How Powerful is Graph Convolution for Recommendation?

Yifei Shen[1], Yongji Wu[2], Yao Zhang[3], Caihua Shan[4], Jun Zhang[1], Khaled B. Letaief[1], Dongsheng Li[4]
[1]HKUST, [2]Duke University, [3]Fudan University, [4]Microsoft Research Asia
yshenaw@connect.ust.hk, wuyongji317@gmail.com, {yaozhang,dongshengli}@fudan.edu.cn,
{eejzhang, eekhaled}@ust.hk, caihuashan@microsoft.com

ABSTRACT

Graph convolutional networks (GCNs) have recently enabled a popular class of algorithms for collaborative filtering (CF). Nevertheless, the theoretical underpinnings of their empirical successes remain elusive. In this paper, we endeavor to obtain a better understanding of GCN-based CF methods via the lens of graph signal processing. By identifying the critical role of smoothness, a key concept in graph signal processing, we develop a unified graph convolution-based framework for CF. We prove that many existing CF methods are special cases of this framework, including the neighborhood-based methods, low-rank matrix factorization, linear auto-encoders, and LightGCN, corresponding to different low-pass filters. Based on our framework, we then present a simple and computationally efficient CF baseline, which we shall refer to as Graph Filter based Collaborative Filtering (GF-CF). Given an implicit feedback matrix, GF-CF can be obtained in a closed form instead of expensive training with back-propagation. Experiments will show that GF-CF achieves competitive or better performance against deep learning-based methods on three well-known datasets, notably with a 70% performance gain over LightGCN on the Amazon-book dataset.

CCS CONCEPTS

• **Information systems** → **Recommender systems**.

KEYWORDS

collaborative filtering, graph convolution, graph signal processing

ACM Reference Format:
Yifei Shen[1], Yongji Wu[2], Yao Zhang[3], Caihua Shan[4], Jun Zhang[1], Khaled B. Letaief[1], Dongsheng Li[4] . 2021. How Powerful is Graph Convolution for Recommendation?. In *CIKM '21: ACM International Conference on Information and Knowledge Management, Nov 01–05, 2021, Queensland, Australia.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3459637.3482264

1 INTRODUCTION

Recommender systems have achieved great successes in many businesses, e.g., for product recommendation on Amazon [25] and playlist generation on Youtube [7], etc. As the algorithmic effectiveness will have a direct impact on the commercial success, building a good recommendation engine, especially via collaborative filtering (CF), remains an active research area, with consistent innovations in both conventional methods [6, 21, 31] and recently emerged deep learning approaches [13, 15, 24].

Over the past decade, we have witnessed great progress in CF algorithms. Model-based methods largely resort to low-dimensional structures in high-dimensional data [42], e.g., low-rank matrix factorization [6, 17, 22, 28] and autoencoders [24, 36, 43]. On the other hand, neighborhood-based methods [1, 39] achieve competitive performance based on simple similarity measures, e.g., the cosine similarity between items. Furthermore, these two types of methods can be incorporated together to improve the performance, e.g., SVD++ [21]. From the graph perspective, the neighborhood-based methods and SVD++ effectively exploit the one-hop information in the user-item interaction graph.

To take advantage of the rich multi-hop neighborhood information, graph convolutional networks (GCNs), e.g., GC-MC [4], NGCF [41], LightGCN [13], have been recently proposed and become state-of-the-art methods for CF. NGCF [41] was inspired by the GCNs developed for attribute graphs [19], and it inherits the key ingredients from GCNs, including initial embeddings, feature transformation, neighborhood aggregation, and nonlinear activation. As the graphs in CF tasks are non-attributed, these operations may not be necessary [13]. Therefore, in LightGCN [13], only the most important components, i.e., trained initial embeddings and graph convolution, are preserved. Removing the unnecessary components leads to easier training and better generalization [46, 47], and thus LightGCN significantly outperforms NGCF in both accuracy and efficiency. While these empirical studies have produced promising results, the underlying reasons for the effectiveness of these methods remain elusive. From the theoretical perspective, an intriguing question is *what plays an essential role in the success of GCN-based methods for CF*. From the practical perspective, it is interesting to investigate *to what extent we can reduce the training cost while effectively exploiting the rich information of the user-item interaction graph*.

This paper endeavors to obtain a better understanding of GCN-based methods and develop a unified framework based on graph convolution that incorporates classic methods. In particular, we identify the importance of a key concept in graph signal processing in developing CF algorithms, namely, *smoothness*. Conceptually, if a user interacted with an item, then their embeddings should be similar. In graph signal processing, the similarity between the embeddings of the interacted user-item pair defines the smoothness of the embedding. Meanwhile, low-pass filters on graphs, e.g., the light convolution in LightGCN [13], are used to promote the smoothness of graph signals. We will therefore argue that it is the smoothness of the embeddings and the low-pass filtering that play a pivotal role in GCN-based methods. By theoretical analysis

This work was done when the first three authors were interns with Microsoft Research Asia. This work was supported in part by the Hong Kong Research Grants Council under Grant No. 16210719.
CIKM '21, Nov 01–05, 2021, Queensland, Australia

ACM ISBN 978-1-4503-8446-9/21/11…$15.00
https://doi.org/10.1145/3459637.3482264

and experiments, we will show that the performance of *untrained* LightGCN is competitive to a trained one when the embedding dimension is sufficiently large, due to the smoothing effect of the light convolution. Inspired by this finding, we derive a closed-form solution for the untrained LightGCN with Infinitely Dimensional Embedding (LGCN-IDE). It is shown that LGCN-IDE outperforms LightGCN by more than 40% on the Amazon-book dataset.

Motivated by its simplicity and effectiveness, we extend LGCN-IDE to incorporate general low-pass filters, which form a unified framework for CF. Surprisingly, it is proved that the neighborhood-based methods [1], low-rank matrix factorization [6], and linear auto-encoders [36] are all special cases of this framework with various classic low-pass filters. This finding verifies the effectiveness of graph convolution with low-pass filters for CF. We further present a simple and computationally efficient CF method, which is an integration of linear filters and an ideal low-pass filter. Given an implicit feedback matrix, our proposed method has a closed-form solution and as such it would not require expensive training. More importantly and despite of its simplicity, the proposed method achieves competitive or better performance compared with deep learning methods.

To summarize, this work has made the following contributions.

(1) By identifying the critical role of the smoothness and low-pass filtering, we provide a novel perspective to understand the algorithms for CF.
(2) Using both theoretical justification and experiments, we show that the *untrained* LightGCN can achieve competitive performance as a trained one when the embedding dimension is sufficiently large. We further derive a closed-form solution for untrained LightGCN with infinitely dimensional embedding.
(3) Built upon the closed-form solution, we develop a general graph filter-based framework for CF. We prove that the neighborhood-based methods, linear auto-encoders, and low-rank matrix factorization are special cases of this framework, corresponding to various classic low-pass filters.
(4) We present a simple and computationally efficient method, named GF-CF. With a small fraction of training time, GF-CF achieves competitive or higher performance compared with the state-of-the-art deep learning methods on three well-known datasets.

The code to reproduce the experiments is available at https://github.com/yshenaw/GF_CF.

2 PRELIMINARIES

2.1 Notations and Terminology

This subsection presents some useful notations and definitions. We first define user set $\mathcal{U}$ and item set $\mathcal{I}$. As in [13], this paper considers the recommendation problem with implicit feedback. The implicit feedback matrix $\boldsymbol{R} \in \{0,1\}^{|\mathcal{U}|\times|\mathcal{I}|}$ is defined as follows:

$$\boldsymbol{R}_{u,i} = \begin{cases} 1, & \text{if } (u,i) \text{ interaction is observed,} \\ 0, & \text{otherwise,} \end{cases}$$

and $\boldsymbol{r}_u$ denotes the u-th row of $\boldsymbol{R}$.

The adjacency matrix of the user-item interaction graph is given by

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{0} & \boldsymbol{R} \\ \boldsymbol{R}^T & \boldsymbol{0} \end{bmatrix}. \tag{1}$$

In this bipartite graph, we denote the neighbors of node k as $\mathcal{N}_k$, and its cardinality as $N_k = |\mathcal{N}_k|$.

We denote the all one column vector of any dimension as $\boldsymbol{1}$, and degree matrices as $\boldsymbol{D}_U = \text{Diag}(\boldsymbol{R}\cdot\boldsymbol{1})$ and $\boldsymbol{D}_I = \text{Diag}(\boldsymbol{1}^T\boldsymbol{R})$. The normalized rating matrix is denoted as

$$\tilde{\boldsymbol{R}} = \boldsymbol{D}_U^{-\frac{1}{2}}\boldsymbol{R}\boldsymbol{D}_I^{-\frac{1}{2}},$$

with $\tilde{\boldsymbol{r}}_u$ as the u-th row of $\tilde{\boldsymbol{R}}$. Similarly, the normalized user-item adjacency matrix is given by

$$\tilde{\boldsymbol{A}} = \begin{bmatrix} \boldsymbol{0} & \tilde{\boldsymbol{R}} \\ \tilde{\boldsymbol{R}}^T & \boldsymbol{0} \end{bmatrix}.$$

We also define the item-item normalized adjacency matrix as

$$\tilde{\boldsymbol{P}} = \tilde{\boldsymbol{R}}^T\tilde{\boldsymbol{R}}.$$

We then define an important concept, namely, Stiefel manifold, which can help to connect low-rank matrix factorization and GCN-based methods in Section 4.2.

