Graph Convolutional Network for Recommendation with Low-pass Collaborative Filters

Wenhui Yu [1]   **Zheng Qin** [1 2]

Abstract

Graph **C**onvolutional **N**etwork (**GCN**) is widely used in graph data learning tasks such as recommendation. However, when facing a large graph, the graph convolution is very computationally expensive thus is simplified in all existing GCNs, yet is seriously impaired due to the oversimplification. To address this gap, we leverage the *original graph convolution* in GCN and propose a **L**ow-pass **C**ollaborative **F**ilter (**LCF**) to make it applicable to the large graph. LCF is designed to remove the noise caused by exposure and quantization in the observed data, and it also reduces the complexity of graph convolution in an unscathed way. Experiments show that LCF improves the effectiveness and efficiency of graph convolution and our GCN outperforms existing GCNs significantly. Codes are available on https://github.com/Wenhui-Yu/LCFN.

1. Introduction

Graph convolution (Shuman et al., 2013) is the extension of convolution from the Euclidean domain to the graph domain and **G**raph **C**onvolutional **N**etwork (**GCN**) is the extension of **C**onvolutional **N**eural **N**etwork (**CNN**). GCN is widely explored in graph learning tasks such as recommendation since it can extract abundant information from the graph data. However, the eigen-decomposition of a large matrix is required in the graph convolution (Shuman et al., 2013), which is extremely space- and time-consuming in real-world applications. To avoid the eigen-decomposition, all existing GCNs constrain (even fix) the convolutional kernel and simplify graph convolution to graph propagation (Defferrard et al., 2016; Kipf & Welling, 2017). **G**raph **N**eural **N**etwork (**GNN**) propagates signals through the graph, and refines

the signal of each node with its neighborhoods (Zhou et al., 2018). By constraining the kernel, GCNs lose the ability of convolution and degenerate to GNNs (discussed in detail in Section 5.4). To address the oversimplification issue, we propose a new GCN that can really leverage the ability of graph convolution and validate the effectiveness in the recommendation task. In our GCN, we propose a **L**ow-pass **C**ollaborative **F**ilter (**LCF**) both to filter the noise in observed data and to reduce the time consumption of graph convolution without hurting its ability.

LCF is devised to remove the noise in the observed data. In this paper, we adopt implicit feedback data (interactions such as "purchase", "click", "watch" records, denoted by binary variables) since it is generally applicable in recommendation tasks. A matrix for observed interactions is illustrated in Figure 1(c), and Figure 1 shows how it is generated: There is an unknown underlying numeric rating matrix $\mathbf{R}_0$ indicating the true preference score for each user towards each item, shown in Figure 1(a). When browsing items on a website, users can only browse a small proportion of the mass items, thus many items are not exposed to the users, including many preferred items. Reflected in the matrix representation, many elements in $\mathbf{R}_0$ are set to 0 randomly, shown in Figure 1(b). To simulate this process, we add exposure noise $\mathbf{N}_1$ to the rating matrix $\mathbf{R}_0$. For the remaining exposed items, a user will interact with them only if she like them. In the matrix representation, we discretize the matrix $\mathbf{R}_0 + \mathbf{N}_1$ with a threshold, which is set as 0.5 in this example. The quantization noise is denoted as $\mathbf{N}_2$, and we have $\mathbf{R} = \mathbf{R}_0 + \mathbf{N}_1 + \mathbf{N}_2$. Our task is to reconstruct $\mathbf{R}_0$ from observed $\mathbf{R}$ and rank all items for each user based on the reconstruction.

Considering $\mathbf{R}_0$ is low frequency due to the strong dependence and $\mathbf{N}_1$ and $\mathbf{N}_2$ are high frequency due to the randomness (we will introduce the reason in detail in Section 3.1), we can remove noise by low-pass filtering. However, different from the images, $\mathbf{R}$, $\mathbf{R}_0$, $\mathbf{N}_1$ and $\mathbf{N}_2$ are all 2D graph signals, thus conventional filters are not effective. For signals defined in the Euclidean domain like images and timing signals, the dependence is based on position, such as pixels near in the space position or sampling points near in the time axis. Nevertheless, for signals defined in the graph

---

[1]Tsinghua University [2]Corresponding author. Correspondence to: Wenhui Yu <yuwh16@mails.tsinghua.edu.cn>, Zheng Qin <qingzh@mail.tsinghua.edu.cn>.
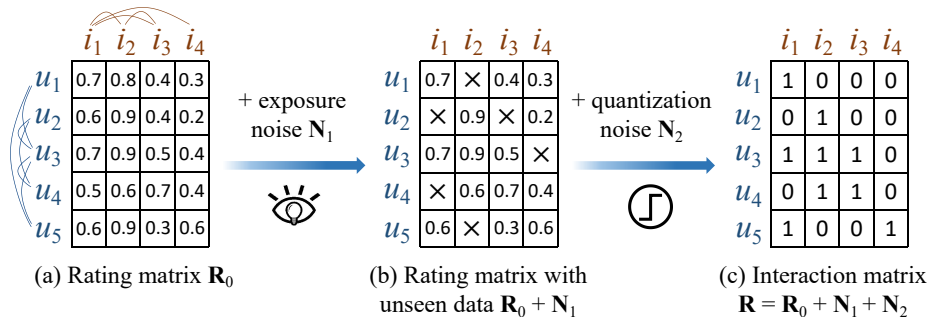
Figure 1. The interaction matrix and different kinds of noise. Curves in Figure 1(a) indicate a user graph and an item graph. Two users are connected if they have interacted with the same item, and two items are connected if they have been interacted by the same user.

domain, the dependence is based on graph connections, for example, we consider the dependence of two rows/columns of $\mathbf{R}_0$ based on the user/item graph shown in Figure 1(a). To remove the graph noise, we devise a 2D graph filter called LCF and reconstruct $\mathbf{R}_0$ by it: $\hat{\mathbf{R}}_0 = LCF(\mathbf{R})$, where $\hat{\mathbf{R}}_0$ is the estimation. Similar to the filter, we also extend 2D convolution from the Euclidean domain to the graph domain to propose GCN. We then inject LCF into GCN to propose our **LCF N**etwork (**LCFN**). LCF not only improves the accuracy by removing the noise, but also reduces the computation of graph convolution since only a small proportion of eigen-vectors are required. To the best of our knowledge, LCFN is the first GCN to use the original graph convolution.

To summarize, this work makes the following contributions:

- We introduce the 2D graph Fourier transform, and then design the graph filter LCF and graph convolution based on it.

- We design a novel GCN named LCFN for recommendation with LCF to improve the performance by removing noise and to improve the efficiency of graph convolution.

- We devise comprehensive experiments on two real-world datasets to demonstrate the effectiveness and efficiency of LCFN.

2. Related Work

After the collaborative filtering (CF) model was proposed (Sarwar et al., 2001; Goldberg, 1992), recommender systems have boomed in decades. Among various CF methods, matrix factorization (MF) (Koren, 2009; Rendle et al., 2009), which encodes the preference of users by underlying embeddings (i.e., latent factors), is a basic yet the most effective recommender model. To improve the presentation capability, many variants have been proposed (He et al., 2017; Wang et al., 2019). Among these models, neural network-based models and graph-based models are the most relevant to our work.

Neural Network-based Recommendation. These years, Neural Network achieves significant success and is widely used in various machine learning and information retrieval domains, including recommendation. Cheng et al. (2016); He et al. (2017) utilized wide and deep networks to learn the nonlinear combination of embeddings. Chen et al. (2017) explored the attention mechanism to highlight important elements. He & McAuley (2016b); Yu et al. (2018) explored side information to enhance the performance of recommendation. Li et al. (2017); Tan et al. (2016) regarded recommendation as a sequence prediction task and utilized RNN or CNN to deal with this issue. Though extensively explored, these models are based on conventional networks and omit the graph structure.

