Revisiting Graph Neural Networks:
All We Have is Low-Pass Filters

Hoang NT
Tokyo Institute of Technology, RIKEN
hoang.nguyen.rh@riken.jp

Takanori Maehara
RIKEN
takanori.maehara@riken.jp

Abstract

Graph neural networks have become one of the most important techniques to solve machine learning problems on graph-structured data. Recent work on vertex classification proposed deep and distributed learning models to achieve high performance and scalability. However, we find that the feature vectors of benchmark datasets are already quite informative for the classification task, and the graph structure only provides a means to denoise the data. In this paper, we develop a theoretical framework based on graph signal processing for analyzing graph neural networks. Our results indicate that graph neural networks only perform low-pass filtering on feature vectors and do not have the non-linear manifold learning property. We further investigate their resilience to feature noise and propose some insights on GCN-based graph neural network design.

1 Introduction

Graph neural networks (GNN) belong to a class of neural networks which can learn from graph-structured data. Recently, graph neural networks for vertex classification and graph isomorphism test have achieved excellent results on several benchmark datasets and continuously set new state-of-the-art performance [1, 15, 24, 27]. Started with the early success of ChebNet [5] and GCN [15] at vertex classification, many variants of GNN have been proposed to solve problems in social networks [11, 29], biology [24, 25], chemistry [7, 9], natural language processing [2, 30], computer vision [19], and weakly-supervised learning [8].

In semi-supervised vertex classification, we observe that the parameters of a graph convolutional layer (GCN) [15] only contribute to overfitting. Similar observations have been reported in both simple architectures such as SGC [26] and more complex ones such as DGI [25]. Based on this phenomenon, Wu et al. [26] proposed to view graph neural networks simply as feature propagation and propose an extremely efficient model with state-of-the-art performance on many benchmark datasets. Kawamoto et al. [13] made a related theoretical remark on untrained GCN-like GNNs under graph partitioning settings. From these previous studies, a question naturally arises: *Why and when do graph neural networks work well for vertex classification?* In other words, is there a condition on the vertex feature vectors for graph neural network models to work well even without training? Consequently, can we find realistic counterexamples in which baseline graph neural networks (e.g. SGC or GCN) fail to perform?

In this study, we provide an answer to the aforementioned questions from the graph signal processing perspective [17]. Formally, we consider a semi-supervised learning problem on a graph. Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, each vertex $i \in \mathcal{V}$ has a feature $\mathbf{x}(i) \in \mathcal{X}$ and label $y(i) \in \mathcal{Y}$, where $\mathcal{X}$ is a d-dimensional Euclidean space $\mathbb{R}^d$ and $\mathcal{Y} = \mathbb{R}$ for regression and $\mathcal{Y} = \{1, \ldots, c\}$ for classification. The task is to learn a hypothesis to predict the label $y(i)$ from the feature $\mathbf{x}(i)$. We then characterize the graph neural networks solution to this problem and provide insights to the mechanism underlying the most commonly used baseline model GCN [15], and its simplified variant SGC [26].
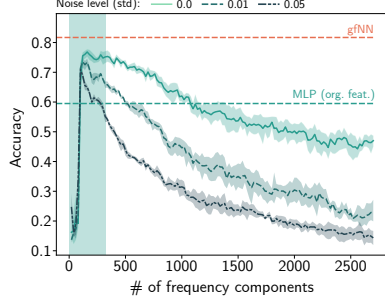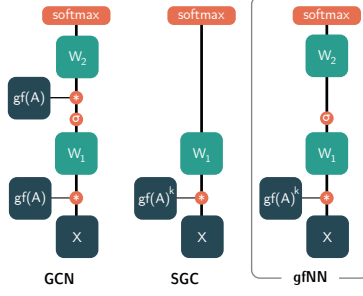
Figure 1: Accuracy by frequency components

Figure 2: A simple realization of gfNN

Graph signal processing (GSP) regards data on the vertices as signals and applies signal processing techniques to understand the signal characteristics. By combining signals (feature vectors) and graph structure (adjacency matrix or its transformations), GSP has inspired the development of learning algorithms on graph-structured data [21]. In a standard signal processing problem, it is common to assume the observations contain some noise and the underlying "true signal" has low-frequency [18]. Here, we pose a similar assumption for our problem.

Assumption 1. *Input features consist of low-frequency true features and noise. The true features have sufficient information for the machine learning task.*

Our first contribution is to verify Assumption 1 on commonly used datasets (Section 3). Figure 1 shows the performance of 2-layers perceptrons (MLPs) trained on features with different numbers of frequency components. In all benchmark datasets, we see that only a small number of frequency components contribute to learning. Adding more frequency components to the feature vectors only decreases the performance. In turn, the classification accuracy becomes even worse when we add Gaussian noise $\mathcal{N}(0, \sigma^2)$ to the features.

