Demystifying Graph Neural Networks with Graph Filter Assessment

Yewen Wang
University of California, Los Angeles
wyw10804@cs.ucla.edu

Ziniu Hu
University of California, Los Angeles
bull@cs.ucla.edu

Yusong Ye
University of California, Los Angeles
yusongye@ucla.edu

Yizhou Sun
University of California, Los Angeles
yzsun@cs.ucla.edu

ABSTRACT

Graph Neural Networks (GNNs) have recently received tremendous attention due to their power in handling graph data for different downstream tasks across different application domains. Many GNN models have been proposed, which mainly differ in their graph filter design with the hope to find the best filter for all the graph data. However, there still lack studies on graph filter assessment from a data perspective. In particular, we raise the following three questions: *(1) Whether there exists an optimal filter that performs the best on all graph data; (2) Which graph properties should be considered for finding the optimal graph filter; and (3) How to design appropriate filters that adapt to a given graph.* In this paper, we focus on addressing the above questions, using semi-supervised node classification task as a case study. We propose a novel assessment tool: Graph Filter Discriminant Score (GFD), for evaluating the effectiveness of graph filters for a given graph in terms of node classification. Using this tool, we find out that there is no single filter that performs the best on all possible graphs, and graphs with different properties are in favor of different graph filters. Based on these findings, we develop Adaptive Filter Graph Neural Network (AFGNN), a simple but powerful model that can adaptively learn data-specific filters. For a given graph, AFGNN leverages graph filter assessment as an extra loss term and learns to combine a set of base filters. Experiments on both synthetic and real-world benchmark datasets have demonstrated that our proposed model has the flexibility in learning a data-specific filter and can consistently provide competitive performance across all the datasets.

1 INTRODUCTION

Graph Neural Networks (GNNs) are a family of powerful tools for representation learning on graph data. They 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 [12], graph classification [26], graph matching [1, 16], recommendation systems [28], and knowledge graphs [20].

KDD'20 workshop, August 2020, San Diego, CA, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

As GNNs have superior performance in graph-related tasks, the question "what makes GNNs so powerful?" is naturally raised. Note that GNNs adopt the concept of the convolution operation into graph domain. To obtain a representation of a specific node with GNN, the node aggregates representations of its neighbors with a graph filter. For a task related to graph topology, the graph filter decides how the information is propagated among nodes, and can help GNN nodes to get better task-specific representations [27]. Therefore, the key to designing robust and effective GNNs should be designing proper graph filters that can best leverage the graph topology for a given graph.

Recently, many GNN architectures are proposed [6, 13, 18, 22, 24, 27], with their own graph filter designs. However, none of them have properly answered the following fundamental questions: *(1) Is there a best filter that works for all graphs? (2) If not, what are the graph properties that will influence the performance of graph filters? (3) Can we design an algorithm to adaptively find the best filter for a given graph?* In this paper, we discuss the above questions, using semi-supervised node classification task as a case study.

Inspired by studies in Linear Discriminant Analysis (LDA), we propose a Graph Filter Discriminant (GFD) Score metric to measure the power of a graph filter in discriminating node representations of different classes on a specific graph. We analyzed some existing GNNs' filters with this assessment method to answer the three aforementioned questions. We found that no single filter design can achieve optimal results on all possible graphs, which means we should adopt different graph filters for different graph data. We then empirically analyze how graph properties influence the optimal choice of graph filters. Based on our findings, we propose the Adaptive Filter Graph Neural Network (AFGNN), which can adaptively learn a good data-specific filter for the given graph. We also use the negative GFD as a an extra loss term to guide the learning. We show that the proposed AFGNN 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: Graph Filter Discriminant Score (GFD), to analyze the effectiveness of graph filters. Using this tool, we find that no filter is the best for all graphs, and the graph filter should be adaptive to the graph data.
- We propose Adaptive Filter Graph Neural Network (AFGNN) that can adaptively learn a proper filter for a specific graph using the GFD Score as guidance.

- We demonstrate that the proposed model can find better filters and achieve better performance compared to existing GNNs, on both real-word benchmark and synthetic datasets.

2 PRELIMINARIES

2.1 Semi-Supervised Node Classification

We denote an undirected graph by $\mathcal{G}(\mathbb{V}, \boldsymbol{A})$, where $\mathbb{V}$ is a set of nodes in this graph, and $\boldsymbol{A}$ is the adjacency matrix for this graph. Some graph data has node feature, for these graphs, let $\boldsymbol{X}$ be the feature matrix, where each row $\boldsymbol{X}_v$ is the feature for node $v \in \mathbb{V}$. For semi-supervised node classification, let $\boldsymbol{Y}$ be the class assignment vector for all the nodes in $\mathbb{V}$, C be the number of classes, and $\boldsymbol{Y}_v \in \{1, \cdots, C\}$ be the class that node v belongs to. Then, the goal is to learn a mapping function $f : \mathbb{V} \rightarrow \{1, \cdots, C\}$ by leveraging node features $\boldsymbol{X}$, graph structure $\boldsymbol{A}$ and the labeled nodes, to predict the class labels for the unlabeled nodes, i.e., $\hat{\boldsymbol{Y}}_v = f(v)$.

2.2 A Review of Existing Graph Filters

Graph Neural Networks are shown to be a promising technique to solve graph-related tasks including the semi-supervised node classification task we introduced before. Various GNNs have been proposed with their own graph filter designs. By examining these designs, we find that most of the GNN operators can fit into a unified framework, i.e., for the l-th layer:

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

where $\mathcal{H}^{(l)}$ is the node representation for l-th layer, $\mathcal{F}(\mathcal{G})$ is the graph filter for graph $\mathcal{G}$, $\boldsymbol{W}^{(l)}$ is the learnable linear transformation parameter. This formula describes the three-step process that involves: (1) a graph convolutional operation (i.e. feature propagation or feature smoothing, denoted as $\mathcal{F}(\mathcal{G})\boldsymbol{H}^{(l-1)}$), (2) a linear transformation (i.e. multiplying $\boldsymbol{W}^{(l)}$), and (3) a non-linear transformation (i.e. $\sigma(\cdot)$). Clearly, the graph convolutional operation $\mathcal{F}(\mathcal{G})\boldsymbol{H}^{(l-1)}$ is the key step that helps GNNs to improve performance. Thus, to design a good GNN, a powerful graph convolutional filter $\mathcal{F}(\mathcal{G})$ is crucial. A summary of some existing filter designs is given in A.1.

Some GNNs use a fixed graph filter. The work of GCN [12] first adopts the convolutional operation on graphs and use the symmetrically normalized adjacency matrix as the graph filter. Several studies propose to use sampling strategy to speed up GNN training [2, 3, 8]), which can be considered as a sparser version of GCN's filter. Some studies such as SGC and GFNN [18, 24] use a higher-order symmetrically normalized adjacency matrix that includes a pre-defined exponent. This would help a node to obtain information from its further neighbors without redundant computation cost.