Graph-based Recommendation. As a link prediction task of a bipartite graph, graph tools are widely used in the recommendation domain. Rao et al. (2015) used graph regularization to smooth the embeddings of connected nodes (users and items in recommendation context). Yu & Qin (2019) extracted spectral features for nodes from the hypergraph Laplacian matrices to optimize model learning. However, these shallow models only get very limited information. Wang et al. (2019); Ying et al. (2018); Berg et al. (2018) utilized deep GNNs for recommendation tasks. GNNs propagate embeddings through the graph to ensure that connected nodes have similar latent representations. Though GNNs perform well, there is still a large room for improvement — GNNs only smooth the embeddings while cannot extract high-level patterns for the graph signal. To address this gap, GCNs are proposed.

Shuman et al. (2013) proposed graph convolution to extract the dependence pattern from the graph data. Since graph convolution on large graphs is extremely time-consuming, Defferrard et al. (2016); Kipf & Welling (2017) first simplified it by various approximations, unfortunately, degenerate GCNs to GNNs. By following them, existing GCNs are indeed GNNs, or variants of GNNs (Yao & Li, 2018; Monti et al., 2017; Zheng et al., 2018). Also, as a result, many

GNNs are claimed as GCNs (Berg et al., 2018; Ying et al., 2018) (We follow the definition of GNN and GCN in (Zhou et al., 2018)). In this paper, we devise LCF to reduce the computation of graph convolution and to remove the noise. Different from existing GCNs, our simplification strategy enhances the graph convolution rather than degenerating it.

3. Preliminaries

In this section, we introduce some background knowledge of the signal processing in the Euclidean domain.

3.1. Fourier Transform

Fourier transform is a widely-used tool for frequency domain analysis. It converts the signal from the time domain to the frequency domain. For a continuous signal $f(t)$, the Fourier transform is: $F(\omega) = \int_{\mathbb{R}} f(t) e^{-j\omega t} dt$ and the inverse transform is $f(t) = \frac{1}{2\pi} \int_{\mathbb{R}} F(\omega) e^{j\omega t} d\omega$. For a digital signal (discrete signal) denoted as $\{s_n\}$, the Fourier transform is: $S_k = \sum_{n=0}^{N-1} (s_n) e^{-\frac{2\pi j}{N} kn}$ and the inverse transform is $s_n = \frac{1}{N} \sum_{k=0}^{N-1} (S_k) e^{\frac{2\pi j}{N} kn}$, where j is the imaginary unit and N is the length of the series $\{s_n\}$.

(a) $s_n^1 = \sin(\frac{\pi}{10} n)$ (b) $s_n^2 = \sin(\frac{7\pi}{10} n)$ (c) $s_n^3 = s_n^1 + s_n^2$

(d) $S_k^1 = |\mathcal{F}(s_n^1)|$ (e) $S_k^2 = |\mathcal{F}(s_n^2)|$ (f) $S_k^3 = |\mathcal{F}(s_n^3)|$
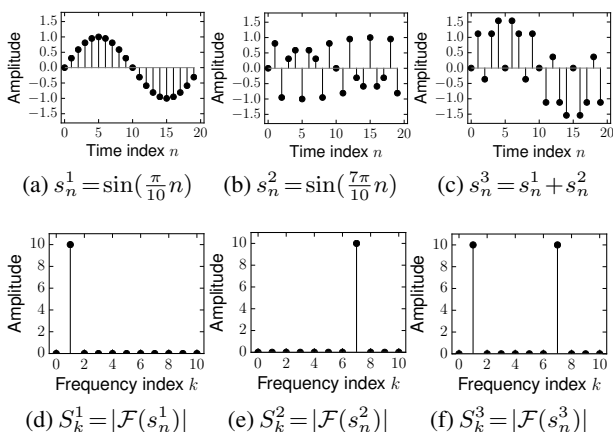
Figure 2. Signals with different frequencies in the time domain and frequency domain.

In this paper, we denote the Fourier transform as $\mathcal{F}(\ )$ and the inverse transform as $\mathcal{F}^{-1}(\ )$. Figure 2 shows some signals with different frequencies. Figures 2(a)(b)(c) show signals in the time domain and Figures 2(d)(e)(f) show signals in the frequency domain. s_n^1 in Figure 2(a) is a low-frequency signal. In the time domain, s_n^1 is very smooth and shows strong dependence between two adjacent sampling points. In the frequency domain, the signal is distributed on the low frequency (shown in Figure 2(d)). In contrast, s_n^2 in Figure 2(b) is a high-frequency signal thus shows weak dependence and is distributed on the high frequency (shown in Figure 2(e)). s_n^3 in Figure 2(c) is the addition of s_n^1 and

s_n^2. They are fully mixed in the time domain, nevertheless can be easily separated in the frequency domain (shown in Figure 2(f)). In telecommunication, low-frequency signals are usually contaminated by high-frequency noise when transmitting through the channel, and we can reconstruct the signal by a low-pass filter.

As discussed, the frequency in frequency domain reflects the dependence in time domain, however, in graph data, the dependence is based on node connections. Recalling the example shown in Figure 1, u_1 and u_3 are connected users, the two rows for them in $\mathbf{R}_0$ are $(0.7, 0.8, 0.4, 0.3)$ and $(0.7, 0.9, 0.5, 0.4)$, and in $\mathbf{N}_1$ are $(0, -0.8, 0, 0)$ and $(0, 0, 0, -0.4)$. We can see that the true preference of two connected nodes shows strong dependence thus $\mathbf{R}_0$ is low-frequency, nevertheless the noise shows weak dependence due to the random position, thus is high-frequency. We can remove the high-frequency graph noise from the low-frequency graph signal by a low-pass graph filter.

3.2. Convolution

In addition to Fourier transform, convolution is a very important signal processing tool which is widely used in CNNs. For two continuous signals $f(t)$ and $g(t)$, the convolution of them is defined as: $f(t) * g(t) = \int_{\mathbb{R}} f(\tau) g(t - \tau) d\tau$. For two digital signals $\{s_n\}_{n=0\cdots N-1}$ and $\{k_m\}_{m=0\cdots M-1}$, the convolution is: $s_n' = \sum_{m=0}^{M-1} k_m (s_{n-m})$, where N and M indicate the length of the signals. In machine learning tasks, M is usually much smaller than N, and $\{k_m\}$ is called the convolutional kernel, $\{s_n'\}$ is the feature map. We can extract local dependence information from the signal by convolution. Intuitively speaking, if $\{s_n\}$ is very similar to $\{k_m\}$ in a local area, $\{s_n'\}$ will be very large in this position.

We also introduce a widely-used property of convolution, which is crucial in graph convolution design: *Convolution Theorem* (Barrett & Wilde, 1960) tells us that convolution in the time domain is equivalent to the product in the frequency domain, i.e., $f(t) * g(t) = \mathcal{F}^{-1}(\mathcal{F}(f(t))\mathcal{F}(g(t)))$. For graph data, convolution is hard to define directly due to the irregular structure of graphs, thus is defined in this way.

4. 2D Low-pass Graph Convolution

In this paper, bold uppercase letters refer to matrices. Assuming there are M users and N items in total, we use matrix $\mathbf{R} \in \mathbb{R}^{M \times N}$ to denote the interactions between users and items. $\mathbf{R}_{ui} = 1$ if user u has interacted with item i and $\mathbf{R}_{ui} = 0$ otherwise. Our task is to predict the missing entities, i.e., 0 in $\mathbf{R}$.

