Low-Pass Graph Convolutional Network for Recommendation

Wenhui Yu[*1], **Zixin Zhang**[*2], **Zheng Qin**[2]

[1]Alibaba Group,
[2]Tsinghua University
jianlin.ywh@alibaba-inc.com, zzx20@mails.tsinghua.edu.cn, qingzh@mail.tsinghua.edu.cn

Abstract

Spectral graph convolution is extremely time-consuming for large graphs, thus existing Graph Convolutional Networks (GCNs) reconstruct the kernel by a polynomial, which is (almost) fixed. To extract features from the graph data by learning kernels, Low-pass Collaborative Filter Network (LCFN) was proposed as a new paradigm with trainable kernels. However, there are two demerits of LCFN: (1) The hypergraphs in LCFN are constructed by mining 2-hop connections of the user-item bipartite graph, thus 1-hop connections are not used, resulting in serious information loss. (2) LCFN follows the general network structure of GCNs, which is suboptimal. To address these issues, we utilize the bipartite graph to define the graph space directly and explore the best network structure based on experiments. Comprehensive experiments on two real-world datasets demonstrate the effectiveness of the proposed model. Codes are available on https://github.com/Wenhui-Yu/LCFN.

Introduction

Convolutional **N**eural **N**etworks (**CNN**s) gain strong power of extracting features by learning kernels (Krizhevsky, Sutskever, and Hinton 2012). To explore convolution in the graph learning tasks such as recommendation, it is extended from the Euclidean domain to the graph domain (Shuman et al. 2013), and is then injected to a deep structure to propose **G**raph **C**onvolutional **N**etworks (**GCN**s) (Kipf and Welling 2017; Defferrard, Bresson, and Vandergheynst 2016; He et al. 2020; Hamilton, Ying, and Leskovec 2017). However, since graph convolution is time-consuming for large graphs, Kipf and Welling (2017) simplified it by fixing the kernel into a 1-order polynomial, thus GCNs lose the ability of extracting features and only propagate embeddings through the graph.

To close this gap, Yu and Qin (2020) proposed an unscathed way to simplify the (spectral) graph convolution by a **L**ow-pass **C**ollaborative **F**ilter (**LCF**). Graph convolution is defined in the frequency domain (Shuman et al. 2013) and all-frequency components of the signal participate in calculation. Filtered by LCF, only a very small proportion of the components (low-frequency components) are reserved and need to be calculated, thus the efficiency is increased greatly. LCF and the graph convolution are used to design an end-to-end GCN, called **LCF N**etwork (**LCFN**), where kernels are trainable. Another benefit of the LCF is to remove the noise. Due to the random exposure and quantization, observed interactions reflect user preference yet contain noise. Yu and Qin (2020) pointed out that the user preference is low-frequency and the noise is high-frequency, thus can be separated by LCF. As we can see, LCF contributes to both efficiency and effectiveness aspects.

However, there is still great room for improvement. Firstly, LCFN used 2D graph convolution, with user-user and item-item hypergraphs defining the graph spaces of the two dimensions. These hypergraphs are constructed by the 2-hop connections of the user-item bipartite graph, thus there is no 1-hop connections, resulting in information loss. For example, when performing convolution on a user, items cannot contribute to the result. What's more, since the feature maps are large and dense matrices, they are factorized in LCFN for storage, making 2D convolution of an interaction matrix degenerates into twice 1D convolution of embeddings. As we can see, LCFN pays the cost of information loss, but does not achieve real 2D graph convolution. Considering this issue, we utilize the observed bipartite graph to define the graph space directly. To be specific, we treat the bipartite graph as a homogeneous graph and design 1D graph convolution. In this case, both users and items contribute to the convolution on a certain user.

Secondly, LCFN focuses attention on introducing a new graph convolution while simply follows the general network structure and model settings of GCNs. Many unnecessary structures and suboptimal model settings are used, increasing the difficulty of model training and degrading the performance. To solve these issuses, we design experiments to explore the optimal structure and settings for the proposed model.