Many recent GNNs were built upon results from graph signal processing. The most common practice is to multiply the *(augmented) normalized adjacency* matrix $I - \tilde{\mathcal{L}}$ with the feature matrix $\mathcal{X}$. The product $(I - \tilde{\mathcal{L}})\mathcal{X}$ is understood as features averaging and propagation [11, 15, 26]. In graph signal processing literature, such operation filters signals on the graph without explicitly performing eigendecomposition on the normalized Laplacian matrix, which requires $O(n^3)$ time [23]. Here, we refer to this augmented normalized adjacency matrix and its variants as graph filters and propagation matrices interchangeably.

Our second contribution shows that multiplying graph signals with propagation matrices corresponds to low-pass filtering (Section 4, esp., Theorem 3). Furthermore, we also show that the matrix product between the observed signal and the low-pass filters is the analytical solution to the true signal optimization problem. In contrast to the recent design principle of graph neural networks [1, 15, 27], our results suggest that the graph convolution layer is simply low-pass filtering. Therefore, learning the parameters of a graph convolution layer is unnecessary. Wu et al. [26, Theorem 1] also addressed a similar claim by analyzing the spectrum-truncating effect of the augmented normalized adjacency matrix. We extend this result to show that all eigenvalues monotonically shrink, which further explains the implications of the spectrum-truncating effect.

Based on our theoretical understanding, we propose a new baseline framework named *gfNN (graph filter neural network)* to empirically analyze the vertex classification problem. gfNN consists of two steps: 1. Filter features by multiplication with graph filter matrices; 2. Learn the vertex labels by a machine learning model. We demonstrate the effectiveness of our framework using a simple realization model in Figure 2.

Our third contribution is the following Theorem:

Theorem 2 (Informal, see Theorem 7, 8)**.** *Under Assumption 1, the outcomes of SGC, GCN, and gfNN are similar to those of the corresponding NNs using true features.*

Theorem 7 implies that, under Assumption 1, both gfNN and GCN [15] have similar high performance. Since gfNN does not require multiplications of the adjacency matrix during the learning phase, it is much faster than GCN. In addition, gfNN is also more noise tolerant.

Finally, we compare our gfNN to the SGC model [26]. While SGC is also computationally fast and accurate on benchmark datasets, our analysis implies it would fail when the feature input is nonlinearly-separable because the graph convolution part does not contribute to non-linear manifold learning. We created an artificial dataset to empirically demonstrate this claim.

2 Graph Signal Processing

In this section, we introduce the basic concepts of graph signal processing. We adopt a recent formulation [10] of graph Fourier transform on irregular graphs.

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a simple undirected graph, where $\mathcal{V} = \{1, \ldots, n\}$ be the set of $n \in \mathbb{Z}$ vertices and $\mathcal{E}$ be the set of edges.[1] Let $A = (a_{ij}) \in \mathbb{R}^{n \times n}$ be the adjacency matrix of G, $D = \mathrm{diag}(d(1), \ldots, d(n)) \in \mathbb{R}^{n \times n}$ be the degree matrix of G, where $d(i) = \sum_{j \in \mathcal{V}} a(i, j)$ is the degree of vertex $i \in \mathcal{V}$. $L = D - A \in \mathbb{R}^{n \times n}$ be the combinatorial Laplacian of G, $\mathcal{L} = I - D^{-1/2} A D^{-1/2}$ be the normalized Laplacian of G, where $I \in \mathbb{R}^{n \times n}$ is the identity matrix, and $L_{\mathrm{rw}} = I - D^{-1} A$ be the random walk Laplacian of G. Also, for $\gamma \in \mathbb{R}$ with $\gamma > 0$, let $\tilde{A} = A + \gamma I$ be the augmented adjacency matrix, which is obtained by adding γ self loops to G, $\tilde{D} = D + \gamma I$ be the corresponding augmented degree matrix, and $\tilde{L} = \tilde{D} - \tilde{A} = L$, $\tilde{\mathcal{L}} = I - \tilde{D}^{-1/2} \tilde{A} \tilde{D}^{-1/2}$, $\tilde{L}_{\mathrm{rw}} = I - \tilde{D}^{-1} \tilde{A}$ be the corresponding augmented combinatorial, normalized, and random walk Laplacian matrices.

A vector $x \in \mathbb{R}^n$ defined on the vertices of the graph is called a *graph signal*. To introduce a graph Fourier transform, we need to define two operations, *variation* and *inner product*, on the space of graph signals. Here, we define the variation $\Delta \colon \mathbb{R}^n \to \mathbb{R}$ and the $\tilde{D}$-inner product by

$$\Delta(x) := \sum_{(i,j) \in \mathcal{E}} (x(i) - x(j))^2 = x^\top L x; \quad (x, y)_{\tilde{D}} := \sum_{i \in \mathcal{V}} (d(i) + \gamma) x(i) y(i) = x^\top \tilde{D} y. \quad (1)$$