Another set of GNNs consider using a learnable graph convolutional filter. Some works [4, 27] propose to include a learnable parameter to augment self-loop skip connection. The filters designed by [9, 14, 17, 23] are polynomials or rational polynomials of original or transformed graph Laplacian matrices, with learnable polynomial coefficients. Graph Attention Networks[22] proposes to assign attention weight to different nodes in a neighborhood, which can be considered as a flexible learnable graph convolutional filter that is a parametric attention function of $\boldsymbol{X}$ and $\boldsymbol{A}$.

Among all these existing graph filters, the fixed filters use the same propagation strategy for all the graph data, though some of them noticed the importance to use different propagation rules [6] and tried to include diverse propagation rules, but their designed filter is still a fixed combination of all the rules for all graphs and therefore still lack flexibility. The learnable filters are adaptive to the graph data, but the existing learnable filters are either too simple or too complex, the simple ones is not adaptive enough while the complex ones would have huge computation cost and are easy to overfit for simple graphs.

2.3 Graph Generator

To systematically analyze the performance of different GNN filters, we test their performance under different graph data with different properties, i.e., graphs with different $\boldsymbol{X}$, $\boldsymbol{A}$, $\boldsymbol{Y}$. Usually, the node classification task requires node features ($\boldsymbol{X}$) and the graph structure ($\boldsymbol{A}$) to be correlated to the intrinsic node labels ($\boldsymbol{Y}$), so taking both correlations into consideration may enhance the performance of this task. To better understand the roles played by each component, we assume the graphical model to generate a graph data is as described in Fig. 1(a). To disclose the relationship between diverse graph filters and properties of graphs, we further make assumptions on how X and A are generated when Y is given. The generation of $\boldsymbol{Y}$, $\boldsymbol{X}|\boldsymbol{Y}$, and $\boldsymbol{A}|\boldsymbol{Y}$ are introduced as follows.

Generating $\boldsymbol{Y}$: Each node is randomly assigned with a class label with probability proportional to its class size. We assume each class c is associated with n_c nodes.

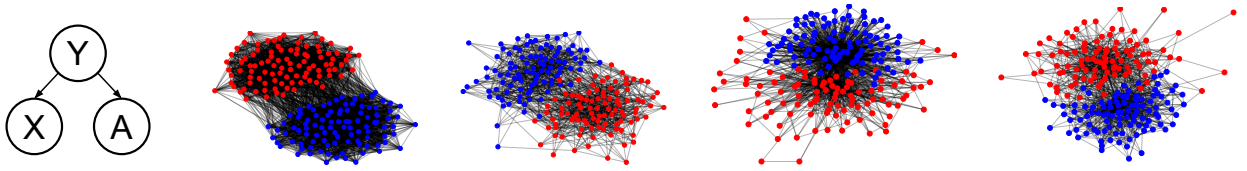
Generating $\boldsymbol{X}|\boldsymbol{Y}$: We assume that node features are sampled from a distribution determined by their labels. For example, we can sample node features of class c from a multivariate Gaussian distribution with the parameters conditioned on c: $\boldsymbol{X}^{(c)} \sim \mathcal{N}(\boldsymbol{\mu}^{(c)}, \boldsymbol{\Sigma}^{(c)})$. We may also sample node features of class c from a circular distribution with radius r_c and noise $noise_c$ conditioned on c.

Generating $\boldsymbol{A}|\boldsymbol{Y}$: We use the classic class-aware graph generator: stochastic block model (SBM [10]), to generate graph structure conditioned on labels. SBM has several assumptions: (1) edges are generated via Bernoulli distributions independently and (2) the parameter of the Bernoulli distribution is determined by the classes of the corresponding pair of nodes v_i and v_j, i.e., $A_{ij}|Y_i, Y_j \sim Ber(p_{Y_iY_j})$, where $p_{Y_iY_j}$ is a parameter determined by the two corresponding classes. In the simplest case, if a pair of nodes (i,j) belong to same class, then the probability that this pair is linked is $p_{Y_iY_j} = p$, otherwise, $p_{Y_iY_j} = q$. We call p internal density and q external density. We assume $p \geq q$, both p and q should be the input of the SBM model. Degree Corrected SBM (DCSBM, [11]) is a variation of SBM, it adds a parameter γ to control the *"power-law"* exponent of degree distribution among nodes. Figure 1(b-e) demonstrates examples of synthetic graphs generated by SBM and DCSBM with different graph structure properties.

Plus, we investigate key properties of graph data to characterize different graphs in A.2.

3 THE ASSESSMENT TOOL: GRAPH FILTER DISCRIMINANT SCORE (GFD)

In this section, we introduce a novel assessment tool for analyzing graph filters. We first review the Fisher score [7], which is widely used to quantify the linear separability of two sets of features. With the Fisher score, we propose the GFD metric to evaluate the graph



(a) Y, X, A dependency (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 Y, X and A. (b), (c) are dense and sparse graph generated by SBM, which have uniform degree distribution. (d), (e) are dense and sparse graph generated by DCSBM, which have power-law degree distribution.

filter on how well it can separate nodes in different classes.

Fisher Score. Given two classes of features $\boldsymbol{X}^{(i)}$, $\boldsymbol{X}^{(j)}$, the Fisher Score is defined as the ratio of their inter-class distance to inner-class distance under the best linear projection $\boldsymbol{w}$ of the raw feature:

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

where $\boldsymbol{\mu}^{(i)}$ and $\boldsymbol{\mu}^{(j)}$ are the mean vector of $\boldsymbol{X}^{(i)}$ and $\boldsymbol{X}^{(j)}$, $\Sigma^{(i)}$ and $\Sigma^{(j)}$ are the co-variance matrix of $\boldsymbol{X}^{(i)}$ and $\boldsymbol{X}^{(j)}$ respectively, $\boldsymbol{w}$ is the linear projection vector which we can understand as a rotation of the coordinate system, and the $\max_{\boldsymbol{w}}$ operation is to find the best direction in which these two class of nodes are most separable. A larger value of J indicates higher separability. Given features, we can get rid of the $\max_{\boldsymbol{w}}$, the derivation is given in Appendix A.3.1. We expand Fisher Score to multiple classes by considering a weighted sum of Fisher Score of each pair of classes as follows:

$$FS(X) = \sum_{i\neq j} \beta_{ij} J(X^{(i)}, X^{(j)}), \quad \beta_{ij} = \frac{n_i + n_j}{(C-1)\sum_k n_k}$$

where n_c is the number of nodes in class c. In the implementation, when the correlation between most pair of features are weak, i.e. most non-diagonal entries in the co-variance matrix are 0, we can keep only the diagonal entries to reduce the huge computation cost brought by the inverse operation.

The Fisher Score can be extended to evaluate non-linearly separable data in addition to linearly separable data. We claim the rationale of such measure by showing that the graph convolution can actually help non-linearly separable data to be linearly separable if the graph filter is chosen properly for a given graph. We give examples on graphs with circular feature distributions to demonstrate this conclusion in Appendix A.3.2. We find 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 [24], which removes all the non-linear activations in the original GCN architecture.