4.1. 2D Graph Fourier Transform

To propose LCF and the graph convolution, we need to propose graph Fourier transform first. Shuman et al. (2013)

extended Fourier transform from the Euclidean domain to the graph domain. In this section, we follow Shuman et al. (2013) and propose a 2D graph Fourier transform. For a continuous 2D signal in Euclidean domain $f(x, y)$, the Fourier transform is:

$$F(\phi,\psi){=}\iint\limits_{\mathbb{R}\,\mathbb{R}} f(x,y)\mathrm{e}^{-j(\phi x+\psi y)}dxdy{=}\left\langle f(x,y), \mathrm{e}^{-j\phi x}{\cdot}\mathrm{e}^{-j\psi y}\right\rangle, \quad (1)$$

which is defined as the inner product of the signal and the product of transform bases on two dimensions. Taking dimension x as an example, the transform base $\mathrm{e}^{-j\phi x}$ and corresponding frequency ϕ are defined as the eigen-function $\mathrm{e}^{-j\phi x}$ and eigen-value $-j\phi$ of the differential operator: $\nabla_x \mathrm{e}^{-j\phi x} = -j\phi \mathrm{e}^{-j\phi x}$.

To extend the above definition to the graph domain, we utilize the differential operator for graph signals: Laplacian matrices. We construct Laplacian matrices $\mathbf{L}^U$ and $\mathbf{L}^I$ for user and item graphs. The transform bases and the frequencies for graph Fourier transform are defined as the eigen-vectors and eigen-values of the Laplacian matrices.

Now, we introduce how to construct the Laplacian matrices. Considering hypergraphs are more powerful in representation and provide more information than simple graphs, we follow Yu & Qin (2019) to use two hypergraphs instead of simple graphs shown in Figure 1(a). The hypergraph is a generalization of the simple graph, where an edge connects any number of nodes rather than just two, called hyperedge. Assuming there are M nodes and N hyperedges, a hypergraph is usually represented with the incidence matrix $\mathbf{H} \in \mathbb{R}^{M \times N}$. Each row of $\mathbf{H}$ is for a node and each column is for a hyperedge. $\mathbf{H}_{ij} = 1$ if node i is connected by hyperedge j and $\mathbf{H}_{ij} = 0$ otherwise. The Laplacian matrix $\mathbf{L}$ is defined as (Zhou et al., 2006):

$$\mathbf{L} = \mathbf{I} - \mathbf{D}^{-\frac{1}{2}}\mathbf{H}\mathbf{W}\boldsymbol{\Delta}^{-1}\mathbf{H}^{\mathsf{T}}\mathbf{D}^{-\frac{1}{2}}, \quad (2)$$

where the diagonal matrix $\mathbf{D} \in \mathbb{R}^{M \times M}$ denotes degrees of nodes, the diagonal matrix $\mathbf{W} \in \mathbb{R}^{N \times N}$ denotes weights of hyperedges, and the diagonal matrix $\boldsymbol{\Delta} \in \mathbb{R}^{N \times N}$ denotes the degrees of hyperedges. The effect of the Laplacian matrix is to take the difference for a signal on the hypergraph (Zhou et al., 2006; Yu & Qin, 2019).

As hypergraphs can represent the interaction data naturally, we proposed two hypergraphs for users and items. In the user hypergraph, users are nodes and items are hyperedges and in the item hypergraph, items are nodes and users are hyperedges. Laplacian matrices for users $\mathbf{L}^U$ and for items $\mathbf{L}^I$ can be constructed by setting $\mathbf{H} = \mathbf{R}$ and $\mathbf{H} = \mathbf{R}^{\mathsf{T}}$ in Equation (2), respectively. After getting the differential operators $\mathbf{L}^U$ and $\mathbf{L}^I$, we decompose them by eigen-decomposition. Considering $\mathbf{L}^U$ and $\mathbf{L}^I$ are positive semi-definite, we have $\mathbf{L}^U = \mathbf{P}\boldsymbol{\Lambda}\mathbf{P}^{\mathsf{T}}$ and $\mathbf{L}^I = \mathbf{Q}\boldsymbol{\Sigma}\mathbf{Q}^{\mathsf{T}}$. Eigen-vector matrices $\mathbf{P}$ and $\mathbf{Q}$ are graph Fourier transform bases for user and item dimensions, respectively and eigen-value matrices $\boldsymbol{\Lambda} = diag(\lambda_1, \lambda_2, \cdots, \lambda_M)$ and $\boldsymbol{\Sigma}{=}diag(\sigma_1, \sigma_2, \cdots, \sigma_N)$ are corresponding frequencies (all eigen-values are in ascending order). Similar to Equation (1), the graph Fourier

transform of signal $\mathbf{R}$ is the inner product of $\mathbf{R}$ and the product of transform bases on two dimensions:

$$\tilde{\mathbf{R}}_{\phi\psi} = \left\langle \mathbf{R}, \mathbf{P}_{*\phi} \cdot \mathbf{Q}_{*\psi}^{\mathsf{T}}\right\rangle = \sum_{u=1}^{M}\sum_{i=1}^{N}\mathbf{R}_{ui}\mathbf{P}_{u\phi}\mathbf{Q}_{i\psi}. \quad (3)$$

Then we have $\tilde{\mathbf{R}} = \mathcal{F}_g(\mathbf{R}) = \mathbf{P}^{\mathsf{T}}\mathbf{R}\mathbf{Q}$, where $\tilde{\mathbf{R}}$ is the graph Fourier transform of $\mathbf{R}$. The inverse transform is $\mathcal{F}_g^{-1}(\tilde{\mathbf{R}}) = \mathbf{P}\tilde{\mathbf{R}}\mathbf{Q}^{\mathsf{T}}$. In graph Fourier transform, the "time domain" corresponds to the "graph domain" and the "frequency domain" (also called "spectral domain") is defined by eigen-values: $\begin{bmatrix} (\lambda_1,\sigma_1) & \cdots & (\lambda_1,\sigma_N) \\ \vdots & \ddots & \vdots \\ (\lambda_M,\sigma_1) & \cdots & (\lambda_M,\sigma_N) \end{bmatrix}$. Note that $\tilde{\mathbf{R}}_{\phi\psi}$ in Equation (3) is the component on frequencies $(\lambda_\phi, \sigma_\psi)$, thus we can also denote $\tilde{\mathbf{R}}_{\phi\psi}$ as $\tilde{\mathbf{R}}(\lambda_\phi, \sigma_\psi)$.

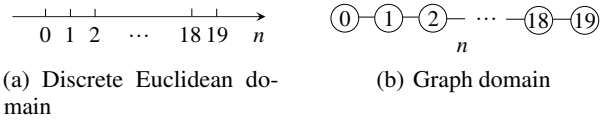(a) Discrete Euclidean domain

(b) Graph domain

Figure 3. Discrete Euclidean domain can be regarded as a special case of graph domain.

Now, we give a brief example to illustrate the effectiveness of the graph convolution. We recall the example introduced in Figure 2. These signals are distributed on discrete Euclidean domain, which can be regarded as a special case of graph domain (please see Figure 3).

(a) $S_k^1 = |\mathcal{F}_g(s_n^1)|$ (b) $S_k^2 = |\mathcal{F}_g(s_n^2)|$ (c) $S_k^3 = |\mathcal{F}_g(s_n^3)|$
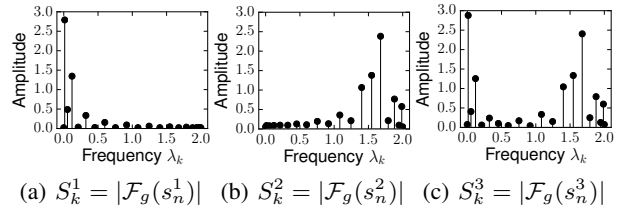
Figure 4. Examples of graph Fourier transform.