We denote by $\|x\|_{\tilde{D}} := \sqrt{(x, x)_{\tilde{D}}}$ the norm induced by $\tilde{D}$. Intuitively, the variation Δ and the inner product $(\cdot, \cdot)_{\tilde{D}}$ specify how to measure the smoothness and importance of the signal, respectively. In particular, our inner product puts more importance on high-degree vertices, where larger γ closes the importance more uniformly. We then consider the generalized eigenvalue problem *(variational form)*: Find $u_1, \ldots, u_n \in \mathbb{R}^n$ such that for each $i \in \{1, \ldots, n\}$, u_i is a solution to the following optimization problem:

$$\text{minimize } \Delta(u) \text{ subject to } (u, u)_{\tilde{D}} = 1, \ (u, u_j)_{\tilde{D}} = 0, \ j \in \{1, \ldots, n\}. \quad (2)$$

The solution u_i is called an i-th generalized eigenvector and the corresponding objective value $\lambda_i := \Delta(u_i)$ is called the i-th *generalized eigenvalue*. The generalized eigenvalues and eigenvectors are also the solutions to the following *generalized eigenvalue problem (equation form)*:

$$L u = \lambda \tilde{D} u. \quad (3)$$

Thus, if (λ, u) is a generalized eigenpair then $(\lambda, \tilde{D}^{1/2} u)$ is an eigenpair of $\tilde{\mathcal{L}}$. A generalized eigenvector with a smaller generalized eigenvalue is smoother in terms of the variation Δ. Hence, the generalized eigenvalues are referred to as the *frequency of the graph*.

The graph Fourier transform is a basis expansion by the generalized eigenvectors. Let $U = [u_1, \ldots, u_n]$ be the matrix whose columns are the generalized eigenvectors. Then, the *graph Fourier transform* $\mathcal{F} \colon \mathbb{R}^n \to \mathbb{R}^n$ is defined by $\mathcal{F} x = \hat{x} := U^\top \tilde{D} x$, and the *inverse graph Fourier transform* $\mathcal{F}^{-1}$ is defined by $\mathcal{F}^{-1} \hat{x} = U \hat{x}$. Note that these are actually the inverse transforms since $\mathcal{F} \mathcal{F}^{-1} = U^\top \tilde{D} U = I$.

The *Parseval identity* relates the norms of the data and its Fourier transform:

$$\|x\|_{\tilde{D}} = \|\hat{x}\|_2. \quad (4)$$

Let $h \colon \mathbb{R} \to \mathbb{R}$ be an analytic function. The *graph filter specified by* h is a mapping $x \mapsto y$ defined by the relation in the frequency domain: $\hat{y}(\lambda) = h(\lambda) \hat{x}(\lambda)$. In the spatial domain, the above equation

---

[1] We only consider unweighted edges but it is easily adopted to positively weighted edges.

is equivalent to $y = h(\tilde{L}_{\text{rw}})x$. where $h(\tilde{L}_{\text{rw}})$ is defined via the Taylor expansion of h; see [12] for the detail of matrix functions.

In a machine learning problem on a graph, each vertex $i \in \mathcal{V}$ has a d-dimensional feature $\mathbf{x}(i) \in \mathbb{R}^d$. We regard the features as d graph signals and define the graph Fourier transform of the features by the graph Fourier transform of each signal. Let $X = [\mathbf{x}(1); \ldots, \mathbf{x}(n)]^\top$ be the feature matrix. Then, the graph Fourier transform is represented by $\mathcal{F}X = \hat{X} =: U^\top \tilde{D}X$ and the inverse transform is $\mathcal{F}^{-1}\hat{X} = U\hat{X}$. We denote $\hat{X} = [\hat{\mathbf{x}}(\lambda_1); \ldots; \hat{\mathbf{x}}(\lambda_n)]^\top$ as the frequency components of X.

3 Empirical Evidence of Assumption 1

The results of this paper deeply depend on Assumption 1. Thus, we first verify this assumption in real-world datasets: Cora, Citeseer, and Pubmed [20]. These are citation networks, in which vertices are scientific papers and edges are citations. We consider the following experimental setting: **1**. Compute the graph Fourier basis U from $\tilde{\mathcal{L}}$; **2**. Add Gaussian noise to the input features: $\mathcal{X} \leftarrow \mathcal{X} + \mathcal{N}(0, \sigma^2)$ for $\sigma = \{0, 0.01, 0.05\}$; **3**. Compute the first k-frequency component: $\hat{\mathcal{X}}_k = U[:\ k]^\top \tilde{D}^{1/2}\mathcal{X}$; **4**. Reconstruct the features: $\tilde{\mathcal{X}}_k = \tilde{D}^{-1/2}U[:\ k]\hat{\mathcal{X}}_k$; **5**. Train and report test accuracy of a 2-layers perceptron on the reconstructed features $\tilde{\mathcal{X}}_k$.
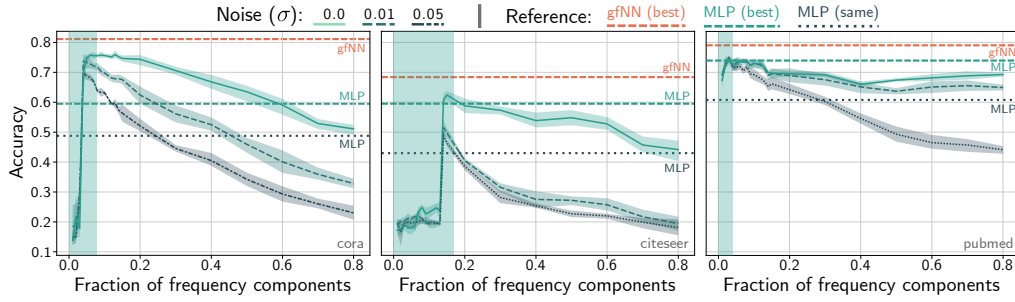

Figure 3: Average performance of 2-layers MLPs on frequency-limited feature vectors (`epochs=20`). The annotated "(best)" horizontal lines show the performance of the best performance using gfNN and a 2-layers MLP trained on the original features.