Definition 2.1. (Stiefel manifold) The Stiefel manifold $\text{St}(n,m)$ is defined as the subspace of orthonormal N-frames in $\mathbb{R}^n$, namely,

$$\text{St}(n,m) = \{\boldsymbol{\Gamma} \in \mathbb{R}^{n\times m} : \boldsymbol{\Gamma}^T\boldsymbol{\Gamma} = \boldsymbol{I}\} \tag{2}$$

where $\boldsymbol{I}$ is the identity matrix.

2.2 Graph Signal Processing

In this subsection, we introduce basic concepts of graph signal processing [8, 29]. We consider a weighted undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with n nodes where $\mathcal{V}$ and $\mathcal{E}$ denote the vertex set and edge set, respectively. The graph can be represented as an adjacency matrix $\boldsymbol{A} \in \mathbb{R}^{n\times n}$. A graph signal is defined as a function $x : \mathcal{V} \to \mathbb{R}$ and it can be represented as a n-dimensional vector $\boldsymbol{x} = [x(i)]_{i\in\mathcal{V}}$. For a graph signal, the derivative is defined as $(\nabla\boldsymbol{x})_{i,j} = \sqrt{A_{i,j}}(x_i - x_j)$.

The smoothness of a graph signal can be measured by the *graph quadratic form*, which is the squared norm of the graph derivative as defined below:

$$S(\boldsymbol{x}) = \frac{1}{2}\|\nabla\boldsymbol{x}\|_F^2 = \sum_{i,j} A_{i,j}(x_i - x_j)^2 = \boldsymbol{x}^T\boldsymbol{L}\boldsymbol{x}.$$

Here, $\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{A}$ is the graph Laplacian matrix[1]. A smaller $\frac{S(\boldsymbol{x})}{\|\boldsymbol{x}\|_2}$ indicates smoother signals.

In many applications, a graph signal is often described in a vector form $x : \mathcal{V} \to \mathbb{R}^d$ and its smoothness can be written as follows:

$$S_2(\boldsymbol{x}) = \sum_{i,j} A_{i,j}\|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2^2. \tag{3}$$

As $\boldsymbol{L}$ is real and symmetric, its eigendecomposition is given by $\boldsymbol{L} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^T$ where $\boldsymbol{\Lambda} = \text{Diag}(\lambda_1, \cdots, \lambda_n)$, $\lambda_1 \le \lambda_2 \le \cdots \le \lambda_n$, and $\boldsymbol{U} = [\boldsymbol{u}_1, \cdots, \boldsymbol{u}_n]$ with $\boldsymbol{u}_i \in \mathbb{R}^n$ being the eigenvector for eigenvalue λ_i.

[1] The graph Laplacian matrix can also be defined by some normalized version of $\boldsymbol{D} - \boldsymbol{A}$, e.g., $\tilde{\boldsymbol{L}} = \boldsymbol{I} - \tilde{\boldsymbol{A}}$.

Next we discuss the frequency of the graph signal and define Fourier Transform on graphs. Intuitively, the graph signal has a higher frequency if it is more oscillatory and not smooth. As λ_1 is the smallest eigenvalue, for any graph signal $\boldsymbol{x} \in \mathbb{R}^n$, we have $\frac{S(\boldsymbol{x})}{\|\boldsymbol{x}\|_2} \geq \frac{S(\boldsymbol{u}_1)}{\|\boldsymbol{u}_1\|_2}$. Thus, the eigenvector with a smaller eigenvalue corresponds to a lower frequency signal component. We can define the Graph Fourier Transform (GFT) basis as the eigenvector matrix $\boldsymbol{U}$ and we call $\hat{\boldsymbol{x}} = \boldsymbol{U}^T\boldsymbol{x}$ as the GFT of the graph signal $\boldsymbol{x}$. Similar to the Fourier transform, GFT is a linear orthogonal transform and its inverse transform is given by $\boldsymbol{x} = \boldsymbol{U}\hat{\boldsymbol{x}}$. GFT enables us to define graph filters and graph convolution.

Definition 2.2. (Graph Filter) Given a graph Laplacian matrix, as well as its eigenvectors and eigenvalues, then the graph filter $\mathcal{H}(\boldsymbol{L})$ is defined as follows:

$$\mathcal{H}(\boldsymbol{L}) = \boldsymbol{U}\text{Diag}(h(\lambda_1), \cdots, h(\lambda_n))\boldsymbol{U}^T.$$

where $h(\cdot)$ is the filter defined on the eigenvalues.

Definition 2.3. (Graph Convolution) The graph convolution of a input signal $\boldsymbol{x}$ and the filter $\mathcal{H}(\boldsymbol{L})$ is defined as follows:

$$\boldsymbol{y} = \mathcal{H}(\boldsymbol{L})\boldsymbol{x} = \boldsymbol{U}\text{Diag}(h(\lambda_1), \cdots, h(\lambda_n))\boldsymbol{U}^T\boldsymbol{x}.$$

Similar to the definition of convolution in classic signal processing, the graph signal is transformed by GFT $\boldsymbol{U}^T$, multiplied by a filter $h(\cdot)$, and transformed back by inverse GFT $\boldsymbol{U}$. In the context of CF, the graph signal is often the observed ratings for a given user [6], or the initial embeddings of users/items [13, 41].

In the signal processing literature, the signal is often smooth and with low frequency, and the noise is often non-smooth and with a high frequency. One important class of filters is the *low-pass* filters, which promotes smoothness of graph signals for denoising. The graph low-pass filters are defined as follows.

Definition 2.4. (Low-pass Filter) For $k = 1, \cdots, n-1$, we define the ratio

$$\eta_k := \frac{\max\{|h(\lambda_{k+1})|, \cdots, |h(\lambda_n)|\}}{\min\{|h(\lambda_1)|, \cdots, |h(\lambda_k)|\}}. \tag{4}$$

The graph filter $\mathcal{H}(\boldsymbol{L})$ is k-low-pass if and only if the low-pass ratio satisfies $\eta_k \in [0, 1)$.

The low-pass ratio defines how much of the high-frequency component of a signal is allowed to pass compared to the low-frequency components. If $\eta_k < 1$, then the filter passes low-frequency signals and is called a low-pass filter. We here list some important low-pass filters and will connect these filters with classic methods for recommendation in Section 4.2.

Linear Filter. The linear filter is given by

$$h(\lambda_i) = \sum_{k=0}^{K} \alpha_k \lambda_i^k, \tag{5}$$

where α_k is the filter's coefficient. It is called *linear* due to its similarity with linear time invariant filters in classic signal processing. We will show that this filter corresponds to LightGCN and neighborhood-based methods.

Ideal Low-pass Filter. The ideal low-pass filter has a cut-off frequency $\bar{\lambda}$. The filter is defined as

$$h(\lambda_i) = \begin{cases} 1, & \text{if } \lambda_i \leq \bar{\lambda} \\ 0, & \text{otherwise.} \end{cases} \tag{6}$$

It is called *ideal* as the high-frequency signals are ideally cut off with no leakage. We will show that this filter corresponds to the low-rank matrix factorization method.

Opinion Dynamics. The opinion dynamics are a *graph diffusion* process, which is a GF-AR(1) model [11]: $\boldsymbol{y}_{t+1} = (1-\beta)(\boldsymbol{I} - \alpha\boldsymbol{L})\boldsymbol{y}_t + \beta\boldsymbol{x}_t$. The steady state opinions are given by $\boldsymbol{y} = \lim_{t\to\infty} \boldsymbol{y}_t = (\boldsymbol{I} + \tilde{\alpha}\boldsymbol{L})^{-1}\boldsymbol{x} = \mathcal{H}(\boldsymbol{L})\boldsymbol{x}$ where $\tilde{\alpha} = \beta(1-\alpha)\alpha$. Thus, the corresponding graph filter is

$$h(\lambda_i) = \frac{1}{1 + \tilde{\alpha}\lambda_i}. \tag{7}$$

In opinion dynamics, the matrix inverse or eigenvalue decomposition is required. Thus, applying this filter introduces a high memory cost. We will show that it is closely related to the linear auto-encoder method.

2.3 LightGCN Brief

LightGCN [13] is a state-of-the-art GCN-based method in CF. In this paper, LightGCN will be used as the vehicle to elaborate our theory and adopted as the main baseline for performance comparison.

LightGCN leverages the user-item interaction graph to propagate the embedding as follows:

$$\boldsymbol{E}^{(k+1)} = \tilde{\boldsymbol{A}}\boldsymbol{E}^{(k)},$$

where $\boldsymbol{E}^{(0)} \in \mathbb{R}^{(|\mathcal{U}|+|\mathcal{I}|)\times d}$ is the learnable initial embedding matrix of users and items. For a K-layer LightGCN, the final embeddings can be computed as follows:

$$\begin{aligned} \boldsymbol{E} &= \alpha_0\boldsymbol{E}^{(0)} + \alpha_1\boldsymbol{E}^{(1)} + \cdots + \alpha_K\boldsymbol{E}^{(K)} \\ &= \alpha_0\boldsymbol{E}^{(0)} + \alpha_1\tilde{\boldsymbol{A}}\boldsymbol{E}^{(0)} + \cdots + \alpha_K\tilde{\boldsymbol{A}}^K\boldsymbol{E}^{(0)}. \end{aligned} \tag{8}$$

The model prediction is defined as the inner product of the user's and item's final representation $y_{ui} = \boldsymbol{e}_u^T\boldsymbol{e}_i$, where $\boldsymbol{e}_u$ and $\boldsymbol{e}_i$ are the corresponding rows of $\boldsymbol{E}$.