Graph Filter Discriminant Score. As mentioned before, the key component that empowers GNNs is the graph filter $\mathcal{F}(\mathcal{G})$. Intuitively, based on the above empirical results and since we use a linear classifier in the end for node classification task, an effective filter should make the representations of nodes in different classes more linearly separable. Therefore, we propose to use Fisher Scores of the node representations before and after applying the graph filter in order to evaluate this filter. We define the GFD Score for the filter $\mathcal{F}(\mathcal{G})$ with respect to feature matrix $\boldsymbol{X}$ as:

$$\begin{aligned} GFD\Big(\mathcal{F}(\mathcal{G}), \boldsymbol{X}\Big) &= FS(\mathcal{F}(\mathcal{G})X) - FS(X) \\ &= \sum_{i\neq j} \beta_{ij} J\big((\mathcal{F}(\mathcal{G})\boldsymbol{X})^{(i)}, (\mathcal{F}(\mathcal{G})\boldsymbol{X})^{(j)}\big) - J\big(\boldsymbol{X}^{(i)}, \boldsymbol{X}^{(j)}\big) \end{aligned}$$

The proposed GFD would be a reasonable metric to evaluate a graph filter's effectiveness, and a better graph filter for a given graph should have a higher GFD score on that graph.

4 CASE STUDY: ASSESSING GRAPH FILTER WITH GFD

The GFD Score we introduced in the above section can be applied to any filter on any given graph. With the help of this assessment tool, we now examine some 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 data that will influence the performance of graph filters?*

We find that most of the current GNNs fall 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. We provide a case study by empirically analyzing how the aforementioned graph properties can affect the optimal choice among this family of graph filters. Note that for other variants of GNN filter, the analysis is similar. For simplicity, we study the roles of the normalization strategy ($\hat{\boldsymbol{A}}$) and the order to use (k) separately, using our assessing tool to show if there exists an optimal choice of $\hat{\boldsymbol{A}}$ and k for different graph data. If not exist, we determine the factors that will influence the choice.

Through the analysis, we use the SBM and DCSBM to generate the structures of synthetic graphs, and use multivariate Gaussian distributions to generate the node features. Without lost of generality, we focus on the two-class classification. We generate graphs with different properties introduced in Section 2 that are controlled by the following hyper-parameters: mean $\mu^{(c)}$ and covariance matrix $\Sigma^{(c)}$ for node feature in class c, class size n_c, internal density p, external density q and the power-law coefficient γ. Our generated synthetic graphs can cover a large range of possible graph properties, and are representative for analyzing different filters.

Analyzing Filter's Normalization Strategy We consider three normalization ways, 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 calculate GFD scores of these three graph filters for graphs generated with different parameters. As shown in Fig. 2, each normalization strategy may outperform others for some graphs, thus,

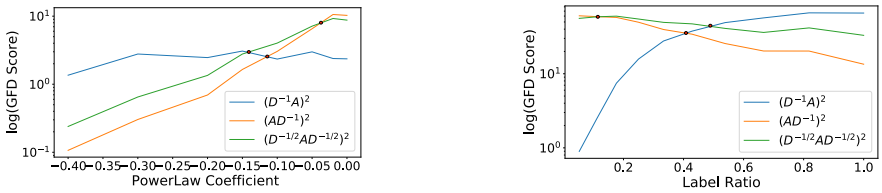


Figure 2: How power law coefficient and label ratio influence the choice of normalization strategy. Parameters of graph generator are given in A.4.3

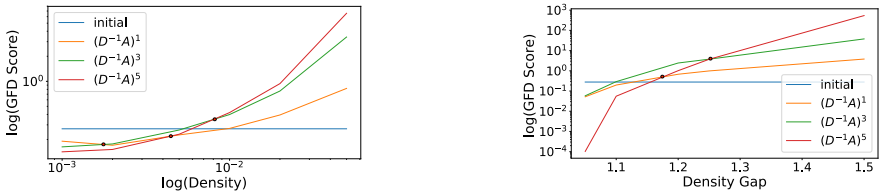


Figure 3: How density and density gap influence the choice of order. Parameters of graph generator are given in A.4.3

we have the conclusion that *no single normalization strategy is optimal for all graphs.*[1]

Note that, with the same order, each filter has the same receptive field, and different normalization strategies affect only on how to assign weights to the neighboring nodes. The row normalization strategy simply takes the mean of features of the node's neighbors. For node v, its representation after the filter is $(\mathcal{F}(\mathcal{G})\boldsymbol{X})_v = \sum_{u\in\mathcal{N}_v} \frac{\boldsymbol{X}_u}{d_v}$. Clearly, this would help to keep every node's new representations in the same range. Using a column-normalization strategy is similar to the PageRank algorithm. While a node propagates its features to neighbors, this normalization strategy takes its degree into account, and may keep a larger representation for higher-degree nodes. Thus, column normalization can be helpful when the when node degree plays an important role for classification. Symmetric normalization combines the properties from both the row and the column normalization. Even in the case where row and column normalization do not perform well, symmetric normalization may still lead to promising performance. We then examine which graph properties influence our choice of the optimal normalization strategy.

Power-law Coefficient (γ) is an important factor that influences the choice of optimal normalization strategy. As shown in Fig. 2, when power-law coefficient γ decreases, row normalization's performance gradually exceed the performance of others. This is because row normalization helps to keep node representations in the same range, so that large representations of high degree nodes can be avoided. Therefore, it prevents nodes with similar degrees getting closer to each other and avoids messing the classification tasks where node degrees are not important.

Label Ratio $(\frac{n_1}{n_2})$ also has big effects. According to Fig. 2, we conclude that when the size of each class becomes more imbalanced, column normalization tends to be helpful. This is because column normalization better leverages degree property during representation smoothing, the representations obtained after propagation are no longer within a same range, and the higher-degree nodes may have larger representations. With column normalization, nodes in large-size classes tend to have larger representation since they are more likely to have higher degree. This can help nodes within different classes become more separable.

Analyzing Filter's Order We then analyze what would be the best order for filters. With a high-order filter, a node can obtain information from its further neighbors, and thus the amount of information it receives during the feature propagation increases. But do we always need more information under any circumstances? The answer is no. We find that, *for different graphs, the order that results in the best performance would be different, but should be in a reasonable range*[1]. We then explore the factors that influence the choice of order. As pointed out in [15], when the order is too big, all the node representations will converge after feature propagation and the classification would be even more difficult, so we only consider orders within a small range in the following.

Density $(\frac{(n_1^2+n_2^2)p+2n_1n_2q}{(n_1+n_2)^2})$ influences the choice of optimal order a lot. As shown in Fig. 3, for orders within reasonable range, when the density grows, filters with higher order tend be better. Note that the feature propagation scheme is based on the assumption that nodes in the same class have a closer connection. So, when the density increases, the connections between nodes get closer. Therefore, high-order filters can help gather richer information and thus reduce the variance of the node's new representations in same class, i.e. it helps same-class nodes get smoother representations.