In Figure 3, we incrementally add normalized Laplacian frequency components to reconstruct feature vectors and train a 2-layers MLPs. We see that all three datasets exhibit a low-frequency nature. The classification accuracy of a 2-layers MLP tends to peak within the top 20% of the spectrum (green boxes). By adding artificial Gaussian noise, we observe that the performance at low-frequency regions is relatively robust, which implies a strong denoising effect.

Interestingly, the performance gap between graph neural network and a simple MLP is much larger when high-frequency components do not contain useful information for classification. In the Cora dataset, high-frequency components only decrease classification accuracy. Therefore, our gfNN outperformed a simple MLP. Citeseer and Pubmed have a similar low-frequency nature. However, the performance gaps here are not as large. Since the noise-added performance lines for Citeseer and Pubmed generally behave like the original Cora performance line, we expect the original Citeseer and Pubmed contain much less noise compared with Cora. Therefore, we could expect the graph filter degree would have little effect in Citeseer and Pubmed cases.

4 Multiplying Adjacency Matrix is Low Pass Filtering

Computing the low-frequency components is expensive since it requires $O(|\mathcal{V}|^3)$ time to compute the eigenvalue decomposition of the Laplacian matrix. Thus, a reasonable alternative is to use a low-pass filter. Many papers on graph neural networks iteratively multiply the (augmented) adjacency matrix $\tilde{A}_{\text{rw}}$ (or $\tilde{A}$) to propagate information. In this section, we see that this operation corresponds to a *low-pass filter*.

Multiplying the normalized adjacency matrix corresponds to applying graph filter $h(\lambda) = 1 - \lambda$. Since the eigenvalues of the normalized Laplacian lie on the interval $[0, 2]$, this operation resembles a *band-*

stop filter that removes intermediate frequency components. However, since the maximum eigenvalue of the normalized Laplacian is 2 if and only if the graph contains a non-trivial bipartite graph as a connected component [4, Lemma 1.7]. Therefore, for other graphs, multiplying the normalized (non-augmented) adjacency matrix acts as a low-pass filter (i.e., high-frequency components must decrease).

We can increase the low-pass filtering effect by adding self-loops (i.e., considering the augmented adjacency matrix) since it shrinks the eigenvalues toward zero as follows.[2]

Theorem 3. *Let $\lambda_i(\gamma)$ be the i-th smallest generalized eigenvalue of $(\tilde{D}, L) = (D + \gamma I)$. Then, $\lambda_i(\gamma)$ is a non-negative number, and monotonically non-increasing in $\gamma \geq 0$. Moreover, $\lambda_i(\gamma)$ is strictly monotonically decreasing if $\lambda_i(0) \neq 0$.*

Note that γ cannot be too large. Otherwise, all the eigenvalues would be concentrated around zero, i.e., all the data would be regarded as "low-frequency"; hence, we cannot extract useful information from the low-frequency components.

The graph filter $h(\lambda) = 1 - \lambda$ can also be derived from a first-order approximation of a Laplacian regularized least squares [3]. Let us consider the problem of estimating a low-frequency true feature $\{\bar{\mathbf{x}}(i)\}_{i \in \mathcal{V}}$ from the observation $\{\mathbf{x}(i)\}_{i \in \mathcal{V}}$. Then, a natural optimization problem is given by

$$\min_{\bar{x}} \sum_{i \in \mathcal{V}} \|\bar{\mathbf{x}}(i) - \mathbf{x}(i)\|_{\tilde{D}}^2 + \Delta(\bar{X}) \tag{5}$$

where $\Delta(\bar{X}) = \sum_{p=1}^d \Delta(\bar{\mathbf{x}}(i)_p)$. Note that, since $\Delta(\bar{X}) = \sum_{\lambda \in \Lambda} \lambda \|\hat{\bar{\mathbf{x}}}(\lambda)\|_2^2$, it is a maximum a posteriori estimation with the prior distribution of $\hat{\bar{\mathbf{x}}}(\lambda) \sim N(0, I/\lambda)$. The optimal solution to this problem is given by $\bar{X} = (I + \tilde{L}_{\text{rw}})^{-1} X$. The corresponding filter is $h'(\lambda) = (1 + \lambda)^{-1}$, and hence $h(\lambda) = 1 - \lambda$ is its first-order Taylor approximation.