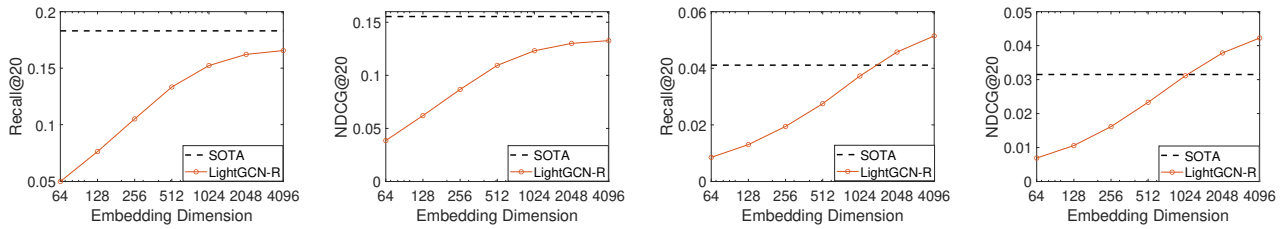
To optimize LightGCN, the Bayesian personalized ranking (BPR) loss [31] is adopted:

$$l_{\text{BPR}} = -\sum_{u=1}^{|\mathcal{U}|} \sum_{i\in\mathcal{N}_u} \sum_{j\notin\mathcal{N}_u} \log\sigma(\boldsymbol{e}_u^T\boldsymbol{e}_i - \boldsymbol{e}_u^T\boldsymbol{e}_j). \tag{9}$$

3 ON THE IMPORTANCE OF SMOOTHNESS AND LOW-PASS FILTERING

In this section, we identify the importance of smoothness and low-pass filters in CF, by using the light convolution in LightGCN as a specific example.

The embeddings play an essential role in CF while smoothness is a key concept in graph signal processing. We observe that there are strong connections between the good embeddings and their smoothness on the graph. We consider the dot product based embedding model. Specifically, let $\boldsymbol{e}_u$ denote the embedding for the u-th user and $\boldsymbol{e}_i$ denote the embedding for the i-th item. The predicted score for the u-th user and i-th item is defined by the dot product



(a) Recall on Gowalla dataset. (b) NDCG on Gowalla dataset. (c) Recall on Amazon-book dataset. (d) NDCG on Amazon-book dataset.

Figure 1: Performance of *untrained* LightGCN versus SOTA, where the SOTA line is LightGCN's performance reported in [13] and LightGCN-R denotes the untrained LightGCN with different embedding dimensions.

$\boldsymbol{e}_u^T \boldsymbol{e}_i$. If $\boldsymbol{R}_{u,i} = 1$, we should promote the similarities between $\boldsymbol{e}_u$ and $\boldsymbol{e}_i$. By the definition of smoothness of graph signals (3), if $\boldsymbol{e}_u$ and $\boldsymbol{e}_i$ are similar for connected user-item pairs, the embeddings are smooth signals on the graph. Consequently, optimizing loss functions, e.g., BPR loss (9), and enhancing the smoothness of the embeddings share the same goal: promoting the similarity between $\boldsymbol{e}_u$ and $\boldsymbol{e}_i$ for $\boldsymbol{R}_{u,i} = 1$.

The above discussion provides a qualitative intuition for the role of smoothness in embeddings. We will now analyze the linear filter in LightGCN to obtain quantitative results. The LightGCN consists of two components: the initial embedding and a linear filter. If the *untrained* LightGCN achieves good performance, it must be the linear filter playing the essential role as the initial embedding is random. The next proposition shows that untrained LightGCN will have a low BPR loss under certain conditions.

Theorem 3.1. *Denote* $N_{\max} = \max_i N_i$ *and* $N_{\min} = \min_i N_i$ *where* $i \in \mathcal{U} \cup \mathcal{I}$. *If* $\boldsymbol{E}^{(0)} \in \mathbb{R}^{(|\mathcal{I}|+|\mathcal{U}|)\times d}$ *follows an i.i.d. uniform distribution over the unit sphere with*

$$d > \frac{C N_{\max}^3 \log(|\mathcal{I}| + |\mathcal{U}|)}{N_{\min}}, \tag{10}$$

then for a one-layer untrained LightGCN, we have

$$\mathbb{P}\left(\left\{\boldsymbol{e}_u^{(1)T}\boldsymbol{e}_i^{(0)} > \boldsymbol{e}_u^{(1)T}\boldsymbol{e}_j^{(0)} | (u,i) \in \mathcal{S}_1, (u,j) \in \mathcal{S}_2\right\}\right) \geq 3/4, \tag{11}$$

where C *is an absolute constant,* $\mathcal{S}_1 = \{(u,i)|\boldsymbol{R}_{u,i} = 1\}$, $\mathcal{S}_2 = \{(u,j)|\boldsymbol{R}_{u,j} = 0\}$.

Remark. (Interpretations of Theorem 3.1) Equation (11) implies a low BPR loss as the predicted score of any positive pair is larger than that of any negative pair. Due to the smoothing effect of light convolution (linear filters), the final embeddings between interacted pairs are similar even if the initial embeddings are random. In Equation (10), $N_{\max}$ and $N_{\min}$ are adopted for a worst-case analysis, and in practice, we can replace them with the average degree. Equation (10) shows that the required embedding dimension of untrained LightGCN grows with the dataset density, which implies untrained LightGCN is more effective on sparse datasets. For the Gaussian initialization adopted in [13], the results are similar as high dimensional Gaussian random vectors concentrate around a sphere (refer to Section 3.1 in [38]). The probability 3/4 can be improved to any probability approaches arbitrarily close to 1. The high-level reason for untrained LightGCN performing well is that the information contained in the rating matrix and the graph are identical. The BPR loss is adopted for exploiting the information in rating matrix while the low-pass filters are to exploit information in the graph. Thus, a proper use of low-pass filters can accelerate the training or even avoid the training. Interestingly, some recent works also reveal that infinitely wide random CNNs achieve better performance than trained ones [2].

Based on Theorem 3.1, we argue that the performance of *untrained* LightGCN improves with the embedding dimension and it should be competitive to a trained one when the embedding dimension is sufficiently large.

To verify this argument, we follow the experiment settings in [13] and conduct the experiments for a 3-layer *untrained* LightGCN. The initial embeddings $\boldsymbol{E}^{(0)}$ is initialized following an i.i.d. Gaussian distribution $\mathcal{N}(0, 0.1)$ as in the original paper. Once the model is initialized, we do not train it but simply compute the user/item embeddings using Equation (8) and then directly test it on the test dataset. We use two sparse datasets, i.e., Gowalla and Amazon-book. The test performance versus the embedding dimension is shown in Fig. 1. As the training/test splitting of two datasets is identical to [13], we regard the LightGCN's performance reported in [13] as the state-of-the-art. We will also compare to LightGCN with large embedding dimensions in Table 4. The experiments agree with our theory well. As the linear filter is to promote the smoothness, it demonstrates the crucial role of smoothness in CF.

However, the untrained LightGCN is not a practical algorithm for recommendation as the large embedding dimension leads to an expensive memory cost and inference time. Fortunately, the untrained LightGCN with infinitely dimensional embedding has a closed-form solution for predicted scores, as shown in the next theorem.

Theorem 3.2. *Consider an untrained LightGCN with* $\boldsymbol{E}^{(0)} \in \mathbb{R}^{(|\mathcal{I}|+|\mathcal{U}|)\times d}$ *following an i.i.d. distribution with zero mean and non-zero variance. As* $d \to \infty$, *the predicted score of the untrained LightGCN follows*

$$\boldsymbol{s}_u = \sum_{k=0}^{K-1} \beta_k \tilde{\boldsymbol{r}}_u (\tilde{\boldsymbol{R}}^T \tilde{\boldsymbol{R}})^k. \tag{12}$$

where β_k *are constants depending on* $[\alpha_k]_{k=0,\cdots,K}$ *in* (8).

Remark. (Interpretations of $\tilde{\boldsymbol{R}}^T\tilde{\boldsymbol{R}}$) As shown in Theorem 3.2, the gram matrix $\tilde{\boldsymbol{R}}^T\tilde{\boldsymbol{R}}$ plays a pivotal role. For sparse binary data,

Table 1: Classic methods versus their corresponding graph filters and spatial GCNs.