Figure 4 show the graph Fourier transform of $\{s_n^1\}$, $\{s_n^2\}$, and $\{s_n^3\}$ introduced in Figure 2. We first convert $\{s_n^1\}$, $\{s_n^2\}$, and $\{s_n^3\}$ to graph signals by distributing them on the graph shown in Figure 3(b), and then transform them. To give a concise illustration, we leverage 1D signals, 1D transform, and simple graph in this example (Shuman et al., 2013). The abscissa in Figure 4 is the frequency given by eigen-values of the Laplacian matrix. We can see that graph Fourier transform can be used for frequency analysis for graph signals.

4.2. Graph Convolution with Low-pass Filters

LCF. For the signal $\mathbf{R}$ in the frequency domain $\mathcal{F}_g(\mathbf{R})$, we want LCF to attenuate the high-frequency component of $\mathbf{R}$

while retain the low-frequency component, thus we leverage the gate function $\tilde{\mathbf{F}} = \begin{bmatrix} \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}$, where $\tilde{\mathbf{F}} \in \{0,1\}^{M \times N}$, $\mathbf{1} \in \{1\}^{\Phi \times \Psi}$ is an all-one matrix and $\mathbf{0}$ is an all-zero matrix. The passband cut-off frequencies on the user dimension and item dimension are λ_Φ and σ_Ψ, respectively. The passed signal is $LCF(\mathbf{R}) = \mathcal{F}_g^{-1}(\mathcal{F}_g(\mathbf{R}) \odot \tilde{\mathbf{F}})$, where $\odot$ is the element-wise product. By tuning Φ and Ψ, we can remove $\mathbf{N}_1$ and $\mathbf{N}_2$ and retain $\mathbf{R}_0$ as much as possible. An intuitive example is represented in Appendix A.

Graph convolution. We extend convolution theorem to the graph domain: Given a convolutional kernel $\mathbf{K} \in \mathbb{R}^{M \times N}$, the convolution of the signal (feature map) is defined as $\mathbf{R}' = \mathbf{R} *_g \mathbf{K} = \mathcal{F}_g^{-1}(\mathcal{F}_g(\mathbf{R}) \odot \tilde{\mathbf{K}})$, where $*_g$ is the graph convolution and $\tilde{\mathbf{K}} = \mathcal{F}_g(\mathbf{K})$ is the kernel in the frequency domain.

Low-pass graph convolution. We first filter the signal $\mathbf{R}$ by LCF $\tilde{\mathbf{F}}$ and then perform the convolution by a kernel $\tilde{\mathbf{K}}$: $\mathbf{R}' = \mathcal{F}_g^{-1}(\mathcal{F}_g(\mathbf{R}) \odot \tilde{\mathbf{F}} \odot \tilde{\mathbf{K}}) = \mathbf{P}((\mathbf{P}^\mathsf{T}\mathbf{R}\mathbf{Q}) \odot \tilde{\mathbf{F}} \odot \tilde{\mathbf{K}})\mathbf{Q}^\mathsf{T}$, we have:

$$\mathbf{R}'_{u'i'} = \sum_{\phi=1}^{M}\sum_{\psi=1}^{N}\left(\sum_{u=1}^{M}\sum_{i=1}^{N}\mathbf{R}_{ui}\mathbf{P}_{u\phi}\mathbf{Q}_{i\psi}\right)\tilde{\mathbf{F}}_{\phi\psi}\tilde{\mathbf{K}}_{\phi\psi}\mathbf{P}_{u'\phi}\mathbf{Q}_{i'\psi}$$
$$= \sum_{\phi=1}^{\Phi}\sum_{\psi=1}^{\Psi}\left(\sum_{u=1}^{M}\sum_{i=1}^{N}\mathbf{R}_{ui}\mathbf{P}_{u\phi}\mathbf{Q}_{i\psi}\right)\tilde{\mathbf{K}}_{\phi\psi}\mathbf{P}_{u'\phi}\mathbf{Q}_{i'\psi}.$$

We can see that only the first Φ/Ψ eigen-vectors of $\mathbf{L}^U/\mathbf{L}^V$ are needed. We define $\bar{\mathbf{P}} = \mathbf{P}_{*,1:\Phi}$, $\bar{\mathbf{Q}} = \mathbf{Q}_{*,1:\Psi}$, and $\bar{\mathbf{K}} = \tilde{\mathbf{K}}_{1:\Phi,1:\Psi}$, thus $\mathbf{R}' = \bar{\mathbf{P}}((\bar{\mathbf{P}}^\mathsf{T}\mathbf{R}\bar{\mathbf{Q}}) \odot \bar{\mathbf{K}})\bar{\mathbf{Q}}^\mathsf{T}$.

5. Low-pass Collaborative Filter Network

In this section, we propose LCFN and analyze it by comparing with some widely-used GNNs and GCNs.

5.1. Model Structure

LCFN is a GCN with our proposed low-pass graph convolution. The structure is illustrated in Appendix B. We stack multiple convolutional layers to extract high-level features. Details of the convolutional layer are introduced below.

Low-pass convolution. In LCFN, we input the feature map from the previous layer $\mathbf{R}'$ to the current layer. Considering these feature maps are large and dense, we represent them in a form of MF: $\mathbf{R}' \approx \mathbf{U}\mathbf{V}^\mathsf{T}$, where $\mathbf{U} \in \mathbb{R}^{M \times K}$ and $\mathbf{V} \in \mathbb{R}^{N \times K}$ are embeddings. To avoid overfitting, we reduce the number of parameters by redefining the convolutional kernel as a rank-1 matrix $\bar{\mathbf{K}} = \mathbf{k}^U\mathbf{k}^{V\mathsf{T}}$, where column vectors $\mathbf{k}^U \in \mathbb{R}^\Phi$ and $\mathbf{k}^V \in \mathbb{R}^\Psi$ are kernels for user and item dimensions, respectively. In this case, performing convolution on the feature map $\mathbf{R}'$ equals to performing convolution on the embeddings: $\mathbf{U}' = \bar{\mathbf{P}}diag(\mathbf{k}^U)\bar{\mathbf{P}}^\mathsf{T}\mathbf{U}$

and $\mathbf{V}' = \bar{\mathbf{Q}}diag(\mathbf{k}^V)\bar{\mathbf{Q}}^\mathsf{T}\mathbf{V}$.

Transformation. We follow existing GNNs and GCNs (Zhou et al., 2018) to use a matrix $\mathbf{T} \in \mathbb{R}^{K \times K}$ to transform (not graph Fourier transform) the convolution of embeddings $\mathbf{U}'$ and $\mathbf{V}'$ as $\mathbf{U}'' = \mathbf{U}'\mathbf{T}$ and $\mathbf{V}'' = \mathbf{V}'\mathbf{T}$, respectively.

Activation. Experiments show that sigmoid function $\sigma(\ )$ is the best choice for LCFN: $\mathbf{U}''' = \sigma(\mathbf{U}'')$ and $\mathbf{V}''' = \sigma(\mathbf{V}'')$.