5   Bias-Variance Trade-off for Low Pass Filters

In the rest of this paper, we establish theoretical results under Assumption 1. To be concrete, we pose a more precise assumption as follows.

Assumption 4 (Precise Version of the First Part of Assumption 1)**.** *Observed features $\{\mathbf{x}(i)\}_{i \in \mathcal{V}}$ consists of true features $\{\bar{\mathbf{x}}(i)\}_{i \in \mathcal{V}}$ and noise $\{\mathbf{z}(i)\}_{i \in \mathcal{V}}$. The true features $\bar{X}$ have frequency at most $0 \leq \epsilon \ll 1$ and the noise follows a white Gaussian noise, i.e., each entry of the graph Fourier transform of Z independently identically follows a normal distribution $N(0, \sigma^2)$.*

Using this assumption, we can evaluate the effect of the low-pass filter as follows.

Lemma 5. *Suppose Assumption 4. For any $0 < \delta < 1/2$, with probability at least $1 - \delta$, we have*

$$\|\bar{X} - \tilde{A}_{rw}^k X\|_D \leq \sqrt{k}\epsilon \|\bar{X}\|_D + O\left(\sqrt{\log(1/\delta) R(2k)}\right) \mathbb{E}[\|Z\|_D], \tag{6}$$

where $R(2k)$ is a probability that a random walk with a random initial vertex returns to the initial vertex after $2k$ steps.

The first and second terms of the right-hand side of (6) are the *bias term* incurred by applying the filter and the *variance term* incurred from the filtered noise, respectively. Under the assumption, the bias increases a little, say, $O(\sqrt{\epsilon})$. The variance term decreases in $O(1/\deg^{k/2})$ where deg is a typical degree of the graph since $R(2k)$ typically behaves like $O(1/\deg^k)$ for small k.[3] Therefore, we can obtain a more accurate estimation of the true data from the noisy observation by multiplying the adjacency matrix if the maximum frequency of $\bar{X}$ is much smaller than the *noise-to-signal ratio* $\|Z\|_D/\|\bar{X}\|_D$.

This theorem also suggest a choice of k. By minimizing the right-hand side by k, we obtain:

---

[2]The shrinking of the maximum eigenvalue has already been proved in [26, Theorem 1]. Our theorem is stronger than theirs since we show the "monotone shrinking" of "all" the eigenvalues. In addition, our proof is simpler and shorter.

[3]This exactly holds for a class of locally tree-like graphs [6], which includes Erdos–Renyi random graphs and the configuration models.

Corollary 6. *Suppose that $\mathbb{E}[\|Z\|_D] \leq \rho\|\bar{X}\|_D$ for some $\rho = O(1)$. Let k^* be defined by $k^* = O(\log(\log(1/\delta)\rho/\epsilon))$, and suppose that there exist constants C_d and $\bar{d} > 1$ such that $R(2k) \leq C_d/\bar{d}^k$ for $k \leq k^*$. Then, by choosing $k = k^*$, the right-hand side of* (6) *is $\tilde{O}(\sqrt{\epsilon})$.*[4] □

This concludes that we can obtain an estimation of the true features with accuracy $\tilde{O}(\sqrt{\epsilon})$ by multiplying $\tilde{A}_{\mathrm{rw}}$ several times.

6 Graph Filter Neural Network

In the previous section, we see that low-pass filtered features $\tilde{A}_{\mathrm{rw}}^k X$ are accurate estimations of the true features with high probability. In this section, we analyze the performance of this operation.

Let the multi-layer (two-layer) perceptron be

$$h_{\mathrm{MLP}}(X|W_1, W_2) = \sigma_2(\sigma_1(XW_1)W_2), \tag{7}$$

where σ_1 is the entry-wise ReLU function, and σ_2 is the softmax function. Note that both σ_1 and σ_2 are contraction maps, i.e., $\|\sigma_i(X) - \sigma(Y)\|_D \leq \|X - Y\|_D$.

Under the second part of Assumption 1, our goal is to obtain a result similar to $h(\bar{X}|W_1, W_2)$. The simplest approach is to train a multi-layer perceptron $h_{\mathrm{MLP}}(X|W_1, W_2)$ with the observed feature. The performance of this approach is evaluated by

$$\|h_{\mathrm{MLP}}(\bar{X}|W_1, W_2) - h_{\mathrm{MLP}}(X|W_1, W_2)\|_{\tilde{D}} \leq \|Z\|_D \rho(W_1)\rho(W_2), \tag{8}$$

where $\rho(W)$ is the maximum singular value of W.

Now, we consider applying graph a filter $\tilde{A}_{\mathrm{rw}}^k$ to the features, then learning with a multi-layer perceptron, i.e., $h_{\mathrm{MLP}}(h(\tilde{L}_{\mathrm{rw}})X|W_1, W_2)$. Using Lemma 5, we obtain the following result.