Method	Low-rank Factorization [6]	Linear Auto-encoder [36]	Neighborhood-based [1]	LGCN-IDE (12)
Input Signal $\bar{\boldsymbol{r}}_u$	$\boldsymbol{D}_I^{-\frac{1}{2}}\boldsymbol{r}_u$	$\tilde{\boldsymbol{r}}_u$	$\boldsymbol{r}_u$	$\tilde{\boldsymbol{r}}_u$
Graph Filter	$h(\lambda_i) = \mathbf{1}_{i\le d}$	$h(\lambda_i) = \frac{1-\lambda_i}{1+\mu-\lambda_i}$	$h(\lambda_i) = 1-\lambda_i$	$h(\lambda_i) = \sum_{k=0}^{K-1}\beta_k(1-\lambda_i)^k$
Corresponding Spatial GCN	Infinite-layer spatial GCN with convolutional normalization (16) (17)	Infinite-layer spatial GCN with layer combination (21)	Single-layer spatial GCN	Multi-layer spatial GCN with layer combination

$(\boldsymbol{D}_I^{-\frac{1}{2}}\boldsymbol{R}^T\boldsymbol{R}\boldsymbol{D}_I^{-\frac{1}{2}})_{ij}$ defines the cosine similarity between item i and item j [1]. Likewise, $(\tilde{\boldsymbol{R}}^T\tilde{\boldsymbol{R}})_{ij}$ provides a similarity measure between item i and item j. Directly using the gram matrix as the item-item similarity results in the neighborhood-based method [1], which was the winner of Millions of Song Competition[2]. The similarity between the neighborhood-based method and LightGCN is not surprising as LightGCN is based on the neighborhood propagation. As LightGCN consists of multi-hop propagation, the term $\tilde{\boldsymbol{R}}^T\tilde{\boldsymbol{R}}$ appears as polynomials. From the graph signal processing perspective, it is a linear filter, which is low-pass.

We call (12) as LightGCN with Infinitely Dimensional Embedding (LGCN-IDE). The performance of LGCN-IDE is shown in Table 3. Remarkably, we see that on Amazon-book dataset, it outperforms the performance of LightGCN reported in [13] by more than 40% under exactly the same training/test data splits.

4 A UNIFIED FRAMEWORK

In this section, we first extend LGCN-IDE to incorporate general low-pass filters, which form a unified framework. Then we prove that this framework unifies the neighborhood-based approaches, low-rank matrix factorization, linear auto-encoders, and linear graph convolutional networks, where different methods correspond to different low pass-filters. Finally, we present a simple yet effective algorithm for CF.

4.1 A Unified Graph Low-pass Filter Based Framework

In this subsection, we extend (12) to incorporate general graph filters. To simplify the notations, we denote $\tilde{\boldsymbol{P}} = \tilde{\boldsymbol{R}}^T\tilde{\boldsymbol{R}}$ in the remaining of the article. Note that $\tilde{\boldsymbol{P}}$ can also be seen as a normalized adjacency matrix for an item-to-item graph, whose eigenvalues are between 0 and 1.

Theorem 4.1. *Let $\lambda_1 \ge \cdots \ge \lambda_{|\mathcal{I}|}$ be the eigenvalues of $\tilde{\boldsymbol{P}}$, then*

$$0 \le \lambda_{|\mathcal{I}|} \le \cdots \le \lambda_1 \le 1.$$

The graph Laplacian of the item-to-item graph is defined as $\tilde{\boldsymbol{L}} = \boldsymbol{I} - \tilde{\boldsymbol{P}}$. In this way, we can apply graph signal processing to the item-to-item graph. Next we elaborate our unified framework, which is an extension of (12) with general graph filters. We consider an input graph signal $\bar{\boldsymbol{r}}_u$, which is some transformation of the users' observed ratings $\boldsymbol{r}_u$. Then a low-pass filter is applied to the graph signal to obtain a filtered signal. Finally, we may scale the obtained graph signal to get the final prediction scores. Denoting the eigendecomposition by $\tilde{\boldsymbol{L}} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^T$, the framework is given by

$$\bar{\boldsymbol{s}}_u = \bar{\boldsymbol{r}}_u\boldsymbol{U}\mathrm{Diag}(h(\lambda_1),\cdots,h(\lambda_n))\boldsymbol{U}^T, \tag{13}$$

where $\bar{\boldsymbol{s}}_u$ is the filtered predicted score, and $h(\cdot)$ is a low-pass filter. From the graph signal processing perspective, it is a graph convolution, i.e., a graph signal $\boldsymbol{r}_u \in \mathbb{R}^{|\mathcal{I}|}$ convolving with a low-pass filter $h(\cdot)$.

4.2 Interpreting Classic Methods from Graph Signal Processing Perspective

Interestingly, some classic works for recommendation can be interpreted as graph signal processing approaches, where the low-pass filter plays an essential role. The classic methods typically involves auto-encoder-based [24, 27, 36], matrix factorization-based [6, 31], and GCN-based ones [13, 41, 51]. In this subsection, we will provide a unified view of the linear methods from the graph signal processing perspective. As the spectral convolution can be transformed into a spatial convolution in GCNs by first-order approximation [19], it is interesting to investigate what kind of GCNs will these classic methods induce. These GCNs induced by classic algorithms can also be seen as white-box neural networks [5]. A test of performance for these GCNs is left for future works.

4.2.1 Low-rank Matrix Factorization. Low-rank matrix factorization is one of the most classic algorithms for CF. Note that GFT is also a matrix factorization where the low-frequency signal components correspond to the principle components of the rating matrix. This observation allows us to connect MF and graph-based methods. We take the objective function in a recent work [6] as an example. Denote d as the embedding dimension, the model is given by

$$\boldsymbol{U}^*, \boldsymbol{V}^* = \underset{\boldsymbol{U}\in\mathbb{R}^{|\mathcal{U}|\times d}, \boldsymbol{V}\in\mathbb{R}^{|\mathcal{I}|\times d}}{\operatorname{argmin}} \|\tilde{\boldsymbol{R}} - \boldsymbol{U}\boldsymbol{V}^T\|_F^2 \quad \text{s.t. } \boldsymbol{V}^T\boldsymbol{V} = \boldsymbol{I}. \tag{14}$$

As shown in [6], $\boldsymbol{V}^*$ contains the smallest K eigenvectors of $\tilde{\boldsymbol{L}}$ and $\boldsymbol{U}^* = \boldsymbol{R}\boldsymbol{D}_I^{\frac{1}{2}}\boldsymbol{V}$. Viewing the eigendecomposition as GFT, it can be interpreted as an ideal low-pass filter (6)

$$h(\lambda_i) = \mathbf{1}_{i\le d}.$$

We then turn low-rank matrix factorization into a spatial convolution fashion. This is more difficult than the conversion in GCN [19] due to the orthogonal constraint and non-convexity of problem (14). Observing that the optimal solution $\boldsymbol{V}^*$ to (14) is also the optimal solution to the following problem

$$\boldsymbol{V}^* = \underset{\boldsymbol{X}\in St(|\mathcal{I}|,d)}{\operatorname{argmax}} \|\tilde{\boldsymbol{R}}\boldsymbol{X}\|_F^2. \tag{15}$$

We can rewrite (15) as spatial convolution by first-order expansion like GCNs [19]. We begin with a random $\boldsymbol{E}^{(0)} \in \mathbb{R}^{|\mathcal{I}|\times d}$, and the

[2]The competition is at https://www.kaggle.com/c/msdchallenge

update rule is given by

$$\hat{\boldsymbol{E}}^{(k)} = \left(\nabla_{\boldsymbol{X}} \|\tilde{\boldsymbol{R}}\boldsymbol{X}\|_F^2\right)\Big|_{\boldsymbol{X}=\boldsymbol{E}^{(k-1)}} = \tilde{\boldsymbol{P}}\boldsymbol{E}^{(k-1)}, \tag{16}$$

$$\boldsymbol{E}^{(i)} = \underset{S \in St(|\mathcal{I}|,d)}{\operatorname{argmax}} \langle \boldsymbol{S}, \hat{\boldsymbol{E}}^{(k)} \rangle \overset{(a)}{=} \hat{\boldsymbol{E}}^{(k)} \left(\hat{\boldsymbol{E}}^{(k)T} \hat{\boldsymbol{E}}^{(k)}\right)^{-\frac{1}{2}}, \tag{17}$$

where (a) follows Proposition 7 in [18]. The final embeddings are given by

$$\boldsymbol{V}^* = \boldsymbol{E} = \boldsymbol{E}^{(\infty)}.$$

Note that (16) is a spatial graph convolution, and (17) is coincidentally equivalent to convolutional normalization for CNNs [26] (refer to (6)-(8) in [26]). The convolutional normalization was proposed to accelerate the training of convolutional networks and improve robustness. From this view, the low-rank matrix factorization is equivalent to an infinite layer GCN with convolutional normalization. As the number of layers is large, it suffers from the over-smoothing issue [23], which hurts the performance.

4.2.2 Linear Auto-encoders. In the linear auto-encoders, e.g., EASE [34] and SLIM [27], the predicted score vector of a user ($\bar{\boldsymbol{s}}_u$) is obtained by the dot product

$$\bar{\boldsymbol{s}}_u = \bar{\boldsymbol{r}}_u \boldsymbol{B},$$

where $\boldsymbol{B} \in \mathbb{R}^{|\mathcal{I}|\times|\mathcal{I}|}$ is a learnable weight matrix. The training objective is some regularized or constrained version of $\min_{\boldsymbol{B}} \sum_u \|\bar{\boldsymbol{r}}_u - \bar{\boldsymbol{r}}_u\boldsymbol{B}\|_2^2$. From the graph signal processing view, it can be interpreted as a graph signal $\bar{\boldsymbol{r}}_u$ convolving with a filter $\boldsymbol{B}$, and $\bar{\boldsymbol{r}}_u = \bar{\boldsymbol{r}}_u\boldsymbol{B}$ is a steady state. This defines a graph diffusion on the corresponding graph like opinion dynamics (7). Next, we show the equivalence between a specific version of linear auto-encoders and the graph diffusion filter.