Density Gap (p/q) also has a big impact. According to Fig. 3, for orders within reasonable range, when the density gap increases, higher-order filters tend to be preferred. This is because when the density gap decreases, for a node, the size of same-class neighbors becomes similar to the size of different-class neighbors. Thus high-order graph convolution operations will mix the representations of all nodes regardless of classes and make node classification more difficult. So for graphs with a small density gap, low-order filters are preferred. An extreme case would be when the density gap is

[1]We also provide examples in A.4 to illustrate that each normalization and order will outperform others in some specific graph data.

close to 1, it indicates the graph structure and node class are nearly independent, then, we should use identity matrix (i.e. order=0) as graph filter and depend fully on the node features instead of both graph structure and node features.

5 LEARNING TO FIND THE OPTIMAL GRAPH FILTER

Now we know that using different graph filters for different graphs is important, then let's consider: *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, AFGNNs can learn to combine an effective filter from a set of filter bases, guided by GDF Scores.

Adaptive Filter Graph Neural Network (AFGNN). For simplicity, we only consider finding the optimal filter for one 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. Note that, we also include the identity matrix, which serves as a skip-connection, to maintain the original feature representation. 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 of AFGNN is defined as a learnable linear combination of these filter bases:

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

where $\psi^{(l)}$ is the learnable vector to combine base filters and $\alpha^{(l)}$ is its normalized version. This formula can also be interpreted as a polynomial graph spectral filter with diverse graph laplacians [5].

Comparing to GNNs with fixed filters, AFGNN can adaptively learn a filter based on any given graph. As no single fixed filter can perform optimally for all graphs, an adaptive filter should have more capacity to learn better representations. Compared to other GNNs with learnable filters such as GAT, AFGNN would be computationally cheaper and can achieve similar or better performance on most datasets. If adding more complex filters into the filter base family, AFGNN can be more powerful.

Training Loss. There are three ways for training, which leads to the following varieties of AFGNN:

AFGNN$_{CE}$: We can simply optimize the model via the downstream tasks, i.e., node classification. But as most of the semi-supervised node classification datasets only contain limited training data, the enlarged filter space will make the model prone to over-fitting.

AFGNN$_{\lambda}$: We consider including the GFD as an extra loss term:

$$\mathcal{L} = (1-\lambda)\mathcal{L}_{CE} + \lambda\mathcal{L}_{GFD}$$

where $\mathcal{L}_{GFD} = -\sum_{l=1}^{L} GFD\big(\mathcal{F}_{AFGNN}(\mathcal{G})^{(l)}, H^{(l-1)}\big)$ 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 $H^{(l-1)}$, $\mathcal{L}_{CE}$ is the cross-entropy loss of the node classification, and $\lambda \in [0, 1]$ trades off the two loss components. For this case, the model is trained by classification loss and GFD loss simultaneously, and thus can learn a better filter that combines the base filters. Since GFD is mainly affected by the feature propagation, so we mainly want to use GFD as a guidance to optimize the filter weights and do not count on it much to optimize linear transformation weights.

AFGNN$_{pr}$: We can also use the GFD loss to pretrain the graph filters. Since the main purpose we introduced the GFD loss is to guide the learning of the combination weight of the graph filters, we can first leverage only the GFD loss $\mathcal{L}_{GFD}$ to pretrain the graph filter, then we optimize the linear transformation parameter $\boldsymbol{W}^{(l)}$s and finetune the graph filter with classification loss $\mathcal{L}_{CE}$.

6 EXPERIMENTS

In this part, we assess if AFGNN can learn powerful filters and perform well for node classification.

Datasets We first evaluate AFGNN on five widely used benchmark datasets: Cora, Citeseer, Pubmed [21], Brazil Air Traffic, and Europe Air Traffic [19]. Among them, we found Cora, Citeseer and Pubmed are not sensitive enough to differentiate the models, so we also generate some synthetic datasets that can better evaluate the pros and cons of each model. With previous analysis, we generate two synthetic datasets: SmallGap and SmallRatio. SmallGap is for the case in which the density gap of the graph is close to 1. This indicates the graph structure does not correlate much to the task, thus we should trust the features more. SmallRatio corresponds to the case in which the dataset is imbalanced, i.e. the size of one class is clearly smaller than the other, and filter $\mathcal{F}(\mathcal{G}) = (\boldsymbol{A}\boldsymbol{D}^{-1})^2$ is the best. The properties of these two synthetic datasets may widely occur in real-world datasets, we generate a sampled version of Open Academic Graph called "OAG Sampled" to justify hard cases may exist in real-world datasets. Detailed description and statistics of all the datasets are shown in A.5 .

Baselines and Settings. We compare against 5 baselines, including GCN, GIN, SGC, GFNN, and GAT. To make fair comparisons, for all the baselines, we set the number of layers (or orders) to be 2.

For all the benchmark datasets, we follow the data split convention [12, 25]. For the synthetic dataset and the OAG Sampled dataset, we conduct 5-fold cross-validation. Each time we pick the model with highest validation accuracy and record its test accuracy. For each dataset partition, we run the experiment 10 times and compute the mean and standard deviation of recorded test accuracy. Details about data split and experiment settings are provided in A.6

Hyper Parameter Tuning for AFGNN$_{\lambda}$ For AFGNN$_{\lambda}$, we do a grid search for the best λ from 0 to 1, details can be found in A.7 and the performance of AFGNN$_{\lambda}$ is shown in Fig. 7. We denote by AFGNN$_{\lambda^*}$ the model with the best λ^* we found in later parts.

Classification Performance. As is shown in Table 1, our proposed AFGNN model can consistently achieve competitive test accuracy. On Pubmed, Brazil, Europe, SmallGap, SmallRatio, and OAG Sampled, AFGNN$_{pr}$ can achieve the best results among all the baseline models. On Citeseer , AFGNN$_{\lambda^*}$ is the best among all the models. On Cora, though GAT outperforms our proposed model, however, as shown in Table 5 in A.8, GAT takes a longer time to train and converge, and has more memory cost. Also, when the given graph is simple, GAT would suffer unavoidable overfitting.

We further compare AFGNN$_{CE}$, AFGNN$_{\lambda^*}$, AFGNN$_{pr}$ to examine the role of GFD loss. The AFGNN$_{CE}$ performs quite poorly on all datasets, implying that the larger search space of the filter without GFD loss is prone to over-fitting, while AFGNN$_{\lambda^*}$ and AFGNN$_{pr}$ perform much better. Also, AGFNN$_{pr}$ tends to perform better on most datasets compared to AFGNN$_{\lambda^*}$, this indicates pretraining

Dataset	GCN	GIN	SGC	GFNN	GAT	**AFGNN$_{CE}$**	**AFGNN$_{\lambda^*}$**	**AFGNN$_{pr}$**
Cora	80.85±0.43	76.37±0.75	81.14±0.05	80.42±0.70	**82.90± 0.01**	60.70±1.86	81.03±0.42	81.54 ±0.70
Citeseer	71.19±0.60	67.85±0.52	71.91±0.01	71.15±0.55	72.20 ±0.07	60.93±0.67	**72.57±0.79**	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	74.11±0.80	78.86±0.16	**79.20±0.01**
Brazil	45.32 ± 8.15	36.63±8.08	58.41±8.42	61.86±6.30	34.80± 10.81	48.00±12.97	60.02±14.20	**62.14± 8.39**
Europe	49.70 ± 4.60	31.95±5.82	54.29 ± 3.56	56.46 ± 3.66	43.01 ± 5.73	43.86 ± 11.00	55.23 ± 8.30	**57.98 ± 4.68**
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**
OAG Sampled	91.77±5.67	85.16±4.58	95.6±4.15	91.63±5.74	95.05±4.36	89.82±5.70	87.90±7.19	**96.34±0.45**

Table 1: Test accuracy of different models. The best result is bold, the second best is underlined

with GFD can help learn a better graph filter so the GFD Score is indeed a powerful tool for assessing the effectiveness of the filter.