Theorem 7. *Suppose Assumption 4 and conditions in Corollary 6. By choosing k as Corollary 6, with probability at least $1 - \delta$ for $\delta < 1/2$,*

$$\|h_{MLP}(\bar{X}|W_1, W_2) - h_{MLP}(\tilde{A}_{rw}^2 X|W_1, W_2)\|_D = \tilde{O}(\sqrt{\epsilon})\mathbb{E}[\|Z\|_D]\rho(W_1)\rho(W_2). \tag{9}$$

□

This means that if the maximum frequency of the true data is small, we obtain a solution similar to the optimal one.

Comparison with GCN   Under the same assumption, we can analyze the performance of a two-layers GCN. The GCN is given by

$$h_{\mathrm{GCN}}(X|W_1, W_2) = \sigma_2(\tilde{A}_{\mathrm{rw}}\sigma_1(\tilde{A}_{\mathrm{rw}}XW_1)W_2). \tag{10}$$

Theorem 8. *Under the same assumption to Theorem 7, we have*

$$\|h_{MLP}(\bar{X}|W_1, W_2) - h_{GCN}(X|W_1, W_2)\|_D \leq \tilde{O}(\sqrt[4]{\epsilon})\mathbb{E}[\|Z\|_D]\rho(W_1)\rho(W_2). \tag{11}$$

This theorem means that GCN also has a performance similar to MLP for true features. Hence, in the limit of $\epsilon \to 0$ and $Z \to 0$, all MLP, GCN, and the gfNN have the same performance.

In practice, gfNN has two advantages over GCN. First, gfNN is faster than GCN since it does not use the graph during the training. Second, GCN has a risk of overfitting to the noise. When the noise is so large that it cannot be reduced by the first-order low-pass filter, $\tilde{A}_{\mathrm{rw}}$, the inner weight W_1 is trained using noisy features, which would overfit to the noise. We verify this in Section 7.2.

Comparison with SGC   Recall that a degree-2 SGC is given by

$$h_{\mathrm{SGC}}(X|W_1) = \sigma_2(\tilde{A}_{\mathrm{rw}}^2 XW_2), \tag{12}$$

i.e., it is a gfNN with a one-layer perceptron. Thus, by the same discussion, SGC has a performance similar to the perceptron (instead of the MLP) applied to the true feature. This clarifies an issue of SGC: if the true features are non-separable, SGC cannot solve the problem. We verify this issue in experiment **E2** (Section 7.3).

---

[4]$\tilde{O}(\cdot)$ suppresses logarithmic dependencies.

7 Experiments

To verify our claims in previous sections, we design two experiments. In experiment **E1**, we add different levels of white noise to the feature vectors of real-world datasets and report the classification accuracy among baseline models. Experiment **E2** studies an artificial dataset which has a complex feature space to demonstrate when simple models like SGC fail to classify.

There are two types of datasets: Real-worlds data (citation networks, social networks, and biological networks) commonly used for graph neural network benchmarking [20, 28, 31] and artificially synthesized random graphs from the two circles dataset. We created the graph structure for the two circles dataset by making each data point into a vertex and connect the 5 closest vertices in Euclidean distance. Table 1 gives an overview of each dataset.

Table 1: Real-world benchmark datasets and synthetic datasets for vertex classification

Dataset	Nodes	Edges	Features	Classes	Train/Val/Test Nodes
Cora	2,708	5,278	1,433	7	140/500/1,000
Citeseer	3,327	4,732	3,703	6	120/500/1,000
Pubmed	19,717	44,338	500	3	60/500/1,000
Reddit	231,443	11,606,919	602	41	151,708/23,699/55,334
PPI	56,944	818,716	50	121	44,906/6,514/5,524
Two Circles	4,000	10,000	2	2	80/80/3,840

7.1 Neural Networks

We compare our results with a few state-of-the-art graph neural networks on benchmark datasets. We train each model using Adam optimizer [14] (`lr=0.2, epochs=50`). We use 32 hidden units for the hidden layer of GCN, MLP, and gfNN. Other hyperparameters are set similar to SGC [26].

gfNN We have three graph filters for our simple model: *Left Norm* (▼), *Augumented Normalized Adjacency* (▲), and *Bilateral* (◆).

SGC [26] Simple Graph Convolution (■) simplifies the Graph Convolutional Neural Network model by removing nonlinearity in the neural network and only averaging features.

GCN [15] Graph Convolutional Neural Network (✖) is the most commonly used baseline.
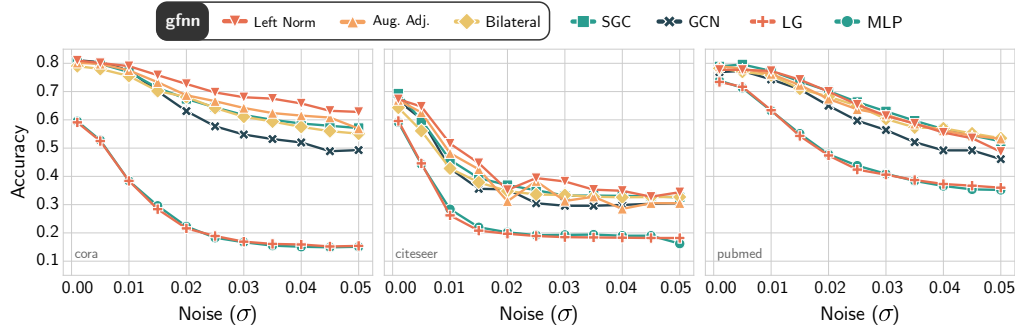

Figure 4: Benchmark test accuracy on Cora (left), Citeseer (middle), and Pubmed (right) datasets. The noise level is measured by standard deviation of white noise added to the features.