LCFN contains L layers and we use upper corner mark (l) to indicate parameters for the l-th layer. The input of each layer is the output of the previous layer, thus we have:

$$\begin{cases} \mathbf{U}^{(l)} = \sigma\left(\mathbf{P}diag\left(\mathbf{k}^{U^{(l)}}\right)\mathbf{P}^\mathsf{T}\mathbf{U}^{(l-1)}\mathbf{T}^{(l)}\right) \\ \mathbf{V}^{(l)} = \sigma\left(\mathbf{Q}diag\left(\mathbf{k}^{V^{(l)}}\right)\mathbf{Q}^\mathsf{T}\mathbf{V}^{(l-1)}\mathbf{T}^{(l)}\right) \end{cases}. \quad (4)$$

After L convolutional layers, we obtain embeddings in different level: $\left\{\mathbf{U}^{(l)}, \mathbf{V}^{(l)}\right\}_{l=0\cdots L}$. We concatenate them as the *predictive embeddings* to give the final prediction:

$$\hat{\mathbf{R}} = \left[\mathbf{U}^{(0)}, \cdots, \mathbf{U}^{(L)}\right]\left[\mathbf{V}^{(0)}, \cdots, \mathbf{V}^{(L)}\right]^\mathsf{T}. \quad (5)$$

To emphasize the effectiveness of our low-pass convolution, we choose the most simple yet widely-used way for embedding fusion (concatenation) and combination (inner product) (Wang et al., 2019; Zheng et al., 2018; Berg et al., 2018). Other advanced fusion and combination functions (He et al., 2017) are left to explore in future work.

5.2. Optimization and Prediction

To optimize our proposed model, we employ the BPR loss (Rendle et al., 2009). We construct pairwise training set $\mathcal{D} = \{(u,i,j)|\mathbf{R}_{ui} = 1 \bigwedge \mathbf{R}_{uj} = 0\}$ to learn the model. For a tuple $(u,i,j) \in \mathcal{D}$, we maximize the margin between u's preference towards i and j by minimizing the BPR loss:

$$\mathcal{L} = \sum_{(u,i,j)\in\mathcal{D}} -\ln\sigma\left(\hat{\mathbf{R}}_{ui} - \hat{\mathbf{R}}_{uj}\right) + \frac{\lambda}{2}\left(\left\|\mathbf{U}^{(0)}\right\|_F^2 + \left\|\mathbf{V}^{(0)}\right\|_F^2 + \|\Theta\|_F^2\right),$$

where $\hat{\mathbf{R}}$ is given in Equation (5). $\mathbf{U}^{(0)}$, $\mathbf{V}^{(0)}$ and $\Theta = \left\{\mathbf{k}^{U^{(l)}}, \mathbf{k}^{V^{(l)}}, \mathbf{T}^{(l)}\right\}_{l=1,\cdots,L}$ are trainable parameters. We learn them by minimizing $\mathcal{L}$ with Adam (Kingma & Ba, 2015). When recommending for the user u, we rank items by $\hat{\mathbf{R}}_u$ and return the top items to u. Of special notice is that by exploring the mean squared error (MSE) loss, our model can be extended to explicit feedbacks directly, which also suffer from the exposure and quantization noise.

5.3. Complexity Analysis

Eigen-decomposition. Original graph convolution requires all eigen-vectors of Laplacian matrices $\mathbf{L}^U$ and $\mathbf{L}^I$ as the transform bases, which costs $O(M^3 + N^3)$ time. In our low-pass convolution, only front Φ/Ψ eigen-vectors are needed.

For sparse matrices $\mathbf{L}^U$ and $\mathbf{L}^I$, the time complexity could be $O(M\Phi^2 + m\Phi + N\Psi^2 + n\Psi)$ with Lanczos method (Grimes et al., 1994; Paratte & Martin, 2016), where m/n is the number of nonzero elements of $\mathbf{L}^U/\mathbf{L}^I$. We cannot omit the linear term $m\Phi + n\Psi$ here because hypergraph Laplacian matrices are not so sparse, and $m\Phi/n\Psi$ is larger than $M\Phi^2/N\Psi^2$ when Φ/Ψ is small.

Model training. Low-pass convolution also makes training more efficient. For the original graph convolution, $\mathbf{U}' = \mathbf{P}diag(\mathbf{k}^U)\mathbf{P}^\mathsf{T}\mathbf{U}$ and $\mathbf{V}' = \mathbf{Q}diag(\mathbf{k}^V)\mathbf{Q}^\mathsf{T}\mathbf{V}$ cost $O(K(M^2 + N^2))$ time due to the multiplication with graph Fourier transform bases. In LCF, we use $\bar{\mathbf{P}}/\bar{\mathbf{Q}}$ to replace $\mathbf{P}/\mathbf{Q}$, the time complexity reduces to $O(K(M\Phi + N\Psi))$.

5.4. Comparison with Existing Models

In this subsection we compare our model with some widely-explored GNNs (Ying et al., 2018; Berg et al., 2018; Wang et al., 2019) and GCNs (Zheng et al., 2018; Yao & Li, 2018).

Comparison with GNNs. GNNs propagate signal (usually node embeddings) in the graph by the propagation matrix $\mathbf{D}^{-1}\mathbf{A}$ or $\mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$ (left or symmetric normalization, where $\mathbf{D}$ denotes the degree of the nodes and $\mathbf{A}$ is the adjacent matrix), and enhance the representation of each node with its neighbours. The propagation matrix contains one-hop connections thus the signal $\mathbf{S}$ is propagated to one-hop neighbours by $\mathbf{D}^{-1}\mathbf{A}\mathbf{S}$ in one GNN layer. Propagated by L layers, the signal of a certain node is enhanced by signals of its neighbours within L hops.

In each layer of our LCFN model, the convolution of the signal $\mathbf{S}$ is $\bar{\mathbf{P}}diag(\mathbf{k})\bar{\mathbf{P}}^\mathsf{T}\mathbf{S}$, where $\bar{\mathbf{P}}$ is the first Φ eigen-vectors of the Laplacian matrix and widely used as the node embedding (Ng et al., 2002). If we fix $\mathbf{k}$ to all-one vector $\mathbf{1}$ (i.e., we only filter the signal by LCF without convolution), $\bar{\mathbf{P}}\bar{\mathbf{P}}^\mathsf{T}$ is an advanced propagation matrix, where the propagations are weighted based on the connection strength by taking all-hop connections into consideration (Yu & Qin, 2019). Even without graph convolution, LCFN still works better than GNN. By learning the convolutional kernel, our LCFN gains further enhancement.

Comparison with GCNs. All existing GCNs avoid eigen-decomposition by simplifying graph convolution. Defferrard et al. (2016) first proposed an efficient GCN with the Chebyshev polynomial and Monti et al. (2017) leveraged this model in recommendation tasks. The convolutional kernel is reconstructed by a K-order polynomial: $\mathbf{k} = \sum_{k=0}^{K} \theta_k T_k(\mathbf{\Lambda})$ and the convolution of the signal $\mathbf{S}$ is $\mathbf{P}\left(\sum_{k=0}^{K} \theta_k T_k(\mathbf{\Lambda})\right)\mathbf{P}^\mathsf{T}\mathbf{S} = \sum_{k=0}^{K} \theta_k T_k(\mathbf{L})\mathbf{S}$. In this way, we do not need to construct $\mathbf{P}$ and $\mathbf{\Lambda}$. Since the Laplacian matrix $\mathbf{L} = \mathbf{I} - \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$ and $T_k(\mathbf{L})$ is a k-order polynomial of $\mathbf{L}$, the convolution becomes $p_K(\mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}})\mathbf{S}$, where $p_K(x)$ is the K-order polynomial of x. In this case, the simplified graph convolution is a linear combination of propagations within K hops. We can see that this GCN is indeed an advanced version of GNN, which can be achieved better by LCFN with only LCF. Another issue is that the density of $T_k(\mathbf{L})$ increases exponentially with the increasing of k, thus the order is very limited, or the convolution will be very space- and time-consuming.