7 CONCLUSION

Understanding the graph filters in GNNs is very important, as it can help to determine whether a GNN will work on a given graph, and can provide important guidance for GNN design. In this paper, we first propose the Graph Filter Discriminant Score as an assessment tool for graph filter evaluation, and then apply it to analyze a family of existing filters. We find that no single fixed filter can produce optimal results on all graphs so an adaptive graph filter would be more powerful. We then develop a simple but effective model: AFGNN, which can learn to combine a family of filters and obtain a task-specific filter. We also propose to add the negative GFD Score as an extra term to the objective function, it can act as a guidance for the model to learn a more powerful filter. Experiments show that our approach outperforms many existing GNNs on both benchmark and synthetic graphs. Future works can be focused on enlarging the filter family to further enhance the performance.

REFERENCES

[1] Yunsheng Bai, Hao Ding, Song Bian, Ting Chen, Yizhou Sun, and Wei Wang. 2019. SimGNN: A Neural Network Approach to Fast Graph Similarity Computation. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining, WSDM 2019, Melbourne, VIC, Australia, February 11-15, 2019*. 384–392. https://doi.org/10.1145/3289600.3290967

[2] Jie Chen, Tengfei Ma, and Cao Xiao. 2018. FastGCN: Fast Learning with Graph Convolutional Networks via Importance Sampling. *CoRR* abs/1801.10247 (2018). arXiv:1801.10247 http://arxiv.org/abs/1801.10247

[3] Jianfei Chen, Jun Zhu, and Le Song. 2018. 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*. 941–949. http://proceedings.mlr.press/v80/chen18p.html

[4] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. 2019. 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*. 257–266.

[5] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in neural information processing systems*. 3844–3852.

[6] Nima Dehmamy, Albert-László Barabási, and Rose Yu. 2019. Understanding the Representation Power of Graph Neural Networks in Learning Graph Topology. In *Advances in Neural Information Processing Systems*. 15387–15397.

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

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

[9] David K Hammond, Pierre Vandergheynst, and Rémi Gribonval. 2011. Wavelets on graphs via spectral graph theory. *Applied and Computational Harmonic Analysis* 30, 2 (2011), 129–150.

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

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

[12] Thomas N. Kipf and Max Welling. 2017. 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*. https://openreview.net/forum?id=SJU4ayYgl

[13] Johannes Klicpera, Stefan Weißenberger, and Stephan Günnemann. 2019. Diffusion Improves Graph Learning. In *Advances in Neural Information Processing Systems*. 13333–13345.

[14] Ron Levie, Federico Monti, Xavier Bresson, and Michael M Bronstein. 2018. Cayleynets: Graph convolutional neural networks with complex rational spectral filters. *IEEE Transactions on Signal Processing* 67, 1 (2018), 97–109.

[15] Qimai Li, Zhichao Han, and Xiao-Ming Wu. 2018. Deeper insights into graph convolutional networks for semi-supervised learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

[16] Yujia Li, Chenjie Gu, Thomas Dullien, Oriol Vinyals, and Pushmeet Kohli. 2019. Graph Matching Networks for Learning the Similarity of Graph Structured Objects. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*. 3835–3845. http://proceedings.mlr.press/v97/li19d.html

[17] Renjie Liao, Zhizhen Zhao, Raquel Urtasun, and Richard S Zemel. 2019. Lanczosnet: Multi-scale deep graph convolutional networks. *arXiv preprint arXiv:1901.01484* (2019).

[18] Takanori Maehara. 2019. Revisiting Graph Neural Networks: All We Have is Low-Pass Filters. *arXiv preprint arXiv:1905.09550* (2019).

[19] Leonardo FR Ribeiro, Pedro HP Saverese, and Daniel R Figueiredo. 2017. struc2vec: Learning node representations from structural identity. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 385–394.

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

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

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

[23] WOK Asiri Suranga Wijesinghe and Qing Wang. 2019. DFNets: Spectral CNNs for Graphs with Feedback-Looped Filters. In *Advances in Neural Information Processing Systems*. 6007–6018.

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

[25] Jun Wu, Jingrui He, and Jiejun Xu. 2019. Net: Degree-specific graph neural networks for node and graph classification. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 406–415.

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

[27] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019. How Powerful are Graph Neural Networks?. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. https://openreview.net/forum?id=ryGs6iA5Km

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

A APPENDIX

A.1 Summary of Graph Filters for Some Existing GNNs.

The work of GCN [12] use the filter $\mathcal{F}(\mathcal{G}) = \tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}$ where $\tilde{A}$ = A+I is the self-augmented adjacency matrix, and $\tilde{D} = \text{diag}(\tilde{d}_1, ..., \tilde{d}_n)$ is the corresponding degree matrix with $\tilde{d}_i = \sum_{j=1}^{n} \tilde{A}_{ij}$.

SGC and GFNN [18, 24] use the filter $\mathcal{F}(\mathcal{G}) = (\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2})^k$ that includes a pre-defined exponent k.

Some works such as GIN [27] propose to use $\mathcal{F}(\mathcal{G}) = A + \epsilon I$ with a learnable parameter ϵ to augment self-loop skip connection.

GAT[22] proposes to assign attention weight to different nodes in a neighborhood, which can be considered as a flexible learnable graph convolutional filter that is a parametric attention function of X and A.

Table 2 summarized the graph filters for some existing GNNs. Those GNNs are regard as baselines for our work.

Table 2: A Summary of Graph Filters of Baseline GNNs. ϵ is a learnable scaler, k is a pre-defined hyper parameter, $\mathcal{N}_i$ is the neighborhood of node i, α is a learnable weight vector, and || indicates concatenation.

GNNs	Graph Convolutional Filters
GCN [12]	$\mathcal{F}(\mathcal{G}) = \tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}$
SGC [24]	$\mathcal{F}(\mathcal{G}) = (\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2})^k$
GFNN [18]	$\mathcal{F}(\mathcal{G}) = (\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2})^k$
GIN [27]	$\mathcal{F}(\mathcal{G}) = A + \epsilon I$
GAT [22]	$\forall i, j, \mathcal{F}(\mathcal{G})_{ij} = \frac{exp(\sigma(\boldsymbol{\alpha}[WX_i \| WX_j]))}{\sum_{k\in\mathcal{N}_i} exp(\sigma(\boldsymbol{\alpha}[WX_i \| WX_k]))} A_{ij}$

A.2 Graph Properties

We then investigate some key properties of graph data to characterize different graphs as follows.