7.2 Denoising Effect of Graph Filters

For each dataset in Table 1, we introduce a white noise $\mathcal{N}(0, \delta^2)$ into the feature vectors with δ in range $(0.01, 0.05)$. According to the implications of Theorem 8 and Theorem 7, GCN should exhibit a low tolerance to feature noise due to its first-order denoising nature. As the noise level increases, we see in Figure 4 that GCN, Logistic Regression (LR), and MLP tend to overfit more to noise. On the other hand, gfNN and SGC remain comparably noise tolerant.

7.3 Expressiveness of Graph Filters

Since graph convolution is simply denoising, we expect it to have little contribution to non-linearity learning. Therefore, we implement a simple 2-D two circles dataset to see whether graph filtering can have the non-linear manifold learning effect. Based on the Euclidean distance between data points, we construct a k-nn graph to use as the underlying graph structure. In this particular experiment, we have each data point connected to the 5 closest neighbors. Figure 5 demonstrates how SGC is unable to classify a simple 500-data-points set visually.
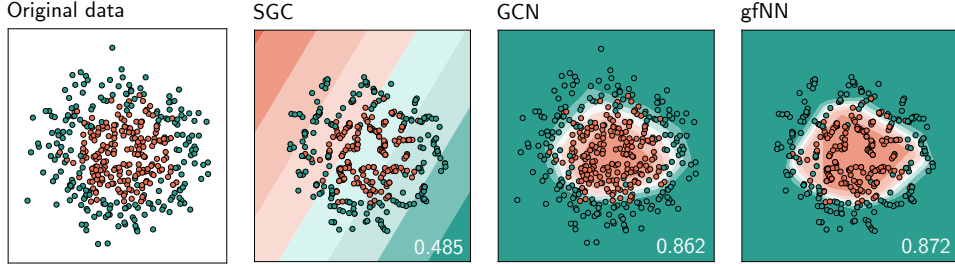

Figure 5: Decision boundaries on 500 generated data samples following the two circles pattern

On the other hand, SGC, GCN, and our model gfNN perform comparably on most benchmark graph datasets. Table 2 compares the accuracy (f1-micro for Reddit) of randomized train/test sets. We see that all these graph neural networks have similar performance (some minor improvement might subject to hyperparameters tuning). Therefore, we believe current benchmark datasets for graph neural network have quite "simple" feature spaces.

In practice, both of our analysis and experimental results suggest that the graph convolutional layer should be thought of simply as a denoising filter. In contrast to the recent design trend involving GCN [1, 22] , our results imply that simply stacking GCN layers might only make the model more prone to feature noise while having the same expressiveness as a simple MLP.

Table 2: Average test accuracy on random train/val/test splits (5 times)

	Denoise	Classifier	Cora	Citeseer	Pubmed	Reddit	PPI	Two Circles
GCN	1st order	*Non-linear*	80.0 ± 1.8	69.6 ± 1.1	79.3 ± 1.3	OOM	OOM	83.9 ± 3.1
SGC	*2nd order*	Linear	77.6 ± 2.2	65.6 ± 0.1	78.4 ± 1.1	94.9 ± 1.2	62.0 ± 1.2	54.5 ± 4.4
gfNN	*2nd order*	*Non-linear*	80.9 ± 1.3	69.3 ± 1.1	81.2 ± 1.5	92.4 ± 2.2	62.2 ± 1.5	84.6 ± 2.5

8 Discussion

There has been little work addressed the limits of the GCN architecture. Kawamoto et al. [13] took the mean-field approach to analyze a simple GCN model to statistical physics. They concluded that backpropagation improves neither accuracy nor detectability of a GCN-based GNN model. Li et al. [16] empirically analyzed GCN models with many layers under the limited labeled data setting and stated that GCN would not do well with little labeled data or too many stacked layers. While these results have provided an insightful view for GCN, they have not insufficiently answered the question: *When we should we use GNN?*.

Our results indicate that we *should* use the GNN approach to solve a given problem if Assumption 1 holds. From our perspective, GNNs derived from GCN simply perform noise filtering and learn from denoised data. Based on our analysis, we propose two cases where GCN and SGC might fail to perform: noisy features and nonlinear feature spaces. In turn, we propose a simple approach that works well in both cases.

Recently GCN-based GNNs have been applied in many important applications such as point-cloud analysis [22] or weakly-supervised learning [8]. As the input feature space becomes complex, we advocate revisiting the current GCN-based GNNs design. Instead of viewing a GCN layer as the convolutional layer in computer vision, we need to view it simply as a denoising mechanism. Hence, simply stacking GCN layers only introduces overfitting and complexity to the neural network design.