As shown in [36], the following linear auto-encoder is able to achieve competitive performance compared with the deep ones [24, 43]. Specifically, we consider the following formulation in [36] for simplicity:

$$\underset{\boldsymbol{B}}{\text{minimize}} \quad \|\tilde{\boldsymbol{R}} - \tilde{\boldsymbol{R}}\boldsymbol{B}\|_F^2 + \mu\|\boldsymbol{B}\|_F^2. \tag{18}$$

As (18) is a ridge regression, we can write down the closed-form solution as

$$\boldsymbol{B}^* = (\tilde{\boldsymbol{P}} + \mu\boldsymbol{I})^{-1}\tilde{\boldsymbol{P}}. \tag{19}$$

Viewing the eigenvalue decomposition as GFT, the graph filter in (19) is given by

$$h(\lambda_i) = \frac{1-\lambda_i}{1+\mu-\lambda_i}. \tag{20}$$

To understand (20) in the content of low-pass filters, the low-pass ratio η_k in (4) is given by

$$\eta_k = \frac{\frac{1-\lambda_{k+1}}{1+\mu-\lambda_{k+1}}}{\frac{1-\lambda_k}{1+\mu-\lambda_k}} = \frac{(1-\lambda_{k+1})(1+\mu-\lambda_k)}{(1-\lambda_k)(1+\mu-\lambda_{k+1})} = 1 - \frac{\mu(\lambda_{k+1}-\lambda_k)}{(1-\lambda_k)(1+\mu-\lambda_{k+1})} < 1.$$

The convolutional filter in (20) is similar to opinion dynamics and is a kind of graph diffusion filter. Like other diffusion-based methods, the memory cost of linear auto-encoder is high as we need to store matrix $\boldsymbol{B}$ in (18).

In the literature, the Neumann series are often adopted to convert the graph diffusion into a spatial convolution [20, 44]. Similarly, we can use it to interpret (19) as spatial GCNs. For $\mu > 1$, (19) can be written as

$$(\tilde{\boldsymbol{P}} + \mu\boldsymbol{I})^{-1}\tilde{\boldsymbol{P}} = \frac{1}{\mu}\left(\sum_{k=0}^{\infty}(-\mu^{-1}\tilde{\boldsymbol{P}})^k\right)\tilde{\boldsymbol{P}}. \tag{21}$$

From this view, the initial embedding is an identity matrix $\boldsymbol{E}^{(0)} = \boldsymbol{I} \in \mathbb{R}^{|\mathcal{I}|\times|\mathcal{I}|}$, the update of corresponding spatial convolution is given by

$$\boldsymbol{E}^{(k)} = \tilde{\boldsymbol{P}}\boldsymbol{E}^{(k-1)},$$

and the final embeddings can be obtained as

$$\boldsymbol{E} = \sum_{k=1}^{\infty} -\left(-\frac{1}{\mu}\right)^k \boldsymbol{E}^{(k)}.$$

The layer combination appears naturally and the coefficients decrease quickly. As discussed in [13], the layer combination is the key to alleviate the over-smoothing issue and improve performance.

4.2.3 Neighborhood-based Approaches. The neighborhood-based approaches are often considered as exploiting first-order graph information in the literature discussions [13]. We consider the following formulation, which utilizes the gram matrix as the similarity matrix [1], i.e., $\bar{\boldsymbol{s}}_u = \boldsymbol{r}_u\tilde{\boldsymbol{P}}$. Obviously, the corresponding filter is a first-order linear filter

$$h(\lambda_i) = 1 - \lambda_i.$$

and the corresponding spatial GCN is a one-layer GCN. This approach is simple and scalable. However, it lacks higher-order information on the graph.

4.2.4 LGCN-IDE. For completeness, we analyze LGCN-IDE (12). By eigendecomposition, the corresponding filter takes the form of

$$h(\lambda_i) = \sum_{k=0}^{K-1} \beta_k (1-\lambda_i)^k.$$

Since it is still a LightGCN, LGCN-IDE naturally corresponds to multi-layer spatial GCN with a layer combination.

4.3 A Simple yet Effective Baseline Algorithm

In this subsection, we develop a simple yet effective baseline algorithm, whose training is as efficient as the inference of LightGCN with a big-O notation. We first analyze the inference computational complexity of LightGCN. We denote the number of non-zero elements in $\boldsymbol{R}$ as η. For a LightGCN with d-dimensional embedding, the inference time is $O(\eta d)$.

The general graph filters require eigendecomposition and thus are not efficient for large-scale recommendation [13]. Fortunately, there are some graph filters that enjoy a high computational efficiency, i.e., linear filters and ideal low-pass filters. In order to obtain linear filters, only the normalization is required during training, and thus the training complexity is $O(\eta)$. A major drawback of the linear filters is that they can hardly obtain a high-order information of the graph.

Table 2: Statistics of the experimented data.

Dataset	# User	# Item	# Interaction	Density
Gowalla	29, 858	40, 981	1, 027, 370	0.00084
Yelp2018	31, 668	38, 048	1, 561, 406	0.00130
Amazon-book	52, 643	91, 599	2, 984, 108	0.00062

For the ideal low-pass filter, only the top-K eigenvectors of $\tilde{\boldsymbol{P}}$ are required. Nevertheless, a direct computation for the top-K eigenvector of $\tilde{\boldsymbol{P}}$ is far from computation and memory efficient because $\tilde{\boldsymbol{P}}$ is not as sparse as $\boldsymbol{R}$. By using the equivalent formulation in (15), the largest eigenvectors can be computed by (16), (17), and this iterative algorithm is called the generalized power method (GPM) in the optimization literature [18]. In GPM, we only need to store $\boldsymbol{R}$ instead of $\tilde{\boldsymbol{P}}$, and the computational complexity is $O((d\eta + d^3)\log(1/\epsilon))$ where ϵ is the desired accuracy for the eigenvectors. This algorithm is efficient as long as $d^2 < \eta$. As discussed before, the ideal low-pass filter is equivalent to an infinite layer GCN without layer combination and it suffers from over-smoothing, which means that it lacks a low-order information in the graph.

As a result, we argue that combining the linear filter and ideal low-pass filter will result in a strong baseline. Specifically, our proposed algorithm, named as Graph Filter based Collaborative Filtering (GF-CF), has the following form

$$\boldsymbol{s}_u = \boldsymbol{r}_u \left(\tilde{\boldsymbol{R}}^T \tilde{\boldsymbol{R}} + \alpha \boldsymbol{D}_I^{-\frac{1}{2}} \bar{\boldsymbol{U}} \bar{\boldsymbol{U}}^T \boldsymbol{D}_I^{\frac{1}{2}} \right), \tag{22}$$

where $\boldsymbol{s}_u$ and $\boldsymbol{r}_u$ denote predicted and observed scores, respectively. Likewise, $\bar{\boldsymbol{U}}$ is the top-K singular vectors of $\tilde{\boldsymbol{R}}$, and α is the tuned parameter. We acknowledge that learning α or transforming (22) into GCNs may lead to better performance. Nevertheless, we will demonstrate that (22) already achieves the state-of-the-art performance.

5 EXPERIMENTS

In this section, we first describe the experimental settings, which exactly follow [13]. Next, we compare our method with the state-of-the-art deep learning methods.

5.1 Experimental Settings

To keep the comparison fair, we use the same datasets, the same train/test splitting, and the identical evaluation metric as in [13]. The statistics of the datasets are listed in Table 2. The evaluation metrics are recall@20 and ndcg@20.

5.1.1 Benchmarks. We follow [13] to set up the benchmarks.

(1) LightGCN [13]: LightGCN is the state-of-the-art method for CF. Please refer to Section 2.3 for a detailed description.
(2) NGCF [41]: NGCF is a nonlinear deep GCN-based method. Besides the components in LightGCN, it contains of feature transformation, and nonlinear activation.
(3) GRMF and GRMF-norm[13, 30]: GRMF adds a graph Laplacian regularizer to the training objective of BPR loss in matrix factorization. In GRMF-norm, the normalized Laplacian is adopted instead of the graph Laplacian.

Table 3: The comparison of overall performance among GF-CF and competing methods. The performance of benchmarks is reproduced from [13].

Dataset	Gowalla		Yelp2018		Amazon-book	
Method	recall	ndcg	recall	ndcg	recall	ndcg
NGCF	0.1570	0.1327	0.0579	0.0477	0.0344	0.0263
Mult-VAE	0.1641	0.1335	0.0584	0.0450	0.0407	0.0315
GRMF	0.1477	0.1205	0.0571	0.0462	0.0354	0.0270
GRMF-norm	0.1557	0.1261	0.0561	0.0454	0.0352	0.0269
LightGCN	0.1830	**0.1554**	0.0649	0.0530	0.0411	0.0315
LGCN-IDE	0.1682	0.1347	0.0609	0.0505	0.0612	0.0514
GF-CF	**0.1849**	0.1518	**0.0697**	**0.0571**	**0.0710**	**0.0584**

(4) Mult-VAE [24]: This is a variational autoencoder based method. The data is assumed to be generated by a multinomial distribution and variational inference is adopted to estimate the parameters.