Finally, we inject the 1D spectral graph convolution into the tuned network, and propose our **L**ow-pass **G**raph **C**onvolutional **N**etwork (**LGCN**). To sum up, the contributions are as follows:

- We propose 1D graph convolution to leverage the direct connections of the observed graph. Then, we design a 1D LCF, and integrate it with 1D graph convolution as the low-pass graph convolution.

[*]These authors contributed equally.

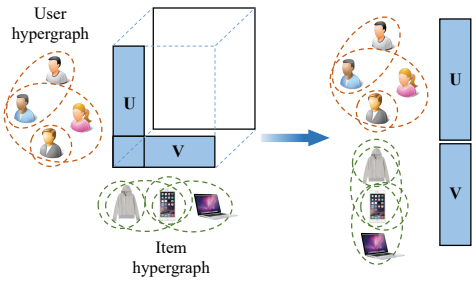
- We inject 1D low-pass graph convolution to a deep structure. We design comprehensive experiments to explore the optimal model structure and settings.
- We conduct experiments on two real-world datasets to validate the effectiveness of our proposed model. Experiments show that the performance of this model is significantly better than state-of-the-art models.

Related Work

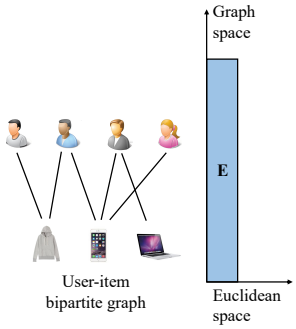
Recommender system is designed to learn user preference from the interaction history, such as clicking, purchasing, etc., and return top items to each user. The most widely-used recommendation model is the latent variable model (Koren 2009; He et al. 2017; He and Chua 2017) which encodes user preference and item properties by embeddings, and measures the user preference towards items based on the distance between the embeddings. In latent variable models, embeddings are the most important parameters. In order to learn better embeddings, many variants have been proposed (Ying et al. 2018; Berg, Kipf, and Welling 2018; Wu et al. 2019b; He et al. 2020; Yu and Qin 2020).

Graph **N**eural **N**etwork (**GNN**) (Wang et al. 2019; Wu et al. 2019b; Ying et al. 2018; Berg, Kipf, and Welling 2018) is one of the most effective paradigm for representation learning. GNNs refine the embeddings of each node with that of the neighbors. To be specific, GNNs propagate embeddings through the bipartite graph. Embeddings are propagated to the 1-hop neighbors by one propagation layer, and by stacking several layers, GNNs achieve long-distance propagation. For each node, the embeddings are refined by the neighbors within L hops, where L is the depth. GNNs ensure connected nodes have similar embeddings thus provide an explicit way to leverage collaborative information. However, GNNs only utilize the graphs to smooth embeddings. Deep models which can utilize graphs to extract high-level features are desired.

Inspired by CNN, graph convolution is proposed to extract features for graph data. Shuman et al. (2013) extended convolution from the Euclidean domain to the graph domain and Defferrard, Bresson, and Vandergheynst (2016) proposed GCN which leverages graph convolution to extract high-level features. To utilize graph convolution for recommendation tasks, Kipf and Welling (2017) improved the efficiency by fixing the kernel, yet made GCNs degenerate into GNNs. Following Kipf and Welling (2017), most existing GCNs (Wang et al. 2019; Ying et al. 2018; Berg, Kipf, and Welling 2018; He et al. 2020) become GNNs or the variants. To deal with this issue, Yu and Qin (2020) proposed a low-pass graph convolutional network with learnable kernels, called LCFN. However, 2D graph convolution of LCFN leads to the information loss. Also, the network structure and model settings are rather suboptimal. Wu et al. (2019a); He et al. (2020) pointed out that the unnecessary components contribute negatively to the model performance. In this paper, we explore better low-pass graph convolution and better model structure.



(a) 2D graph convolution in LCFN



(b) 1D graph convolution in this paper

Figure 1: The comparison between 1D and 2D graph convolution

1D Low-pass Graph Convolution

In this section, we introduce 1D low-pass graph convolution. $\mathbf{U} \in \mathbb{R}^{M \times K}$ and $\mathbf{V} \in \mathbb{R}^{N \times K}$ are user and item embeddings, where K is the dimensionality. $\mathbf{E} = \begin{bmatrix} \mathbf{U} \\ \mathbf{V} \end{bmatrix}$ is the node embedding matrix.