A further simplification is proposed by Kipf & Welling (2017) and Monti et al. (2017); Zheng et al. (2018) then leveraged it in recommendation. Kipf & Welling (2017) fixed the kernel as $\theta(2\mathbf{I} - \mathbf{\Lambda})$, thus the convolution of $\mathbf{S}$ is $\mathbf{P}\theta(2\mathbf{I}-\mathbf{\Lambda})\mathbf{P}^\mathsf{T}\mathbf{S} = (\mathbf{I}+\mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}})\mathbf{S}\theta$. It is obvious that GCN degenerates to GNN thoroughly: The convolution of the signal is the combination of the signal itself and signals from one-hop neighbours. The parameter θ is regarded as the kernel and extended to a matrix $\mathbf{\Theta}$, nevertheless it is actually a transformation matrix widely used in GNNs (Berg et al., 2018; Ying et al., 2018; Wang et al., 2019) as well as in our model (matrix $\mathbf{T}$). Some GNNs (Gilmer et al., 2017; Hamilton et al., 2017) are declared equivalent to GCNs, unfortunately, the truth is by following Kipf & Welling (2017), most GCNs degenerate to GNNs.

In this paper, we provide a new strategy to simplify the computation. We only need to construct front eigen-vectors of the Laplacian matrix, which can be solved in acceptable time. In existing work, the kernel is not trainable (Kipf & Welling, 2017; Monti et al., 2017; Zheng et al., 2018) or partially trainable (Defferrard et al., 2016; Monti et al., 2017), which cripples the graph convolution seriously. In our LCFN model, the convolutional kernel is fully trained from the data. Another point to support that our model is more powerful is that: in all existing GCNs, the signal $\mathbf{S}$ is multiplied with a sparse matrix, while in LCFN, the signal $\mathbf{S}$ is multiplied with a dense convolution operator.

6. Experiment

Experiments are conducted in this section to demonstrate the effectiveness and efficiency of LCFN by comparing it against several state-of-the-art models on two real-world datasets. We focus on answering three research questions:

RQ1. How is the performance of our LCFN model?

RQ2. How is the efficiency improvement by LCF?

RQ3. How is the effectiveness improvement by LCF?

6.1. Experimental Setup

Datasets. We adopt two real-world datasets, *Amazon* and *Movielens*, to evaluate our model.

Table 1. Statistics of datasets

Dataset	Purchase	User	Item	Sparsity
Amazon	347,393	20,247	11,589	99.8519%
Movielens	995,154	6,022	3,043	94.5694%

- **Amazon.** This *Amazon* dataset (He & McAuley, 2016a) is the user reviews collected from the E-commerce website *Amazon.com*. In this paper we adopt the *Electronic* category.
- **MovieLens.** This *Movielens* dataset (Harper & Konstan, 2015) is collected through the movie website *movielens.umn.edu* and we use the 1M version.

These two datasets are all explicit feedbacks, we set the interaction (u, i) as "1" if u rated i and "0" otherwise. We filter *Amazon* and *Movielens* with 10- and 20-core respectively (when filtered with n-core, we remove users and items within n interactions) to reduce the sparsity. Datasets after filtering are shown in Table 1. Each dataset is split into training set (80%), validation set (10%), and test set (10%) randomly. We train models on the training set, determine all hyperparameters on the validation set, and report the performances on the test set.

Baselines. We adopt the following methods as baselines for performance comparison.

- **MF:** This **M**atrix **F**actorization method encodes users and items with embeddings by factorizing the observed matrix (Koren, 2009). It is a basic yet competitive baseline for item recommendation.
- **GCMC:** This **G**raph **C**onvolutional **M**atrix Completion is a GNN-based recommendation model which utilizes a graph auto-encoder to propagate features to neighbours (Berg et al., 2018). In our experiments, features are learnable embeddings and we set $R = 1$ for implicit feedbacks.
- **NGCF:** This **N**eural **G**raph **C**ollaborative **F**iltering method is a GNN model which propagates embeddings through the graph to refine the representation of users and items (Wang et al., 2019).
- **SCF:** This **S**pectral **C**ollaborative **F**iltering method is the state-of-the-art GCN model for recommendation (Zheng et al., 2018). The convolutional kernel is fixed to $\theta(\mathbf{I} + \mathbf{\Lambda})$. By extending θ to $\mathbf{\Theta}$, the convolution of a signal $\mathbf{S}$ is $(2\mathbf{I} - \mathbf{D}^{-1}\mathbf{A})\mathbf{S}\mathbf{\Theta}$.
- **CGMC:** This **C**onvolutional **G**eometric **M**atrix Completion model is also a GCN model, which leverages graph convolution to extract features (Yao & Li, 2018). The kernel is set to $\theta(\mathbf{I} - (1-\sigma)\mathbf{\Lambda})$ and the convolution of a signal $\mathbf{S}$ is $(\sigma\mathbf{I} + (1-\sigma)\mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}})\mathbf{S}\mathbf{\Theta}$, where σ is a hyperparameter to balance self signal and neighbour signals.

Noting that MF, GCMC, CGMC are designed for explicit feedbacks, we use BPR (Rendle et al., 2009) to optimize all models in our experiments.

Evaluation Protocols. To evaluate the performances of our proposed model and baselines, we rank all items for each user in validation/test set and recommend the top-2 items to the user. We then adopt two metrics, F_1-score@2 and normalized discounted cumulative gain (NDCG@2) to evaluate the performance (results for top-$\{2, 5, 10, 20, 50, 100\}$ recommendation can be found in Appendix C). F_1-score is extensively used to test the accuracy of binary classifier and NDCG is a position-sensitive metric widely used to measure the ranking quality. We recommend top-2 items to each user and calculate metrics, and finally use the average metrics of all users as the model performance.

Parameter Setting. To compare fairly, all models in our experiments are tuned with the following strategies: The maximum iteration number is set to 200. In each iteration, we enumerate all positive samples to train the model and then test it. The learning rate η and the regularization coefficient λ are determined by grid searching in the coarse grain range of $\{0.0001, 0.001, 0.01\} \otimes \{0.001, 0.01, 0.1\}$ and then in the fine grain range, which is based on the result of coarse tuning. For example, if a certain model achieves the best performance when $\eta = 0.01$ and $\lambda = 0.1$, we then tune it in the range of $\{0.002, 0.005, 0.01, 0.02, 0.05\} \otimes \{0.02, 0.05, 0.1, 0.2, 0.5\}$. The batch size is set as 10,000 to utilize GPU. We evaluate different embedding dimensionality D in the range of $\{8, 16, \cdots, 128\}$. The layer number L is determined in the range of $\{1, 2, \cdots, 5\}$. To explore the best passband cut-off frequency, we define $F = \frac{\Phi}{M} = \frac{\Psi}{N}$, and tune our model with respect to F in the range of $\{0.0002, 0.0005, 0.001, \cdots, 0.2\}$. The experiments are repeated 5 times for each parameter setting in model tuning and 10 times for testing. The embedding layer of all deep models in our experiments are initialized by pretrained MF.

6.2. Performance of LCFN (RQ1)

Performances of all models on two datasets are show in Table 2. "Imp" means relative improvement and "BB" means the best baseline. In GNNs and GCNs, the graph is used to provide connection information, thus we should gain performance improvement over MF. However, to our surprise, MF is a very competitive baseline and GNN/GCN baselines only outperform it very slightly when compared fairly. In these baselines, GCMC and NGCF are GNNs while SCF and CGMC are GCNs. As we discussed, all existing GCNs lose the ability of graph convolution and degenerate to GNNs, or advanced version of GNN but with the same mechanism. This viewpoint is also supported by the experiments — SCF and CGMC perform almost the same with GCMC
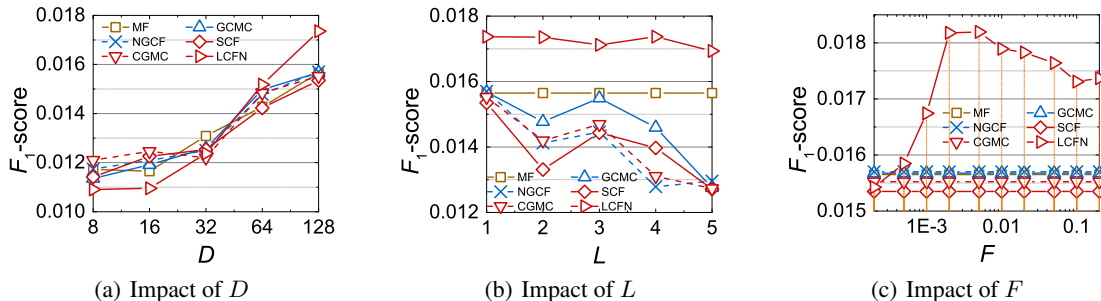
Figure 5. Model tuning (*Amazon*, validation set).

and NGCF. Utilizing unscathed graph convolution to extract features, LCFN gains significant improvement over existing GNNs and GCNs: LCFN outperforms BB 16.18% on F_1-score and 17.71% on NDCG for the best case.