In [41], it has been shown that NGCF outperforms GC-MC [4], Pinsage [49], NeuMF [15], CMN [10], MF [31], HOP-Rec [48] on the same train/test splitting. Thus, we will not include these methods as benchmarks. We also do not compare with full rank models [27, 35] due to the out of memory on Amazon-book dataset. The hyperparameter settings are identical to [13].

For the proposed graph filter based methods, we focus on the following two variants:

(1) GF-CF: The proposed simple baseline method for CF in (22).
(2) LGCN-IDE: The untrained LightGCN with infinitely dimensional embedding. The closed-form is given in (12).

For the implementation of graph filters, we adopt Scipy [40] for sparse operation.

5.2 Performance Comparison

The performance of the proposed methods and other benchmarks are shown in Table 3. Despite the simplicity, GF-CF achieves competitive or better performance than deep learning-based methods.

5.2.1 LGCN-IDE versus LightGCN. LGCN-IDE is an untrained LightGCN with an infinitely dimensional embedding. On Gowalla and Yelp2018, which are of small sizes, LightGCN outperforms LGCN-IDE. However, LGCN-IDE outperforms LightGCN by a large margin on the large-scale dataset, i.e., the Amazon-book dataset. In LightGCN, the known scores are compressed into limited dimensional vectors, which restricts the expressiveness. In contrast, in LGCN-IDE, the ratings are directly used as the graph signal without compression. Additionally, LightGCN is trained with a stochastic gradient descent (SGD) while LGCN-IDE has a closed-form solution. As the size of the dataset increases, the optimization by SGD becomes more difficult. We suspect that these two reasons contribute to the large performance gain of LGCN-IDE over LightGCN in the Amazon-book dataset.

5.2.2 Graph filters versus deep learning-based methods. In Table 3, the simple graph filter achieves competitive or better performance compared with deep learning-based methods. LightGCN also outperforms NGCF by removing the non-linear transformations. From the universal approximation theory [16], deep neural networks can

Table 4: The comparison of performance and training time of GF-CF and LightGCN.

Dataset	Gowalla			Yelp2018			Amazon-book		
Method	recall	ndcg	training time	recall	ndcg	training time	recall	ndcg	training time
LightGCN-64	0.1830	0.1554	2.77×10^4s	0.0649	0.0530	5.15×10^4s	0.0411	0.0315	1.27×10^5s
LightGCN-128	0.1878	0.1591	3.31×10^4s	0.0671	0.0550	5.66×10^4s	0.0459	0.0353	1.81×10^5s
LightGCN-256	0.1893	0.1606	4.54×10^4s	0.0689	0.0568	8.09×10^4s	0.0481	0.0371	2.98×10^5s
LightGCN-512	0.1892	0.1604	7.28×10^4s	0.0689	0.0569	1.33×10^5s	0.0485	0.0375	5.26×10^5s
GF-CF	0.1849	0.1518	30.5s	0.0697	0.0571	46.0s	0.0710	0.0584	65.8s

approximate linear functions easily. Nevertheless, linear functions are non-trivial to learn for a neural network trained with SGD. A recent theoretical study demonstrates that it is impossible for neural networks with tanh, cosine, or quadratic activation to extrapolate the linear functions well [46]. With ReLU activation, A neural network can extrapolate linear functions well if the training data cover all directions (e.g., a hypercube covering the origin) [46], which is not trivial to satisfy in practice. This theoretical result suggests that learning linear functions is a non-trivial task. In addition, deep neural networks do well in extracting complicated features, but CF with implicit feedback is in lack of rich features. Owing to these two factors, the linear models are able to outperform deep models in CF with implicit feedback.

5.3 Comparison with LightGCN of Large Embedding Dimension

In this subsection, we compare GF-CF with LightGCNs of different embedding dimensions. For the untrained LightGCN, the performance improves significantly with the dimension as shown in Fig. 1. The natural questions are 1) does the performance of trained LightGCN increase significantly as the dimension grows; 2) how does GF-CF perform compared with LightGCN with large embedding dimensions. We validate these questions empirically in Table 4. The experiments in this subsection are conducted on a server with an Intel Xeon(R) CPU E5-2698 v4 @ 2.20GHz and a Tesla V100 GPU. For the implementation of LightGCN, we download the source code from https://github.com/gusye1234/LightGCN-PyTorch and train 1000 epochs as the original paper[3]. Due to the excessive training cost, we do not train LightGCN with an embedding dimension of more than 512. As shown in Table 4, GF-CF still achieves competitive or higher performance than LightGCN with large embedding dimensions. As the embedding dimension grows, the performance improvement of LightGCN becomes marginal, which is similar to matrix factorization and neural collaborative filtering [32]. The overall training time of GF-CF is even smaller than 1 training epoch consumed by LightGCN. It demonstrates that GF-CF is a simple but hard-to-beat baseline method for CF.

6 RELATED WORKS

6.1 Collaborative Filtering Methods

Collaborative filtering (CF) plays a fundamental role in modern recommender systems [7]. One popular paradigm is the model-based CF methods. In such methods, the users and items are parameterized by (low-dimensional) vectors and the interactions are reconstructed based on the embeddings and model weights. The classic matrix factorization (MF) maps the ID of users and items as embedding vectors and uses the dot product between embedding vectors as predicted scores. The dot product model can be further improved by using neural networks [15, 37]. Another classic model-based CF is to reconstruct the score for an item by a transformation of the scores for other items, from linear auto-encoders (e.g., SLIM [27]) to deep auto-encoders (e.g., Multi-VAE [24]). Another paradigm is graph-based CF methods. The early works (e.g., Item-rank [12] and Bi-rank [14]) exploit the label propagation on graph and belong to the neighborhood-based methods. These methods are often considered as heuristics and inferior to model-based methods due to the lack of training. Recent works address this issue by developing GCN-based methods and train GCNs in an end-to-end manner, e.g., GC-MC [4], NGCF [41], and LightGCN [13].

Notice that the information contained in the sparse rating matrix or graph formulation are identical and GFT is a matrix factorization. In this paper, we unify the two paradigms from the graph signal processing view and identify that the low-pass filters are the underlying key component in the two paradigms. In addition, we show that different paradigms correspond to different low-pass filters and these filters can be incorporated together to improve the performance.

6.2 Spectral and Spatial GCNs

The spectral GCNs are developed from graph signal processing with learned graph filters, which enjoy theoretical guarantees from graph signal processing theory [33]. Nevertheless, GFT requires full eigendecomposition, which induces prohibitive computation for large-scale graphs. The spectral CF [52] and LCF [50] belong to this category and thus they cannot be applied on large-scale datasets. To speed up the computation, the spatial GCNs based on 1-hop neighbor propagation were proposed [45]. In each layer of spatial GCNs, only neighborhood aggregations are required, and thus the computational cost is extensively reduced. In the context of recommendation, spatial GCNs contain GCMC [4], NGCF [41], LightGCN [13], and PinSage [49]. A unique advantage of these methods is the scalability, meaning that they can be applied to large-scale sparse datasets. A recent theoretical study unified the spectral and spatial GCNs and demonstrates that they are all low-pass filters [3]. In the paper, we also unify the classic CF methods via low-pass filtering, which explains the success of GCNs in CF.

[3]We notice that training LightGCN for $400-600$ instead of 1000 epochs only introduce a slight performance loss, which reduces the training time of LightGCN, but this does not affect our conclusion as we have more than three magnitudes of speedups.

7 CONCLUSIONS

In this paper, we identified the importance of smoothness in the embeddings in a successful recommendation both theoretically and empirically, which bridges CF and graph signal processing theory. Via the lens of graph signal processing, we showed that the neighborhood-based methods, low-rank matrix completion, and linear auto-encoders are all graph convolution with low-pass filters. This further validated the power of graph convolution for recommendation. In addition to our theoretical analysis, we also developed a simple but hard-to-beat baseline algorithm, GF-CF. It was demonstrated that GF-CF achieves competitive or better performance than deep learning-based methods. We believe that the insights of this investigation are inspirational to the principled GCN architecture design for recommender systems. In the future, we will implement the GCNs induced by classic algorithms in Table 1 and exploit additional information, e.g., social networks and knowledge graphs.

8 PROOFS

8.1 Proof of Theorem 3.1

Proof. We first prove that (11) holds when the mutual coherence [9]) of the embeddings satisfies

$$\epsilon = M_{\boldsymbol{E}^{(0)}} < \sqrt{\frac{N_{\min}}{2N_{\max}^3}}, \tag{23}$$

and then show that as $d > \frac{CN_{\max}^3 \log(|\mathcal{I}|+|\mathcal{U}|)}{N_{\min}}$, (23) holds with probability at least 3/4.

$$\begin{aligned}
&\boldsymbol{e}_i^{(1)T}\boldsymbol{e}_j^{(0)} - \boldsymbol{e}_i^{(1)T}\boldsymbol{e}_k^{(0)} \\
=&\left(\frac{1}{\sqrt{N_i}}\sum_{l\in\mathcal{N}_i}\frac{1}{\sqrt{N_l}}\boldsymbol{e}_l^{(0)}\right)^T(\boldsymbol{e}_j^{(0)} - \boldsymbol{e}_k^{(0)}) \\
\geq&\frac{1}{\sqrt{N_i}}\left(\frac{1}{\sqrt{N_j}} - \frac{N_i-1}{\sqrt{N_{\min}}}\epsilon - \frac{N_i}{\sqrt{N_{\min}}}\epsilon\right) \\
\geq&\frac{1}{\sqrt{N_i}}\left(\frac{1}{\sqrt{N_{\max}}} - \frac{2N_{\max}}{\sqrt{N_{\min}}}\epsilon\right) \overset{(a)}{>} 0
\end{aligned}$$

where (a) follows the assumption that $\epsilon < \sqrt{\frac{N_{\min}}{2N_{\max}^3}}$. With Lemma 8.1, we see that $\epsilon < \sqrt{\frac{N_{\min}}{2N_{\max}^3}}$ as $d > \frac{CN_{\max}^3 \log(|\mathcal{I}|+|\mathcal{U}|)}{N_{\min}}$. □

Lemma 8.1. *(Theorem 3.5 in [42]) Let $\boldsymbol{A} \in \mathbb{R}^{n\times m}$ with rows i.i.d. chosen from the uniform distribution on the sphere. Then with probability at least 3/4,*

$$M_{\boldsymbol{A}} \leq C\sqrt{\frac{\log n}{m}}.$$

where C is an absolute constant.