Figure 1 shows the comparison of 2D and 1D low-pass graph convolution. We use dotted lines to indicate hyperedges of hypergraphs and use solid lines to indicate edges of the graph. Blue rectangles indicate embeddings and the white rectangle indicates the feature map. As shown in Figure 1(a), 2D graph convolution on feature maps becomes twice 1D graph convolution on embeddings in LCFN. Figure 1(b) shows the 1D graph convolution on $\mathbf{E}$ proposed in this paper. Although $\mathbf{E}$ is a 2D matrix, it is indeed 1D graph data since only one dimension of $\mathbf{E}$ is in the graph space and the other dimension is in the Euclidean space. That means only the adjacency relationship between rows is defined by the graph, and the adjacency relationship between columns is still determined by their position. As a result, only 1D graph convolution is needed.

Since that graph convolution is defined in the frequency domain, we first introduce the graph Fourier transform, and then design the low-pass graph filter LCF and graph convolution. Finally we integrate them to obtain the low-pass graph convolution. To formulate each part concisely, we use certain column of the embedding $\mathbf{e} = \mathbf{E}_{*k}$ as the graph data,

and for all columns, we do the same thing.

Graph Fourier Transform

1D Fourier transform is defined as the inner product of signal and bases of the frequency domain:

$$F(\omega) = \int_{\mathbb{R}} f(t) \mathrm{e}^{-j\omega t} dt = \left\langle f(t), \mathrm{e}^{-j\omega t} \right\rangle, \tag{1}$$

where $f(t)$ is the time-domain waveform, representing change of the signal over time. $F(\omega)$ is the frequency-domain waveform, representing the distribution of the signal on the frequency. $\mathrm{e}^{-j\omega t}$ is the base of the frequency domain, where j is the imaginary unit and ω is the angular frequency. $\mathrm{e}^{-j\omega t}$ satisfies: $\nabla_t \mathrm{e}^{-j\omega t} = -j\omega \mathrm{e}^{-j\omega t}$ thus is the eigen-function of the differential operator in the time domain, and the eigen-value $-j\omega$ defines the frequency of corresponding base. To extend this definition to graph data, we need to (1) introduce the difference operator in the graph space; (2) construct the eigen-function to get bases; and (3) calculate the inner product of the signal and the base. The difference operator in the graph space is Laplacian matrix: $\mathbf{L} = \mathbf{I} - \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$, where $\mathbf{A} = \begin{bmatrix} \mathbf{0} & \mathbf{R} \\ \mathbf{R}^\mathsf{T} & \mathbf{0} \end{bmatrix}$ is adjacency matrix of the bipartite graph and $\mathbf{D}$ is the diagonal degree matrix: $\mathbf{D}_{ii} = \sum_j \mathbf{A}_{ij}$. For graph data $\mathbf{e}$ (i.e., $\mathbf{E}_{*k}$), the result of Laplacian matrix performing on it is:

$$(\mathbf{L}\mathbf{e})_i = \frac{1}{\sqrt{\mathbf{D}_{ii}}} \sum_{j \in \mathcal{N}_i} \left(\frac{\mathbf{e}_i}{\sqrt{\mathbf{D}_{ii}}} - \frac{\mathbf{e}_j}{\sqrt{\mathbf{D}_{jj}}} \right) \tag{2}$$

where $\mathcal{N}_i$ is the set of connected nodes of i. Equation (2) shows that the effect of Laplacian matrix $\mathbf{L}$ is to take difference of each node in the neighborhood.