Table 2. Recommendation performance (test set)

Model	Amazon		Movielens	
	F_1-score	NDCG	F_1-score	NDCG
MF	0.01356	0.01742	0.07506	0.25100
GCMC	0.01377	0.01742	0.07689	**0.26316**
NGCF	0.01349	0.01733	0.07453	0.25436
SCF	0.01424	0.01787	0.07808	0.26173
CGMC	0.01363	0.01728	0.07705	0.25790
LCFN	**0.01654**	**0.02104**	**0.08151**	0.26129
Imp MF	22.00%	20.77%	8.60%	4.10%
Imp BB	16.18%	17.71%	4.40%	-0.71%

Comparing CGMC and SCF, we can see that the kernel $\theta(\mathbf{I}+\mathbf{\Lambda})$ is better than $\theta(\mathbf{I}-(1-\sigma)\mathbf{\Lambda})$. For existing GCNs, since the convolutional kernels are fixed, we need to design better kernels based on our experience yet the improvement is usually very marginal. In the computer vision domain, the evolution from hand-crafted features such as Histogram of Oriented Gradient (HOG) (Dalal & Triggs) to end-to-end models such as CNN (Krizhevsky et al., 2012) shows that the power of CNN is derived from learning the kernels from the data. In our LCFN, we leverage the original graph convolution and play its roles by designing an end-to-end GCN. Experiments also show that our learnable kernels are more powerful than the hand-crafted kernels.

6.3. Model Tuning

The result of model tuning is shown in Figure 5. For a fair comparison, we tune all models with the same strategy introduced in Subsection 6.1. Due to the limited space, we only report experiments on *Amazon* dataset on account of the similar performances on the two datasets. To make sure all models achieve their best performance for each hyperparameter setting, we retune them with respect to η and λ and initialize the deep models with pretrained MF for each D, L, and F in Figure 5.

Figure 5(a) illustrates how the predictive embedding dimensionality D impacts the models. Denoting K is the embedding dimensionality of the input layer and L is the depth of the models, we define $D = K$ for MF and $D = (L+1)K$ for all GNNs/GCNs, where the predictive embeddings are the concatenation of embeddings from the input layer and L convolutional/propagation layers. As shown in the figure, we gain better performance by increasing the representation ability. In this tuning procedure, we fix $L = 1$ and $F = 0.2$.

The number of layers L is analyzed in Figure 5(b). Berg et al. (2018) fixed L to 1 in GCMC yet it is an experimental conclusion depending on specific datasets, thus we also explore the best L for GCMC in our experiments. In GNNs and GCNs, with the increasing of L, D also increases, thus we are not sure if the improvement is caused by the increasing of L or of D. To control variables, we fix D to 128 when setting different L. From Figure 5(b) we can see that the properties of GNN and GCN baselines are pretty similar, this is also experimental evidence of our key viewpoint: existing GCNs degenerate to GNNs. LCFN shows different properties because we really perform convolution rather than propagating embeddings. Figure 5(b) also shows that high-order propagation is harmful, which is supported by Berg et al. (2018). Wang et al. (2019) got the different conclusion because they did not control variables. As LCFN gains no improvement with a deeper structure, we set $L = 1$ for efficiency. In this procedure, we fix $D = 128$ and $F = 0.2$.

Figure 5(c) shows the sensitivity analysis of the cut-off frequencies F. When F is too small, the signal is over-filtered and much useful information is corrupted. However, when F is too large, the noise is reserved hence the performance also decreases. LCFN achieves the best performance when $F = 0.005$. In this procedure, we fix $D = 128$ and $L = 1$.

In our experiments, all deep models are pretrained to get better performance. Here we also discuss the effectiveness of pretraining. In Table 3, "No-pre" means no pretraining, "Pre" means pretraining, and "Imp" means relative improvement. Please note that we also retune all models to make
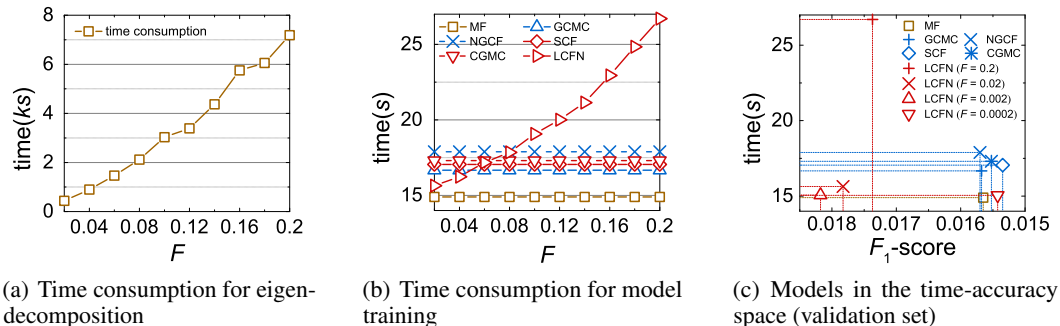
(a) Time consumption for eigen-decomposition

(b) Time consumption for model training

(c) Models in the time-accuracy space (validation set)

Figure 6. Efficiency and Effectiveness improvement of LCF.

Table 3. Effectiveness of pretraining (F_1-score, validation set)

Model	Amazon			Movielens		
	No-pre	Pre	Imp	No-pre	Pre	Imp
GCMC	0.01466	0.01568	6.91%	0.07288	0.07828	7.42%
NGCF	0.01190	0.01570	31.93%	0.07031	0.07877	12.03%
SCF	0.01149	0.01535	33.59%	0.07300	0.07760	6.29%
CGMC	0.01265	0.01552	22.69%	0.07169	0.07700	7.41%
LCFN	0.01743	0.01819	4.38%	0.08013	0.07994	-0.24%

sure they achieve their best performances without pretraining. Compared with baselines, another advantage of LCFN is that it dose not rely on pretraining.

Comparing with Tables 2 and 3, we can observe an interesting phenomenon. NDCG performs pretty well on the validation set yet performs poorly on the test set, while for SCF, we face the opposite situation. Compared with the basic GNN model GCMC, NGCF additionally propagates signals based on element-wise preference similarity, hence shows better performance on the validation set. However, this strategy may make the model easily get overfitted to the validation set. Benefiting from the well-designed kernel, SCF shows better generalization and performs the best among all baselines on the test set.

6.4. Efficiency Enhancement by LCF (RQ2)

As we analyzed in Subsection 5.3, LCF improves the efficiency of GCN. In this subsection, we report some experimental evidence.

The efficiency improvement are caused by two aspects: constructing transform bases (eigen-decomposition), and model training. Figure 6(a) shows the time consumption of eigen-decomposition with different F. The time complexity is $O((M^3 + N^3)F^2 + (mM + nN)F)$, and Figure 6(a) shows that the time complexity is mainly determined by the linear term of F. As shown in Figure 5(c), LCFN achieves the best performance when $F = 0.005$. In this case, we can save considerable time. The time consumption of one iteration during training is illustrated in Figure 6(b). The training efficiency of LCFN is also satisfactory: When $F = 0.005$, LCFN is obviously faster than all GNN/GCN baselines.