8.2 Proof of Theorem 3.2

Proof. We first separate the embeddings $\boldsymbol{E}^{(k)}$ into user embeddings $\boldsymbol{U}^{(k)}$ and item embeddings $\boldsymbol{V}^{(k)}$, and the individual update is given by

$$\boldsymbol{U}^{(k+1)} = \tilde{\boldsymbol{R}}\boldsymbol{V}^{(k)}, \quad \boldsymbol{V}^{(k+1)} = \tilde{\boldsymbol{R}}^T\boldsymbol{U}^{(k)}.$$

The final embeddings are

$$\begin{aligned}
\boldsymbol{V} &= \left(\alpha_0\boldsymbol{V}^{(0)} + \alpha_1\tilde{\boldsymbol{R}}^T\boldsymbol{U}^{(0)} + \alpha_2\tilde{\boldsymbol{R}}^T\tilde{\boldsymbol{R}}\boldsymbol{V}^{(0)} + \alpha_3\tilde{\boldsymbol{R}}^T\tilde{\boldsymbol{R}}\tilde{\boldsymbol{R}}^T\boldsymbol{U}^{(0)} + \cdots\right) \\
&= \left(\sum_{i=0}^{2i\leq K}\alpha_{2i}(\tilde{\boldsymbol{R}}^T\tilde{\boldsymbol{R}})^i\boldsymbol{V}^{(0)} + \sum_{i=0}^{2i+1\leq K}\alpha_{2i+1}(\tilde{\boldsymbol{R}}^T\tilde{\boldsymbol{R}})^i\tilde{\boldsymbol{R}}^T\boldsymbol{U}^{(0)}\right)
\end{aligned}$$

and $\boldsymbol{U}$ can be computed similarly.

The final prediction of untrained LightGCN with infinitely dimensional embedding is given by

$$\begin{aligned}
\boldsymbol{S} = \boldsymbol{U}\boldsymbol{V}^T =& \left(\sum_{i=0}^{2i\leq K}\alpha_{2i}(\tilde{\boldsymbol{R}}^T\tilde{\boldsymbol{R}})^i\boldsymbol{U}^{(0)} + \sum_{i=0}^{2i+1\leq K}\alpha_{2i+1}(\tilde{\boldsymbol{R}}\tilde{\boldsymbol{R}}^T)^i\tilde{\boldsymbol{R}}\boldsymbol{V}^{(0)}\right) \\
&\cdot\left(\sum_{i=0}^{2i\leq K}\alpha_{2i}(\tilde{\boldsymbol{R}}^T\tilde{\boldsymbol{R}})^i\boldsymbol{V}^{(0)} + \sum_{i=0}^{2i+1\leq K}\alpha_{2i+1}(\tilde{\boldsymbol{R}}^T\tilde{\boldsymbol{R}})^i\tilde{\boldsymbol{R}}^T\boldsymbol{U}^{(0)}\right)^T
\end{aligned}$$

For a pair of matrices $\boldsymbol{X}, \boldsymbol{Y}$, if the rows of $\boldsymbol{X} \in \mathbb{R}^{*\times d}, \boldsymbol{Y}^{(0)} \in \mathbb{R}^{*\times d}$ follow independently identical distribution, due to the linearity of dot product, we have $\lim_{d\to\infty}\boldsymbol{X}\boldsymbol{Y}^T = \mathbb{E}[\boldsymbol{x}_1\boldsymbol{y}_1^T]$, where $\boldsymbol{x}_1$ (resp. $\boldsymbol{y}_1$) denotes the first column of $\boldsymbol{X}$ (resp. $\boldsymbol{Y}$).

Thus, as $d \to \infty$, we have

$$\lim_{d\to\infty}\boldsymbol{S} = \mathbb{E}_{\boldsymbol{U}^{(0)},\boldsymbol{V}^{(0)}}[\boldsymbol{S}] = \sum_{k=0}^{K-1}\beta_k\tilde{\boldsymbol{R}}(\tilde{\boldsymbol{R}}^T\tilde{\boldsymbol{R}})^k.$$

where β_k depends on $[\alpha_k]_{i=0,\cdots,K}$.

For a given user u, the estimated scores is shown as $\boldsymbol{s}_u = \tilde{\boldsymbol{r}}_u\sum_{k=0}^{K-1}\beta_k(\tilde{\boldsymbol{R}}^T\tilde{\boldsymbol{R}})^k$.

□

8.3 Proof of Theorem 4.1

Proof. We observe that $\tilde{\boldsymbol{A}}^2 = \begin{bmatrix}\tilde{\boldsymbol{R}}\tilde{\boldsymbol{R}}^T & \boldsymbol{0}\\ \boldsymbol{0} & \tilde{\boldsymbol{R}}^T\tilde{\boldsymbol{R}}\end{bmatrix}$. As $\tilde{\boldsymbol{A}}^2$ is block diagonal, eigenvalues of $\tilde{\boldsymbol{A}}^2$ are a concatenation of eigenvalues of $\tilde{\boldsymbol{R}}\tilde{\boldsymbol{R}}^T$ and $\tilde{\boldsymbol{R}}^T\tilde{\boldsymbol{R}}$. For the largest eigenvalue, we have

$$\lambda_{\max}(\tilde{\boldsymbol{R}}^T\tilde{\boldsymbol{R}}) = \lambda_{\max}(\tilde{\boldsymbol{R}}\tilde{\boldsymbol{R}}^T) = \lambda_{\max}(\tilde{\boldsymbol{A}}^2) = \lambda_{\max}(\tilde{\boldsymbol{A}})^2 \overset{(a)}{=} 1.$$

where (a) follows Lemma 8.2. As $\tilde{\boldsymbol{R}}^T\tilde{\boldsymbol{R}}$ is positive semi-definite, $\lambda_{\min}(\tilde{\boldsymbol{R}}^T\tilde{\boldsymbol{R}}) \geq 0$. This finishes the proof. □

Lemma 8.2. *Let $\lambda_1 \leq \lambda_2 \leq \cdots\lambda_{|\mathcal{I}|+|\mathcal{U}|}$ be eigenvalues of $\tilde{\boldsymbol{A}}$. Then $-1 \leq \lambda_1 \leq \lambda_2 \leq \cdots\lambda_{|\mathcal{I}|+|\mathcal{U}|} = 1$.*

Proof. First, observing that $\forall\boldsymbol{x} \in \mathbb{S}^{n-1}$, we have

$$\boldsymbol{x}^T(\boldsymbol{I} - \tilde{\boldsymbol{A}})\boldsymbol{x} = \sum_{i,j}A_{i,j}\left(\frac{x(i)}{\sqrt{d(i)}} - \frac{x(j)}{\sqrt{d(j)}}\right)^2 \geq 0.$$

Thus, $-1 \leq \boldsymbol{x}^T(\boldsymbol{I}-\tilde{\boldsymbol{A}})\boldsymbol{x} \leq 1$. Furthermore, using the vector $\boldsymbol{x} = \boldsymbol{D}^{\frac{1}{2}}\boldsymbol{1}$, we get

$$\tilde{\boldsymbol{A}}\boldsymbol{x} = \boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{A}\boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{D}^{\frac{1}{2}}\boldsymbol{1} = \boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{A}\boldsymbol{1} = \boldsymbol{D}^{-\frac{1}{2}}\mathrm{diag}(\boldsymbol{D}) = \boldsymbol{D}^{\frac{1}{2}}\boldsymbol{1}.$$

This implies that the largest eigenvalue of $\tilde{\boldsymbol{A}}$ is 1. □

REFERENCES

[1] Fabio Aiolli. 2013. Efficient top-N recommendation for very large scale binary rated datasets. In *Proceedings of the ACM Conference on Recommender Systems*. 273–280.

[2] Sanjeev Arora, Simon S Du, Zhiyuan Li, Ruslan Salakhutdinov, Ruosong Wang, and Dingli Yu. 2020. Harnessing the power of infinitely wide deep nets on small-data tasks. In *Proceedings of the International Conference on Learning Representations*.

[3] Muhammet Balcilar, Renton Guillaume, Pierre Héroux, Benoit Gaüzère, Sébastien Adam, and Paul Honeine. 2021. Analyzing the Expressive Power of Graph Neural Networks in a Spectral Perspective. In *Proceedings of the International Conference on Learning Representations*.