As the difference operator is discrete, we use eigen-vectors rather than eigen-functions as the bases of the frequency space. Performing eigen-decomposition, we get: $\mathbf{L} = \mathbf{P}\mathbf{\Lambda}\mathbf{P}^\mathsf{T}$, where $\mathbf{P}$ is the eigen-vector matrix. Columns of $\mathbf{P}$ can form a set of normalized orthogonal bases, which defines the frequency domain. $\mathbf{\Lambda} = diag([\lambda_1, \cdots, \lambda_{M+N}])$ is the diagonal eigen-value matrix and all eigen-values are arranged in ascending order. Eigen-values define the frequency of the corresponding eigen-vector. Now, we can extend Equation (1) to graph data. Graph Fourier transform is the inner product of the signal $\mathbf{e}$ and the bases $\{\mathbf{P}_{*\phi}\}_{\phi=1,\cdots,M+N}$:

$$\tilde{\mathbf{e}}_\phi = \langle \mathbf{e}, \mathbf{P}_{*\phi} \rangle = \sum_{i=1}^{M+N} \mathbf{e}_i \mathbf{P}_{i\phi}.$$

For each column of $\mathbf{E}$, we calculate the inner product. The graph Fourier transform for entire embeddings $\mathbf{E}$ written in a closed form is: $\tilde{\mathbf{E}} = \mathcal{F}_g(\mathbf{E}) = \mathbf{P}^\mathsf{T}\mathbf{E}$, where $\tilde{\mathbf{E}} \in \mathbb{R}^{(M+N)\times K}$ is the frequency-domain waveform of $\mathbf{E}$. In this paper, we use the tilde to mark the signal in frequency domain. Similarly, we can get the inverse graph Fourier transform: $\mathbf{E} = \mathcal{F}_g^{-1}(\tilde{\mathbf{E}}) = \mathbf{P}\tilde{\mathbf{E}}$, where $\mathcal{F}_g(\ )$ and $\mathcal{F}_g^{-1}(\ )$ represent the graph Fourier transform and its inverse transform, respectively. In graph Fourier transform, the time domain is the graph space, and the frequency domain is defined by the eigen-values: $[\lambda_1, \cdots, \lambda_{M+N}]$. The ϕ-th row $\tilde{\mathbf{E}}_\phi$ is the component of the signal $\mathbf{E}$ at the frequency λ_ϕ. Thus, $\tilde{\mathbf{E}}_\phi$ can be denoted as $\tilde{\mathbf{E}}(\lambda_\phi)$.

Low-pass Graph Filter

The low-pass graph filter removes high-frequency components in the frequency domain and retains low-frequency components. We use a 1D gate function to define the filter in frequency domain: $\tilde{\mathbf{f}} = \begin{bmatrix} \mathbf{1} \\ \mathbf{0} \end{bmatrix}$, where $\tilde{\mathbf{f}} \in \{0,1\}^{M+N}$, $\mathbf{1} \in \{1\}^{\Phi}$ is an all-one column vector; $\mathbf{0} \in \{0\}^{M+N-\Phi}$ is an all-zero column vector; and Φ (strictly speaking, λ_Φ) is the passband cutoff frequency. To remove the high-frequency components, we need to transform $\mathbf{e}$ into the frequency domain, and conduct element-wise product with the filter $\tilde{\mathbf{f}}$, and finally transform it back to the time domain: $LCF(\mathbf{e}) = \mathcal{F}_g^{-1}(\mathcal{F}_g(\mathbf{e}) \odot \tilde{\mathbf{f}})$. Since the graph filter uses the connections, i.e., collaborative information in the graph, it is named Low-pass Collaborative Filter (LCF). For the entire embeddings matrix $\mathbf{E}$, we have:

$$LCF(\mathbf{E}) = \mathcal{F}_g^{-1}\Big(diag\big(\tilde{\mathbf{f}}\big) \cdot \mathcal{F}_g(\mathbf{E})\Big) = \bar{\mathbf{P}}\bar{\mathbf{P}}^\mathsf{T}\mathbf{E}, \tag{3}$$

where $\bar{\mathbf{P}} = \mathbf{P}_{*,1:\Phi}$ are the first Φ eigen-vectors. We can find that when $\Phi = M + N$, the filter becomes an all-pass filter and do not change the embeddings $\mathbf{E}$. By tuning LCF with respect to Φ, we can retain the useful signal and remove noise as much as possible. We illustrate the effectiveness of the 1D LCF by comparing it against GCN and 2D LCF in LCFN in the Analysis of 1D LCF Section.