Density of Graph. The *"density of graph"* is the ratio of the number of existing edges and the number of node pairs in a graph: $\frac{(n_1^2+n_2^2)p+2n_1n_2q}{(n_1+n_2)^2}$, it controls the overall connectivity of the graph.

Density Gap. We call p/q the *"density gap"*, which controls how closely the graph generated by SBM correlates with labels.

Power-law Coefficient. The *"power-law coefficient"* γ in DCSBM is to control the degree distribution among nodes, it usually is within range $(-1, 0)$. With a small γ is, the exponent of power-law would be big. When $\gamma = 0$, DCSBM is exactly same as SBM.

Label Ratio. Given a pair of classes (i, j), we care about its label ratio, which is defined as n_i/n_j, without lost of generality, we assume the class size satisfies $n_j \geq n_i$, then the label ratio should be within range $(0, 1]$, the smaller the label ratio is, the more imbalanced the classes are.

Feature Statistics. Given a class of nodes, we care about the *"mean"* and *"co-variance matrix"* of node features. These feature statistics would also be important for our later analysis.

Among them, density of graph, density gap, and power-law coefficient are closely related to A, label ratio is closely related to Y, and feature statistics is closely related to X.

A.3 Details About GFD

A.3.1 Closed Form Solution of Fisher Score.

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

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

□

Thus we have $J(\boldsymbol{X}^{(i)}, \boldsymbol{X}^{(j)}) = (\boldsymbol{\mu}^{(i)} - \boldsymbol{\mu}^{(j)})^\top(\Sigma^{(i)}+\Sigma^{(j)})^{-1\top}(\boldsymbol{\mu}^{(i)} - \boldsymbol{\mu}^{(j)})$.

A.3.2 Fisher Score for Non-linearly Separable Data Evaluation.

As shown in Figure 4(a)~(d), if we use a proper filter, the convolutional operation can transform three circular distributions, which are non-linearly separable, into three linearly separable clusters. Moreover, as shown in Figure 4(e)~(h), even if the original features of different classes are sampled from the same distribution, the proper graph filter can help to linearly separate the data. This phenomenon shows that if the graph structure ($\mathcal{G}$) is correlated with the task (Y), a proper filter alone is powerful enough to empower GNNs with non-linearity, without any non-linear activation.

A.3.3 Illustration of Graph Generator for Figure 4.

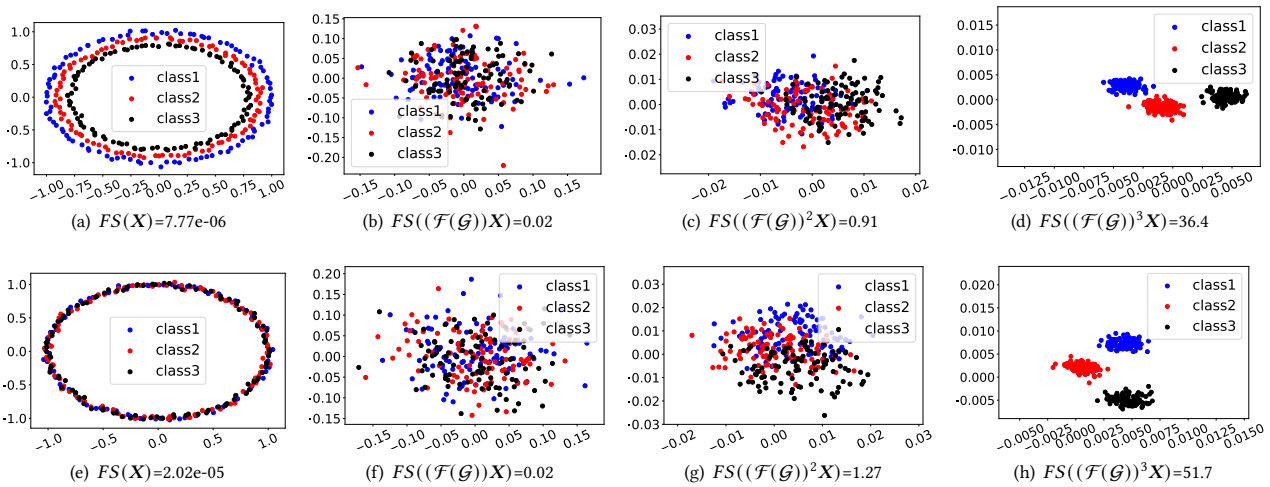
For Figure 4, both graphs include three classes of the same size and has structure generated by SBM with $p = 0.6$ and $q = 0.03$. The first graph's feature follows a circular distribution with radius 1,0.9,0.8 and Gaussian noise 0.02 for each class. The second graph's feature follows a circular distribution with radius 1 and Gaussian noise 0.02 for the three classes.

A.4 No Best Filter for All Graphs

A.4.1 Examples of "No Best Normalization Strategy for All".

Figure 5 provides two examples to show there is no best normalization strategy for all graphs. For both examples, we fix the order to be 2.

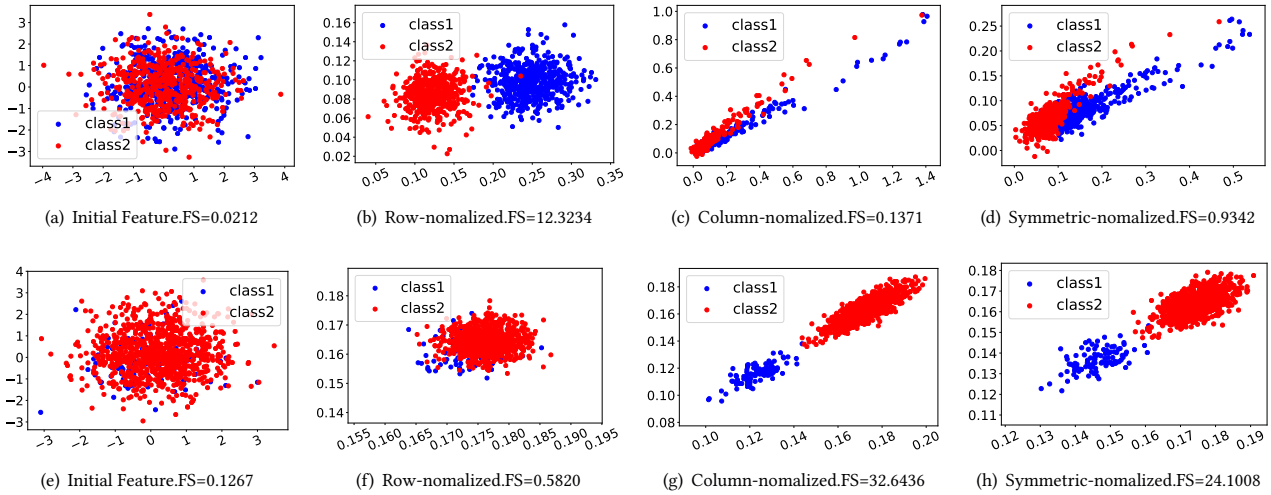
The first row shows a case in which row normalization is better than the other two. The corresponding graph contains 2 classes of nodes with size 500. The graph structure is generated by DCSBM with $p = 0.3$, $q = 0.05$, power law coefficient $\gamma = -0.9$. The features for two classes satisfy multivariate 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