[4] Rianne van den Berg, Thomas N Kipf, and Max Welling. 2017. Graph convolutional matrix completion. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

[5] Kwan Ho Ryan Chan, Yaodong Yu, Chong You, Haozhi Qi, John Wright, and Yi Ma. 2021. ReduNet: A White-box Deep Network from the Principle of Maximizing Rate Reduction. *arXiv preprint arXiv:2105.10446* (2021).

[6] Chao Chen, Dongsheng Li, Junchi Yan, Hanchi Huang, and Xiaokang Yang. 2021. Scalable and Explainable 1-Bit Matrix Completion via Graph Signal Learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

[7] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep neural networks for youtube recommendations. In *Proceedings of the ACM Conference on Recommender Systems*. 191–198.

[8] Xiaowen Dong, Dorina Thanou, Laura Toni, Michael Bronstein, and Pascal Frossard. 2020. Graph signal processing for machine learning: A review and new perspectives. *IEEE Signal Processing Magazine* 37, 6 (2020), 117–127.

[9] David L Donoho and Michael Elad. 2003. Optimally sparse representation in general (nonorthogonal) dictionaries via ℓ_1 minimization. *Proceedings of the National Academy of Sciences* 100, 5 (2003), 2197–2202.

[10] Travis Ebesu, Bin Shen, and Yi Fang. 2018. Collaborative memory network for recommendation systems. In *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval*. 515–524.

[11] Noah E Friedkin. 2011. A formal theory of reflected appraisals in the evolution of power. *Administrative Science Quarterly* 56, 4 (2011), 501–529.

[12] Marco Gori, Augusto Pucci, V Roma, and I Siena. 2007. Itemrank: A random-walk based scoring algorithm for recommender engines.. In *Proceedings of the AAAI International Joint Conference on Artificial Intelligence*. 2766–2771.

[13] Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, Yongdong Zhang, and Meng Wang. 2020. Lightgcn: Simplifying and powering graph convolution network for recommendation. In *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval*. 639–648.

[14] Xiangnan He, Ming Gao, Min-Yen Kan, and Dingxian Wang. 2016. Birank: Towards ranking on bipartite graphs. *IEEE Transactions on Knowledge and Data Engineering* 29, 1 (2016), 57–71.

[15] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In *Proceedings of the International Conference on World Wide Web*. 173–182.

[16] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. 1989. Multilayer feedforward networks are universal approximators. *Neural networks* 2, 5 (1989), 359–366.

[17] Yifan Hu, Yehuda Koren, and Chris Volinsky. 2008. Collaborative filtering for implicit feedback datasets. In *Proceedings of the IEEE International Conference on Data Mining*. 263–272.

[18] Michel Journée, Yurii Nesterov, Peter Richtárik, and Rodolphe Sepulchre. 2010. Generalized power method for sparse principal component analysis. *Journal of Machine Learning Research* 11, 2 (2010).

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

[20] Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. 2019. Predict then propagate: Graph neural networks meet personalized pagerank. In *Proceedings of the International Conference on Learning Representations*.

[21] Yehuda Koren. 2008. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 426–434.

[22] Dongsheng Li, Chao Chen, Wei Liu, Tun Lu, Ning Gu, and Stephen M. Chu. 2017. Mixture-Rank Matrix Approximation for Collaborative Filtering. In *Proceedings of the Advances in Neural Information Processing Systems (NIPS'17)*. 477–485.

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

[24] Dawen Liang, Rahul G Krishnan, Matthew D Hoffman, and Tony Jebara. 2018. Variational autoencoders for collaborative filtering. In *Proceedings of the International World Wide Web Conference*. 689–698.

[25] G. Linden, B. Smith, and J. York. 2003. Amazon.com recommendations: item-to-item collaborative filtering. *IEEE Internet Computing* 7, 1 (2003), 76–80.

[26] Sheng Liu, Xiao Li, Yuexiang Zhai, Chong You, Zhihui Zhu, Carlos Fernandez-Granda, and Qing Qu. 2021. Convolutional Normalization: Improving Deep Convolutional Network Robustness and Training. *arXiv preprint arXiv:2103.00673* (2021).

[27] Xia Ning and George Karypis. 2011. SLIM: Sparse linear methods for top-n recommender systems. In *Proceedings of the IEEE International Conference on Data Mining*. IEEE, 497–506.

[28] Rong Pan, Yunhong Zhou, Bin Cao, Nathan N Liu, Rajan Lukose, Martin Scholz, and Qiang Yang. 2008. One-class collaborative filtering. In *Proceedings of the IEEE International Conference on Data Mining*. 502–511.

[29] Raksha Ramakrishna, Hoi-To Wai, and Anna Scaglione. 2020. A User Guide to Low-Pass Graph Signal Processing and Its Applications: Tools and Applications. *IEEE Signal Processing Magazine* 37, 6 (2020), 74–85.

[30] Nikhil Rao, Hsiang-Fu Yu, Pradeep Ravikumar, and Inderjit S Dhillon. 2015. Collaborative Filtering with Graph Information: Consistency and Scalable Methods. In *Proceedings of the Advances in Neural Information Processing Systems*, Vol. 2. 7.

[31] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2012. BPR: Bayesian personalized ranking from implicit feedback. In *Proceedings of the AUAI Conference on Uncertainty in Artificial Intelligence*.

[32] Steffen Rendle, Walid Krichene, Li Zhang, and John Anderson. 2020. Neural collaborative filtering vs. matrix factorization revisited. In *Proceedings of the ACM Conference on Recommender Systems*. 240–248.

[33] Luana Ruiz, Fernando Gama, and Alejandro Ribeiro. 2021. Graph Neural Networks: Architectures, Stability, and Transferability. *Proc. IEEE* (2021).

[34] Harald Steck. 2019. Embarrassingly shallow autoencoders for sparse data. In *Proceedings of the World Wide Web Conference*. 3251–3257.

[35] Harald Steck. 2019. Markov random fields for collaborative filtering. *Proceedings of the Advances in Neural Information Processing Systems* 32.

[36] Harald Steck. 2020. Autoencoders that don't overfit towards the Identity. *Proceedings of the Advances in Neural Information Processing Systems* 33.

[37] Yi Tay, Luu Anh Tuan, and Siu Cheung Hui. 2018. Latent relational metric learning via memory-based attention for collaborative ranking. In *Proceedings of the World Wide Web Conference*. 729–739.

[38] Roman Vershynin. 2018. *High-dimensional probability: An introduction with applications in data science*. Vol. 47. Cambridge university press.

[39] Koen Verstrepen and Bart Goethals. 2014. Unifying nearest neighbors collaborative filtering. In *Proceedings of the ACM Conference on Recommender systems*. 177–184.

[40] Pauli Virtanen, Ralf Gommers, Travis E Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, et al. 2020. SciPy 1.0: fundamental algorithms for scientific computing in Python. *Nature methods* 17, 3 (2020), 261–272.

[41] Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. 2019. Neural graph collaborative filtering. In *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval*. 165–174.

[42] John Wright and Yi Ma. 2021. *High-Dimensional Data Analysis with Low-Dimensional Models: Principles, Computation, and Applications*. Cambridge University Press.

[43] Yao Wu, Christopher DuBois, Alice X Zheng, and Martin Ester. 2016. Collaborative denoising auto-encoders for top-n recommender systems. In *Proceedings of the ACM International Conference on Web Search and Data Mining*. 153–162.

[44] Louis-Pascal Xhonneux, Meng Qu, and Jian Tang. 2020. Continuous graph neural networks. In *Proceedings of the International Conference on Machine Learning*. 10432–10441.

[45] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2018. How powerful are graph neural networks?. In *Proceedings of the International Conference on Learning Representations*.

[46] Keyulu Xu, Jingling Li, Mozhi Zhang, Simon S Du, Ken-ichi Kawarabayashi, and Stefanie Jegelka. 2020. How neural networks extrapolate: From feedforward to graph neural networks. In *Proceedings of the International Conference on Learning Representations*.

[47] Keyulu Xu, Mozhi Zhang, Stefanie Jegelka, and Kenji Kawaguchi. 2021. Optimization of Graph Neural Networks: Implicit Acceleration by Skip Connections and More Depth. In *Proceedings of the International Conference on Machine Learning*.

[48] Jheng-Hong Yang, Chih-Ming Chen, Chuan-Ju Wang, and Ming-Feng Tsai. 2018. HOP-rec: high-order proximity for implicit recommendation. In *Proceedings of the ACM Conference on Recommender Systems*. 140–144.

[49] 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 ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 974–983.

[50] Wenhui Yu and Zheng Qin. 2020. Graph Convolutional Network for Recommendation with Low-pass Collaborative Filters. In *Proceedings of the International Conference on Machine Learning*. PMLR, 10936–10945.

[51] Yao Zhang, Yun Xiong, Dongsheng Li, Caihua Shan, Kan Ren, and Yangyong Zhu. 2021. CoPE: Modeling Continuous Propagation and Evolution on Interaction Graph. In *Proceedings of the International ACM Conference on Information and Knowledge Management*.

[52] Lei Zheng, Chun-Ta Lu, Fei Jiang, Jiawei Zhang, and Philip S Yu. 2018. Spectral collaborative filtering. In *Proceedings of the ACM conference on Recommender Systems*. 311–319.