Graph Convolution

Convolution theorem tells us that convolution in the time domain is equivalent to multiplication in the frequency domain (Barrett and Wilde 1960). Hence, to perform graph convolution on embeddings with the convolution kernel, we only need to transform them into frequency domain by graph Fourier transform, multiply them, and transform the result back to time domain: $\mathbf{e} *_g \mathbf{k} = \mathcal{F}_g^{-1}\left(\mathcal{F}_g(\mathbf{e}) \odot \tilde{\mathbf{k}}\right)$, where $*_g$ represents graph convolution; column vector $\mathbf{k} \in \mathbb{R}^{M+N}$ is convolution kernel and $\tilde{\mathbf{k}} = \mathcal{F}_g(\mathbf{k})$ is the corresponding frequency-domain waveform. In our model, we define and train the kernel directly in frequency domain. For embeddings $\mathbf{E}$, the convolution result is:

$$\begin{aligned} \mathbf{E} *_g \mathbf{k} =& \mathcal{F}_g^{-1}\left(diag(\tilde{\mathbf{k}}) \cdot \mathcal{F}_g(\mathbf{E})\right) \\ =& \mathbf{P} diag(\tilde{\mathbf{k}}) \mathbf{P}^\mathsf{T}\mathbf{E}, \end{aligned} \tag{4}$$

In our graph convolutional neural network, the convolution kernel $\tilde{\mathbf{k}}$ of each graph convolutional layer can be learned from the data.

Low-pass Graph Convolution

For the embeddings, we first smooth it by LCF, and then extract high-level features by graph convolution, and we can

get the low-pass graph convolution:

$$\begin{aligned}\mathbf{E}\bar{*}_g\mathbf{k} =&\mathcal{F}_g^{-1}\left(diag(\tilde{\mathbf{f}})\cdot diag(\tilde{\mathbf{k}})\cdot\mathcal{F}_g(\mathbf{E})\right)\\ =&\bar{\mathbf{P}}diag(\bar{\mathbf{k}})\bar{\mathbf{P}}^\top\mathbf{E},\end{aligned}\tag{5}$$

where $\bar{*}_g$ indicates low-pass graph convolution; $\bar{\mathbf{P}}=\mathbf{P}_{*,1:\Phi}$ contains the first Φ eigen-vectors and $\bar{\mathbf{k}}=\tilde{\mathbf{k}}_{1:\Phi}$ is the convolutional kernel of low-pass graph convolution. Comparing Equations (4) and (5), we can see that in graph convolution, all the eigen-vectors need to be constructed and included in the convolution computation, while in the low-pass graph convolution, only the first Φ eigen-vectors are required. The time cost of constructing $\mathbf{P}$ is $O((M+N)^3)$. However, since $\mathbf{L}$ is sparse, $\bar{\mathbf{P}}$ can be constructed efficiently by Lanczos algorithm (Grimes, Lewis, and Simon 1994) with $O(n\Phi^2)$ time, where n is the number of non-zero elements in $\mathbf{L}$. Experiments show that $\Phi\ll M+N$, so our low-pass graph convolution is much more efficient.

Analysis of 1D LCF

1D LCF is the key component of low-pass graph convolution. Here, we illustrate the effectiveness of our proposed 1D LCF by comparing it against 2D LCF in LCFN and existing GCNs.

Compared with 2D LCF

Yu and Qin (2020) expounded the effect of 2D LCF from the perspective of removing the high-frequency noise in the user-item interaction matrix, while in this paper, we provide another perspective of smoothing embeddings, and they are equivalent (Yu and Qin 2020).

Though with the same principle, 1D LCF works better than 2D LCF since they use different graphs. 1D LCF utilizes the observed graph and 2D LCF utilizes two hypergraphs, thus 1-hop connections are lost in 2D LCF. For example, when smoothing user embeddings, only user embeddings are used in 2D LCF yet all embeddings are used in our 1D LCF.

Compared with GCN

In fact, the propagation in vanilla GCNs is also a low-pass graph filter. The difference between propagation and LCF is that LCF can remove noise and retain signal more thoroughly. In other words, our LCF is an advanced version of propagation with better effect. We will demonstrate this viewpoint in both time domain and frequency domain.