References

[1] Sami Abu-El-Haija, Bryan Perozzi, Amol Kapoor, Hrayr Harutyunyan, Nazanin Alipourfard, Kristina Lerman, Greg Ver Steeg, and Aram Galstyan. Mixhop: Higher-order graph convolution architectures via sparsified neighborhood mixing. In *Proceedings of the 36th International Conference on International Conference on Machine Learning*, volume 97. JMLR, 2019.

[2] Joost Bastings, Ivan Titov, Wilker Aziz, Diego Marcheggiani, and Khalil Sima'an. Graph convolutional encoders for syntax-aware neural machine translation. *Empirical Methods in Natural Language Processing*, 2017.

[3] Mikhail Belkin and Partha Niyogi. Semi-supervised learning on riemannian manifolds. *Machine Learning*, 56(1-3):209–239, 2004.

[4] Fan R.K. Chung. *Spectral graph theory*. American Mathematical Society, 1997.

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

[6] Amir Dembo, Andrea Montanari, and Nike Sun. Factor models on locally tree-like graphs. *The Annals of Probability*, 41(6):4162–4213, 2013.

[7] Alex Fout, Jonathon Byrd, Basir Shariat, and Asa Ben-Hur. Protein interface prediction using graph convolutional networks. In *Advances in Neural Information Processing Systems*, pages 6530–6539, 2017.

[8] Victor Garcia and Joan Bruna. Few-shot learning with graph neural networks. *International Conference on Learning Representations*, 2017.

[9] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70, pages 1263–1272. JMLR, 2017.

[10] Benjamin Girault, Antonio Ortega, and Shrikanth S. Narayanan. Irregularity-aware graph fourier transforms. *IEEE Transactions on Signal Processing*, 66(21):5746–5761, 2018.

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

[12] Nicholas J. Higham. *Functions of matrices: theory and computation*, volume 104. SIAM, 2008.

[13] Tatsuro Kawamoto, Masashi Tsubaki, and Tomoyuki Obuchi. Mean-field theory of graph neural networks in graph partitioning. In *Advances in Neural Information Processing Systems*, pages 4361–4371, 2018.

[14] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 2015.

[15] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017.

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

[17] Antonio Ortega, Pascal Frossard, Jelena Kovačević, José MF Moura, and Pierre Vandergheynst. Graph signal processing: Overview, challenges, and applications. *Proceedings of the IEEE*, 106 (5):808–828, 2018.

[18] Lawrence R. Rabiner and Bernard Gold. Theory and application of digital signal processing. *Englewood Cliffs, NJ, Prentice-Hall, Inc.*, 1975.

[19] Adam Santoro, David Raposo, David G. Barrett, Mateusz Malinowski, Razvan Pascanu, Peter Battaglia, and Timothy Lillicrap. A simple neural network module for relational reasoning. In *Advances in Neural Information Processing Systems*, pages 4967–4976, 2017.

[20] 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.

[21] David I. Shuman, Sunil K. Narang, Pascal Frossard, Antonio Ortega, and Pierre Vandergheynst. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *arXiv preprint arXiv:1211.0053*, 2012.

[22] Diego Valsesia, Giulia Fracastoro, and Enrico Magli. Learning localized generative models for 3d point clouds via graph convolution. *International Conference on Learning Representations*, 2019.

[23] Saeed V. Vaseghi. *Advanced digital signal processing and noise reduction*. John Wiley & Sons, 2008.

[24] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *International Conference on Learning Representations*, 2017.

[25] Petar Veličković, William Fedus, William L. Hamilton, Pietro Liò, Yoshua Bengio, and R. Devon Hjelm. Deep graph infomax. *International Conference on Learning Representations*, 2019.

[26] Felix Wu, Tianyi Zhang, Amauri Holanda de Souza Jr., Christopher Fifty, Tao Yu, and Kilian Q. Weinberger. Simplifying graph convolutional networks. In *Proceedings of the 36th International Conference on International Conference on Machine Learning*, volume 97. JMLR, 2019.

[27] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *International Conference on Learning Representations*, 2019.

[28] Zhilin Yang, William W Cohen, and Ruslan Salakhutdinov. Revisiting semi-supervised learning with graph embeddings. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning*, volume 48. JMLR, 2016.

[29] Jiani Zhang, Xingjian Shi, Junyuan Xie, Hao Ma, Irwin King, and Dit-Yan Yeung. Gaan: Gated attention networks for learning on large and spatiotemporal graphs. In *Proceedings of the Thirty Fourth Conference on Uncertainty in Artificial Intelligence*, 2018.

[30] Yuhao Zhang, Peng Qi, and Christopher D. Manning. Graph convolution over pruned dependency trees improves relation extraction. *Empirical Methods in Natural Language Processing*, 2018.

[31] Marinka Zitnik and Jure Leskovec. Predicting multicellular function through multi-layer tissue networks. *Bioinformatics*, 33(14):i190–i198, 2017.