(a) $FS(\boldsymbol{X})$=7.77e-06 (b) $FS((\mathcal{F}(\mathcal{G}))\boldsymbol{X})$=0.02 (c) $FS((\mathcal{F}(\mathcal{G}))^2\boldsymbol{X})$=0.91 (d) $FS((\mathcal{F}(\mathcal{G}))^3\boldsymbol{X})$=36.4

(e) $FS(\boldsymbol{X})$=2.02e-05 (f) $FS((\mathcal{F}(\mathcal{G}))\boldsymbol{X})$=0.02 (g) $FS((\mathcal{F}(\mathcal{G}))^2\boldsymbol{X})$=1.27 (h) $FS((\mathcal{F}(\mathcal{G}))^3\boldsymbol{X})$=51.7

Figure 4: Each row corresponds to a graph, $\mathcal{F}(\mathcal{G}) = \tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}$. The Fisher Score is provided in each sub-figure's caption. The graph generator parameters are given in A.3.3



(a) Initial Feature.FS=0.0212 (b) Row-nomalized.FS=12.3234 (c) Column-nomalized.FS=0.1371 (d) Symmetric-nomalized.FS=0.9342

(e) Initial Feature.FS=0.1267 (f) Row-nomalized.FS=0.5820 (g) Column-nomalized.FS=32.6436 (h) Symmetric-nomalized.FS=24.1008

Figure 5: Examples of "No Best Normalization Strategy for All"

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.4.2 Examples of "No Best Order for All". Figure 6 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.4.3 Illustration of Graph Generator for Curves in Section 4. For the curves indicating how powerlaw coefficient influence the choice of normalization in Figure 2, we generate the graph structure by DCSBM with fixed $p = 0.3$, $q = 0.2$ (so the density and density gap are also fixed) and with power-law coefficient γ varies from -0.4 to 0. The graph contains two classes of nodes, with size 300 and 700 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). We compare the filters with different normalization strategy but with same order 2.

For the curves indicating how label ratio influence the choice of normalization in Figure 2, we generate a set of graphs with graph structure generated by SBM with fixed density $\frac{(n_1^2+n_2^2)p+2n_1n_2q}{(n_1+n_2)^2} = 0.4$ and density gap $p/q = 0.2$, note that the corresponding p and q for each graph would be different when $n1/n2$ changes. The graph contains two classes of nodes, with total size 1000, but the label ratio varies from 0 to 1. The feature for each class satisfies multivariate normal distribution with identity co-variance matrix, and with mean (0,0) and (0.5,0.5). We compare the filters with different normalization strategy but with same order 2.

For the curves indicating how density influence the choice of normalization in Figure 3, we generate a set of graphs with the graph structure generated by SBM with fixed density gap $p/q = 1.5$ and with density varies from 0.0025 to 0.5. The graph contains two classes of nodes, both with 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). We compare the filters all with row normalization but with different orders.

For the curves indicating how density gap influence the choice of normalization in Figure 3, we generate a set of graphs with the graph structure generated by SBM with fixed density 0.25 and with density gap varies from 1 to 3. The graph contains two classes of nodes, both with 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). We compare the filters all with row normalization but with different orders.

A.5 Details About Dataset

A.5.1 Benchmark Dataset. We use five benchmark dataset: Cora, Citeseer, Pubmed, Brazil Air Traffic, and Europe Air Traffic for the node classification task.

Cora, Citeseer, Pubmed are the most widely used benchmark dataset for node classification. They are citation networks, where each node represents a document, each edge is a citation link, and the node feature is a sparse bag-of-words feature vector. The class of a node is the field that this document belongs to.

Brazil and Europe are two Air-Traffic networks that are also popular for node classification task. Each node represents an airport, and each edge indicates the existence of commercial flights between the airports. The class of a node is given based on the level of activity measured by flights or people that passed the airports. These two datasets do not have node attributes, so follow the work of [25], we use the one-hot encoding of node degrees.

Table 3: Statistics of Benchmark Dataset

Dataset	**Cora**	**Citeseer**	**Pubmed**	**Brazil**	**Pubmed**
Nodes	2708	3327	19717	131	399
Edges	5429	4732	44338	1038	5995
Classes	7	6	3	4	4
Feature	1433	3703	500	-	-

The detailed statictics of theses datasets are shown in table3.

A.5.2 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 parameters are given in our code.

A.5.3 OAG Small Ratio Dataset. We generate a real-world dataset with imbalanced classes to justify hard cases may exist in real-world datasets. We download a large scale academic graph called Open Academic Graph (OAG), and choose two fields that have a large disparity in the number of papers: (1) "History of ideas", which consists of 1041 papers; (2) "Public history", which consists of 150 papers. Obviously this two classes are imbalanced, and fall in the large label ratio gap problem. We run supplementary experiment on the generated OAG graph, the experiment setting remains the same as experiment settings for synthetic graphs.

A.6 Details About Experiment Setting

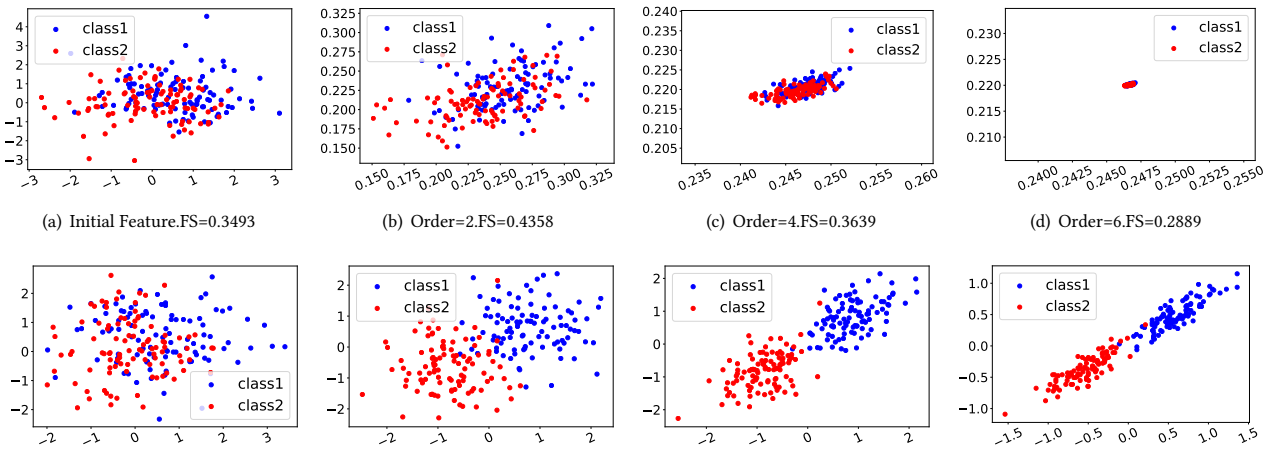
A.6.1 Data Split. For Cora, Citeseer and Pubmed, follow the data split convention in the work of [12], we take 20 nodes in each class to form the training set, take 500 nodes to form validation set, and take 1000 nodes to form test set.