Since the datasets are too large, the original graph convolution ($F = 1$) is inapplicable. However, from the trend of the curves in Figures 6(a)(b), we can infer that we improve the efficiency significantly than the GCN with the original graph convolution.

6.5. Effectiveness Enhancement by LCF (RQ3)

The effectiveness improvement of LCF has been represented in Figure 5(c): We gain dramatic improvement by setting a small passband cut-off frequency. We further represent all models in the time-accuracy space in Figure 6(c). The horizontal axis represents the time consumption of one iteration in training, and the vertical axis represents F_1-score. Noting that F_1-score is in descending order, models in the lower left area are better. By tuning appropriate F, LCFN outperforms GNN/GCN baselines significantly in both time and accuracy aspects.

7. Conclusion and Future Work

In this paper, we first designed a LCF to remove the high-frequency exposure noise and quantization noise from the observed interaction matrix to uncover the underlying rating matrix. We then proposed the 2D graph convolution by extending convolution from the Euclidean domain to the graph domain. Finally, we designed a deep GCN and injected the LCF into it. LCF removes the noise to improve the prediction accuracy as well as provides a new way to design applicable graph convolution in an unscathed way.

For future work, we have interests in validating the effectiveness of LCFN in other tasks, such as social network, knowledge graph, and even computer vision, since images are also 2D low-frequency signals contaminated by high-frequency noise. Moreover, we want to inject side information into LCFN, such as user reviews, visual features. We also want to propose multi-graph LCFN. We construct graphs by such as tags (Chen et al., 2016) or item transfer Morkov chain (Rendle et al., 2010), and use these graphs for convolution in multiple channels.

References

Barrett, L. C. and Wilde, C. A power booktitle development of the convolution theorem. *American Mathematical Monthly*, pp. 893–894, 1960.

Berg, R. V. D., Kipf, T. N., and Welling, M. Graph convolutional matrix completion. In *KDD*, 2018.

Chen, J., Zhang, H., He, X., Nie, L., Liu, W., and Chua, T.-S. Attentive collaborative filtering: Multimedia recommendation with item- and component-level attention. In *SIGIR*, pp. 335–344, 2017.

Chen, X., Qin, Z., Zhang, Y., and Xu, T. Learning to rank features for recommendation over multiple categories. In *SIGIR*, pp. 305–314, 2016.

Cheng, H.-T., Koc, L., Harmsen, J., Shaked, T., Chandra, T., Aradhye, H., Anderson, G., Corrado, G., Chai, W., Ispir, M., Anil, R., Haque, Z., Hong, L., Jain, V., Liu, X., and Shah, H. Wide & deep learning for recommender systems. In *DLRS*, pp. 7–10, 2016.

Dalal, N. and Triggs, B. Histograms of oriented gradients for human detection. In *CVPR*.

Defferrard, M., Bresson, X., and Vandergheynst, P. Convolutional neural networks on graphs with fast localized spectral filtering. In *NIPS*, pp. 3837–3845, 2016.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. Neural message passing for quantum chemistry. In *ICML*, volume 70, pp. 1263–1272, 2017.

Goldberg, D. Using collaborative filtering to weave an information tapestry. *Communications of the Acm*, pp. 61–70, 1992.

Grimes, R. G., Lewis, J. G., and Simon, H. D. A shifted block lanczos algorithm for solving sparse symmetric generalized eigenproblems. *Siam Journal on Matrix Analysis & Applications*, pp. 228–272, 1994.

Hamilton, W., Ying, Z., and Leskovec, J. Inductive representation learning on large graphs. In *NIPS*, pp. 1024–1034. 2017.

Harper, F. M. and Konstan, J. A. The movielens datasets: History and context. *ACM Trans. Interact. Intell. Syst.*, pp. 19:1–19:19, 2015.

He, R. and McAuley, J. Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. In *WWW*, pp. 507–517, 2016a.

He, R. and McAuley, J. VBPR: visual bayesian personalized ranking from implicit feedback. In *AAAI*, pp. 144–150, 2016b.

He, X., Liao, L., Zhang, H., Nie, L., Hu, X., and Chua, T.-S. Neural collaborative filtering. In *WWW*, pp. 173–182, 2017.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In *ICLR*, 2015.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.

Koren, Y. The bellkor solution to the netflix grand prize. *Netflix prize documentation*, pp. 1–10, 2009.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In *NIPS*, pp. 1106–1114, 2012.

Li, J., Ren, P., Chen, Z., Ren, Z., Lian, T., and Ma, J. Neural attentive session-based recommendation. In *CIKM*, pp. 1419–1428, 2017.

Monti, F., Bronstein, M. M., and Bresson, X. Geometric matrix completion with recurrent multi-graph neural networks. In *NIPS*, pp. 3697–3707, 2017.

Ng, A. Y., Jordan, M. I., and Weiss, Y. On spectral clustering: Analysis and an algorithm. In *NIPS*, pp. 849–856, 2002.

Paratte, J. and Martin, L. Fast eigenspace approximation using random signals. *Computing Research Repository - CoRR*, 2016.

Rao, N., Yu, H.-F., Ravikumar, P. K., and Dhillon, I. S. Collaborative filtering with graph information: Consistency and scalable methods. In *NIPS*, pp. 2107–2115, 2015.

Rendle, S., Freudenthaler, C., Gantner, Z., and Schmidt-Thieme, L. BPR: bayesian personalized ranking from implicit feedback. In *UAI*, pp. 452–461, 2009.

Rendle, S., Freudenthaler, C., and Schmidt-Thieme, L. Factorizing personalized markov chains for next-basket recommendation. In *WWW*, pp. 811–820, 2010.

Sarwar, B., Karypis, G., Konstan, J., and Riedl, J. Item-based collaborative filtering recommendation algorithms. In *WWW*, pp. 285–295, 2001.

Shuman, D. I., Narang, S. K., Frossard, P., Ortega, A., and Vandergheynst, P. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE Signal Processing Magazine*, pp. 83–98, 2013.

Tan, Y. K., Xu, X., and Liu, Y. Improved recurrent neural networks for session-based recommendations. In *DLRS*, pp. 17–22, 2016.

Wang, X., He, X., Wang, M., Feng, F., and Chua, T.-S. Neural graph collaborative filtering. In *SIGIR*, 2019.

Yao, K.-L. and Li, W.-J. Convolutional geometric matrix completion. *CoRR*, 2018.

Ying, R., He, R., Chen, K., Eksombatchai, P., Hamilton, W. L., and Leskovec, J. Graph convolutional neural networks for web-scale recommender systems. In *KDD*, pp. 974–983, 2018.

Yu, W. and Qin, Z. Spectrum-enhanced pairwise learning to rank. In *WWW*, pp. 2247–2257, 2019.

Yu, W., Zhang, H., He, X., Chen, X., Xiong, L., and Qin, Z. Aesthetic-based clothing recommendation. In *WWW*, pp. 649–658, 2018.

Zheng, L., Lu, C.-T., Jiang, F., Zhang, J., and Yu, P. S. Spectral collaborative filtering. In *RecSys*, pp. 311–319, 2018.

Zhou, D., Huang, J., and Schölkopf, B. Learning with hypergraphs: Clustering, classification, and embedding. In *NIPS*, pp. 1601–1608, 2006.

Zhou, J., Cui, G., Zhang, Z., Yang, C., Liu, Z., and Sun, M. Graph neural networks: A review of methods and applications. *CoRR*, 2018.