Time domain We first compare LCF with propagation in the time domain. Considering there are many variants of GCNs with subtle differences, we take He et al. (2020) as an example. Adjacency matrix with symmetric normalization $\mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$ is used for propagation. Embeddings after 1-hop propagation is $\mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}\mathbf{E}$. In this propagation strategy, the observed connection is used to measure the similarity (propagation weight) between nodes. If nodes i and j are connected, the similarity is $\frac{1}{\sqrt{\mathbf{D}_{ii}\mathbf{D}_{jj}}}$, and the similarity is 0 otherwise. In the recommendation tasks, only a small part of the connections can be observed due to insufficient exposure. Therefore, this similarity measurement is ineffective and biased. For example, if the connection between nodes i and j is missed, their embeddings cannot be propagated to each other. To address this issue, GCNs connect them by multi-hop connections. However, the embeddings decay seriously through a long distance propagation, therefore contributing little to refining the embeddings of each other. LCF in the time domain is a propagation matrix $\bar{\mathbf{P}}\bar{\mathbf{P}}^\top$. The similarity of i and j is determined by the inner product of the vector: $(\bar{\mathbf{P}}\bar{\mathbf{P}}^\top)_{ij}=\bar{\mathbf{P}}_i\bar{\mathbf{P}}_j^\top$. Ng, Jordan, and Weiss (2002) pointed out that $\bar{\mathbf{P}}$ encodes the graph structure and can be used to measure the similarity between nodes in node clustering. $\bar{\mathbf{P}}\bar{\mathbf{P}}^\top$ is a similarity metric that takes multi-hop connections into account. Compared to the normalized adjacency matrix, $\bar{\mathbf{P}}\bar{\mathbf{P}}^\top$ is more effective and unbiased. For example, if the connection of nodes i and j is missed, there are still strong multi-hop connections since they are adjacent nodes. In this case $(\bar{\mathbf{P}}\bar{\mathbf{P}}^\top)_{ij}$ can be very large, and their embeddings can be propagated to each other effectively.

Frequency domain Next, we compare the LCF against propagation in the frequency domain. 1-hop propagation is:

$$\begin{aligned}&\mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}\mathbf{E}=(\mathbf{I}-\mathbf{L})\mathbf{E}=\mathbf{P}(\mathbf{I}-\mathbf{\Lambda})\mathbf{P}^\top\mathbf{E}\\ =&\mathcal{F}_g^{-1}\cdot(diag([1-\lambda_1,\cdots,1-\lambda_{M+N}])\cdot\mathcal{F}_g(\mathbf{E}))\end{aligned}\tag{6}$$

Comparing Equation (3) with Equation (6), we can conclude that propagation in the time domain is equivalent to a low-pass graph filter in the frequency domain, and the form of the filter is: $[1-\lambda_1,\cdots,1-\lambda_{M+N}]$. As introduced in the main body, the eigen-values $[\lambda_1,\cdots,\lambda_{M+N}]$ are arranged in ascending order, so the elements of filter $[1-\lambda_1,\cdots,1-\lambda_{M+N}]$ are arranged in descending order. We can see that the component of frequency λ_ϕ is attenuated by $1-\lambda_\phi$. According to the property of the Laplacian matrix, we have $0=\lambda_1<\lambda_2\leq\cdots\leq\lambda_{M+N}\leq 2$ (Shuman et al. 2013). The direct component (frequency λ_1 is 0) is all preserved, and with the increasing of the frequency λ_ϕ, the attenuation coefficient $|1-\lambda_\phi|$ decreases first and then increases (noting that a signal attenuated by a negative attenuation coefficient $1-\lambda_\phi$ equals to attenuated by $|1-\lambda_\phi|$ and invert the phase of the signal). It is evident that this filter cannot retain the low-frequency components and suppress the high-frequency components thoroughly. On the contrary, the frequency form of LCF is a gate function. The passband cut-off frequency λ_Φ is determined by estimating the frequency range of the signal and noise. By separating the signal and noise with λ_Φ, the signal can be completely reserved and the noise can be completely removed.

