hubs show up in the upper right corner from both class, which is harmful for classification. We generate this example to illustrate the benefit of row normalization because row normalization would be very helpful for a graph with power-law degree distribution, which contains some nodes with unusually large degree (those nodes are called hubs), since it can help avoid those hubs obtaining larger representations and thus be mis-classified.

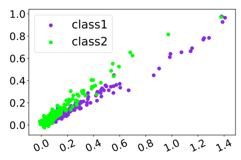
The second row shows a case in which column normalization is better than the other two. The corresponding graph contains 2 classes of nodes with size 900 and 100 respectively. The graph structure is generated by SBM with $p = 0.3$, $q = 0.2$. The features for two classes satisfy multivariate distribution with an identity co-variance matrix, and with mean (-0.2,-0.2) and (0.2,0.2) respectively. We generate this example to illustrate the benefit of column normalization because under this case, we should consider taking more degree information into consideration. Therefore, column normalization would be more helpful.



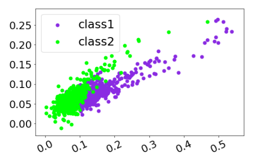
(a) Initial Feature. GFD=0.0212



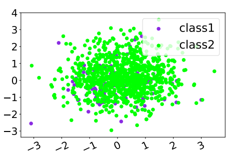
(b) Row-nomalized. GFD=12.3236



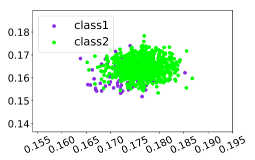
(c) Column-nomalized. GFD=0.1371



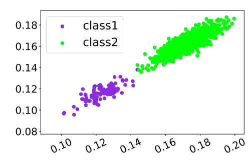
(d) Symmetric-nomalized. GFD=0.9343



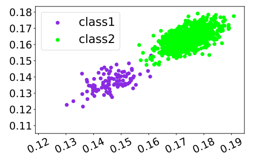
(e) Initial Feature. GFD=0.1267



(f) Row-nomalized. GFD=0.5820



(g) Column-nomalized. GFD=32.6420



(h) Symmetric-nomalized. GFD=24.0931

Figure 6: Examples of “No Best Normalization Strategy for All”

A.3.2 Examples of “No Best Order for All”

Figure 7 provides two examples to show there is no best order for all graphs. For both examples, we fix the normalization strategy to be row normalization, and varies order to be 2, 4, 6.

The first row shows a case in which small order is better than the large ones. The corresponding graph contains 2 classes of nodes with same size 500. The graph structure is generated by SBM with $p = 0.215$, $q = 0.2$. The features for two classes satisfy multivariate distribution with an identity co-variance matrix, and with mean (0.5,0.5) and (0,0) respectively.

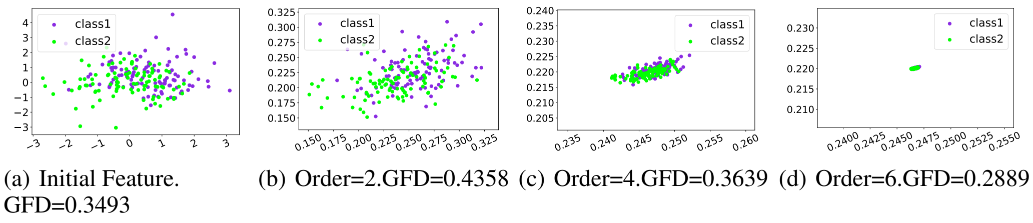
The second row shows a case in which large order is better than the smaller ones. The corresponding graph contains 2 classes of nodes with same size 500. The graph structure is generated by SBM with $p = 0.75$, $q = 0.6$. The features for two classes satisfy multivariate distribution with an identity co-variance matrix, and with mean (0.5,0.5) and (0,0) respectively.