For Brazil and Europe, follow the data split convention in the work of [25], the training, validation and test sets are randomly assigned with equal number of nodes. We generate 10 different partitions for each dataset.

For the synthetic dataset and the OAG Sampled dataset, we conduct 5-fold cross-validation. we randomly split the nodes into 5 groups of the same size, take one group as the training set, one as the validation set and the remaining three as the test set.

We run the experiment 10 times for each partition of each dataset, and record the mean and standard deviation for each dataset.

A.6.2 Model Configuration and Hyper Parameters. For GCN, SGC, GFNN, GAT, we follow their public implementations. For GIN, the initial code is not for node classification task, so we implement the model following [27] to conduct experiments.



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

(e) Initial Feature.FS=0.3691 (f) Order=2.FS=4.6346 (g) Order=4.FS=11.5741 (h) Order=6.FS=14.7328

Figure 6: Examples of "No Best Order for All"

Table 4: Baseline's Accuracy on Benchmark Dataset Reported by Literature

	Cora	Citeseer	Pubmed	Brazil	Europe
GCN	81.5	70.3	79.0	43.2±6.4	37.1± 4.6
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	–	–
GAT	83.0±0.7	72.5±0.7	79.0±0.3	38.3±12.6	42.4 ± 7.3

Table 5: Time Cost

	Cora			Citeseer			Pubmed		
	time	num	total	time	num	total	time	num	total
$AFGNN_0$	0.055	96.5	5.309	0.116	117.5	13.579	0.376	137.0	51.456
$AFGNN_1$	0.086	129.3	11.1	0.136	155.4	21.177	0.379	136.1	51.62
$AFGNN_\infty$(filter)	0.106	53	5.593	0.146	48	7.023	0.377	200	75.456
$AFGNN_\infty$(classification)	0.005	200	0.914	0.006	400	2.293	0.006	400	2.246
$AFGNN_\infty$(overall)	-	-	6.507	-	-	9.315	-	-	77.702
GAT	0.156	382.8	59.625	0.168	379.3	63.462	-	-	-

Table 6: Memory(MB) Cost

	Cora	Citeseer	Pubmed
$AFGNN_0$	861	1369	1351
$AFGNN_1$	863	1369	1351
$AFGNN_\infty$	863	1369	1351
GAT	1733	2345	-

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.7 Hyper Parameter Tuning for $AFGNN_\lambda$

For $AFGNN_\lambda$, we do a grid search for the best λ from 0 to 1, we take 0.05 as interval. We take Cora, Citeseet and Pubmed as examples to show the grid search results. For each dataset with a certain λ, we follow the datasplit convention, run the experiment 10 times, and record the mean and standard deviation of the classification accuracy. The results are shown in Figure 7. We denote the best λ as λ^*, and use the corresponding $AFGNN_{\lambda^*}$ in the later classification performance comparison.

A.8 Baseline Accuracy on Benchmark Dataset

We report the accuracy of node classification task for baseline models on Cora, Citeseer, Pubmed, Brazil, and Europe provided by corresponding literature. Since GIN [27] is not originally evaluated on node classification task, we do not have the reported number here. The results is in Table 4.

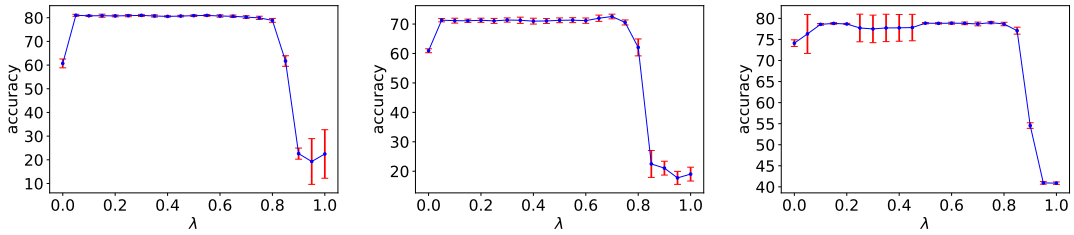
A.9 Time and Memory Cost Comparison

Both our AFGNN model and GAT model have a learnable filter. We provide time and memory complexity comparison on benchmark datasets here to compare these two models.

As shown in Table 5, GAT's time cost is at least three times of AFGNN's time cost on both Cora and Citeseer dataset. As shown in Table 6, AFGNN's memory cost on both Cora and Citeseer are half of GAT's memory cost. GAT does not have recorded time and memory cost for Pubmed dataset because it requires too much memory cost and is not able to run on GPU. Therefore, AFGNN needs less time and memory cost than GAT.

A.10 Graph Filter Discriminant Analysis.

We are also interested to see if the proposed method can indeed learn the best combination of filters from the base filter family. To do so, we take 3 benchmark dataset and 2 synthetic dataset as examples, calculate the GFD Score of filter learned by $AFGNN_{CE}$, $AFGNN_{\lambda^*}$,



(a) Cora (b) Citeseer (c) Pubmed

Figure 7: Performance of AFGNN$_\lambda$ with different λ. For Cora, the best λ is 0.05, for Citeseer, it is 0.7, and for Pubmed, it is 0.5.

Filters	Cora	Citeseer	Pubmed	SmallGap	SmallRatio
$\boldsymbol{I}$	0	0	0	0	0
$\tilde{D}^{-1}\tilde{A}$	20.53	23.19	10.3	-33.16	2.63
$(\tilde{D}^{-1}\tilde{A})^2$	**45.24**	42.26	25.95	-15.96	6.88
$\tilde{A}\tilde{D}^{-1}$	15.20	18.94	0.69	-33.16	7.05
$(\tilde{A}\tilde{D}^{-1})^2$	36.40	36.88	10.99	-17.86	**76.75**
$\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}$	18.72	22.15	8.34	-33.16	4.89
$(\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2})^2$	41.52	40.23	22.86	-16.55	46.07
AFGNN$_{CE}$	5.08	6.33	0.72	0.16	4.88
AFGNN$_{\lambda^*}$	**45.00**	**42.32**	**26.56**	**0.46**	74.19
AFGNN$_{pr}$	**45.00**	**42.63**	**26.56**	**0.55**	**76.75**

Table 7: GFD after applying different filters on benchmark and synthetic datasets.

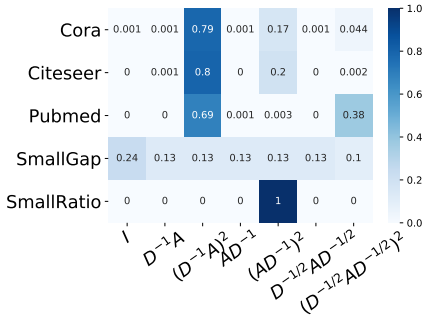


Figure 8: Base filter combination Learned by AFGNN$_{pr}$

AFGNN$_{pr}$ and the seven base filters on the test set for each of these datasets. The results shown in Table 7 and Figure **??** show that our proposed method can indeed learn a powerful combined filter on all the datasets. Specifically, our proposed adaptive filter not only can pick out the best base filter but can even learn a better combination. We thereby conclude that the proposed GFD loss can help find an appropriate filter for a given dataset.