A.3.3 Illustration of Graph Generator for Curves in Section 3.2

For the curves indicating how powerlaw coefficient influence the choice of normalization in Figure 3, we generate the corresponding graphs structure by DCSBM with fixed $p = 0.3$, $q = 0.2$ and varies the powerlaw coefficient from -0.3 to 0. The graph contains two classes of nodes, and is of size 400 and 600 for each class respectively. The feature for each class satisfies multivariate normal distribution with identity co-variance matrix, and with mean (0,0) and (0.2,0.2).

For the curves indicating how label ratio influence the choice of normalization in Figure 3, we generate the corresponding graphs structure by SBM with fixed $p = 0.3$, $q = 0.1$ and varies the label ratio. The graph contains a total number of 1000 nodes in two classes. The feature for each class satisfies multivariate normal distribution with identity co-variance matrix, and with mean (0,0) and (0.5,0.5).

For the curves indicating how density influence the choice of normalization in Figure 4, we generate the corresponding graphs structure by SBM with fixed density gap $p/q = 1.5$ and varies the density


(a) Initial Feature. GFD=0.3493 (b) Order=2.GFD=0.4358 (c) Order=4.GFD=0.3639 (d) Order=6.GFD=0.2889

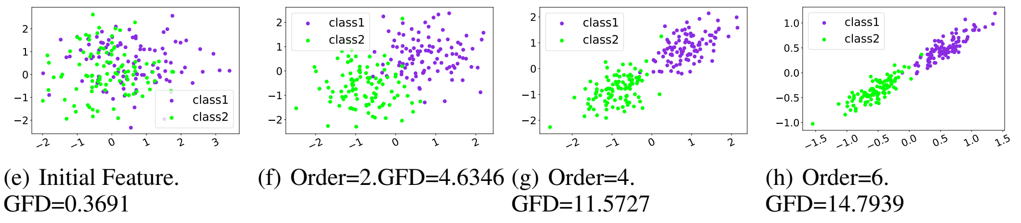

(e) Initial Feature. GFD=0.3691 (f) Order=2.GFD=4.6346 (g) Order=4. GFD=11.5727 (h) Order=6. GFD=14.7939

Figure 7: Examples of “No Best Order for All”

by varying q. The graph contains two classes of node of size 500. The feature for each class satisfies multivariate normal distribution with identity co-variance matrix, and with mean (0,0) and (0.5,0.5).

For the curves indicating how density gap influence the choice of normalization in Figure 4, we generate the corresponding graphs structure by SBM with fixed density $p + q = 0.6$ and varies the density gap. The graph contains two classes of node of size 500. The feature for each class satisfies multivariate normal distribution with identity co-variance matrix, and with mean (-0.2,-0.2) and (0.2,0.2).

A.4 Flowchart of AFGNN for node classification

The following flowchart (Figure 8) describes the process of how a one-layer AFGNN tackle node classification task.

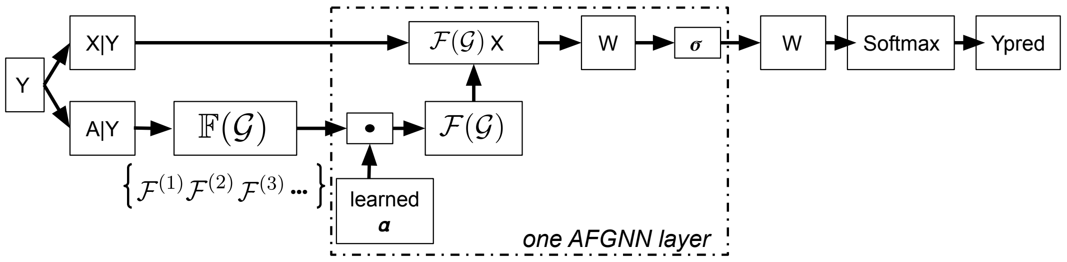
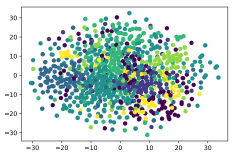

Y
X|Y
A|Y
$\mathbb{F}(\mathcal{G})$
$\{\mathcal{F}^{(1)}\mathcal{F}^{(2)}\mathcal{F}^{(3)}\dots\}$
•
learned α
$\mathcal{F}(\mathcal{G})$
$\mathcal{F}(\mathcal{G})$ X
W
σ
W
Softmax
Ypred
one AFGNN layer

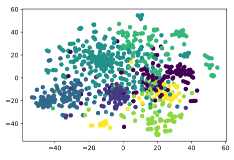
Figure 8: Flowchart of a 1-layer AFGNN for node classification.

A.5 Feature Propagation using different filters

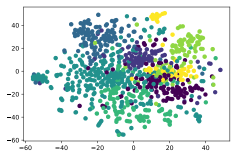
We reduced the dimension of feature by t-SNE(Maaten & Hinton (2008)). We annotate the filter and the GFD Score in title of each subfigure. Noted that, identity also corresponds to the initial feature. The figure is the feature representation obtained after conduct graph convolution operation once with the corresponding filter.



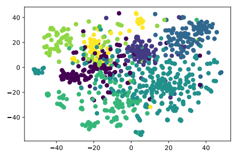
(a) Identity:13.15



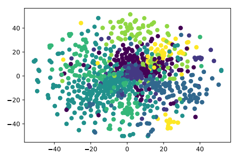
(b) AFGNN:58.28



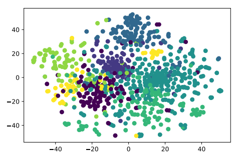
(c) row1:33.68



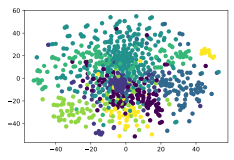
(d) row2:58.48



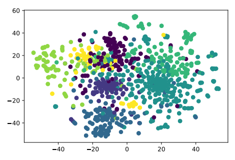
(e) col1:28.35



(f) col2:49.64



(g) sym1:31.87



(h) sym2:54.76

Dataset	**Nodes**	**Edges**	**Classes**	**Features Dimension**
Cora	2708	5429	7	1433
Citeseer	3327	4732	6	3703
Pubmed	19717	44338	3	500

Table 4: Statistics of Benchmark Dataset

A.6 Dataset

A.6.1 Benchmark Dataset

We use three benchmark dataset: Cora, Citeseer and Pubmed for the node classification task. Their statictics are in table4.

A.6.2 Synthetic Dataset

We also generated two synthetic datasets: SmallGap and SmallRatio. For SmallGap, we use SBM to generate a two class network with $p = 0.2$ and $q = 0.199$. The density gap p/q is very small in this case. They have the same number of nodes and both have 64 dimension features sampled from gaussian distributions with different mean and same variance. For SmallRatio, we use SBM to generate a two class network, which has 200 nodes for one class and 800 nodes for the other. This dataset is called SmallRatio because $n1/n2 = 0.25$ is small. Their 64 features are sampled from gaussian distributions with different mean and different variance. The detailed generation process and parameter can be found in our code.

A.7 Model Hyper Parameters

For GCN, SGC, GFNN, GAT, we directly use their public implementations. For GIN, the initial code is not for node classification task, so we modify their code following Xu et al. (2019) to conduct experiments.

We tune the number of epochs based on convergence performance. For learning rate and weight decay, we follows the parameter setting provides by the corresponding public implementations unless we find better parameters. The tuned parameters can be found in our code resource.

A.8 Baseline Accuracy on Benchmark Dataset

We report the accuracy of node classification task for baseline models on Cora, Citeseer, and Pubmed provided by corresponding literature. Since GIN (Xu et al., 2019) is not originally evaluated on node classification task, we do not have the reported number here. The results is in Tabel 5.

	Cora	Citeseer	Pubmed
GCN	81.5	70.3	79.0
GIN	–	–	–
SGC	81.0±0.0	71.9±0.1	78.9±0.0
GFNN	80.9±1.3	69.3±1.1	81.2±1.5
GFN	–	–	–
GAT	83.0±0.7	72.5±0.7	79.0±0.3

Table 5: Baseline's Accuracy on Benchmark Dataset