In long-distance propagation of conventional GCNs, we face over-smooth issue. It is easy to explain in the frequency domain. After k-hop propagation, we have:

$$\begin{aligned}&\left(\mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}\right)^k\mathbf{E}\\ =&\mathcal{F}_g^{-1}\left(diag\left(\left[(1-\lambda_1)^k,\cdots,(1-\lambda_{M+N})^k\right]\right)\cdot\mathcal{F}_g(\mathbf{E})\right).\end{aligned}$$

We can see that the filter becomes $\left[(1-\lambda_1)^k,\cdots,(1-\right.$

If with transformation	With	Without
F_1-score@2	0.01975	0.01995

Table 1: Effectiveness of transformation

Activation	–	Sigmoid	ReLU	Tanh
F_1-score@2	0.01995	0.01845	0.01911	0.01977

Table 2: Effectiveness of different activation functions

$\lambda_{M+N})^k\big]$, thus except the direct component, all components decay significantly with the increasing of k. However, filtering by LCF k times will not lead to over-smooth issue, since $\tilde{\mathbf{f}} \odot \cdots \odot \tilde{\mathbf{f}} = \tilde{\mathbf{f}}$ and filtered by LCF k times equals filtered by LCF one time:

$$\begin{aligned} &LCF(\cdots LCF(\mathbf{E})) = \left(\bar{\mathbf{P}}\bar{\mathbf{P}}^\mathsf{T}\right)^k\mathbf{E} \\ =&\bar{\mathbf{P}}\left(\bar{\mathbf{P}}^\mathsf{T}\bar{\mathbf{P}}\right)^{k-1}\bar{\mathbf{P}}^\mathsf{T}\mathbf{E} = \bar{\mathbf{P}}\bar{\mathbf{I}}^{k-1}\bar{\mathbf{P}}^\mathsf{T}\mathbf{E} \\ =&\bar{\mathbf{P}}\bar{\mathbf{P}}^\mathsf{T}\mathbf{E} = LCF(\mathbf{E}). \end{aligned}$$

We benefit from LCF in two aspects: (1) By smoothing embeddings, LCF uses collaborative information explicitly to enhance learning embeddings. (2) More importantly, it improves the efficiency of original graph convolution significantly, and makes it possible to design a GCN with trainable convolution kernels. As analyzed before, our model with only LCF is better than existing GCNs. By additionally learning convolutional kernels to extract features, our model is more powerful in predicting user preference.

Exploring the Optimal Model Structure

In this section, we design experiments to explore the best structure for LGCN. We follow He et al. (2020) to initialize the experimental settings of the experiments in this section, that is: no transformation; no activation; sum pooling; inner product for prediction; BPR loss; Adam as optimizer. F_1-score@2 is reported on the validation set of *Amazon*.

Graph Convolutional Layer

There are graph convolution, transformation and activation modules in general graph convolutional layer.

Low-pass Graph Convolution Given Equation (5), we can perform low-pass graph convolution on $\mathbf{E}$ with kernel $\bar{\mathbf{k}}$ to obtain the feature map $\mathbf{E}'$.

Transformation Embedding transformation is widely used in GCNs (Berg, Kipf, and Welling 2018; Kipf and Welling 2017; Yu and Qin 2020; Zhou et al. 2018). It is used to transform feature maps into a new space. We design experiments to verify the effectiveness of transformation. The result in Table 1 indicates that transformation brings no benefits. We consider that the feature maps and original embeddings are still in the same latent space, leading it unhelpful to do spatial transformation.

Activation In our experiments, we try sigmoid, ReLU, and tanh. Result is reported in Table 2. An interesting observation is that LGCN performs better with no activation or tanh than with sigmoid or ReLU. The possible reason is that activated by sigmoid or ReLU, all dimensions in embeddings are positive. However, negative values means the user/item does not match this latent factor. This information is very important in the encoding preference. We discard activation function since it brings no improvement.

As we can see, in the graph convolutional layer, only the low-pass graph convolution is indispensable and all other operations lead no enhancement.

Network Structure

Our model has a multi-layer structure, including an embedding layer and L graph convolutional layers. We use superscript (l) to mark parameters of the l-th graph convolutional layer. Feature maps from the previous layer $\mathbf{E}^{(l-1)}$ is input in current layer and we get the new feature maps $\mathbf{E}^{(l)}$:

$$\mathbf{E}^{(l)} = \bar{\mathbf{P}}diag(\bar{\mathbf{k}}^{(l)})\bar{\mathbf{P}}^\mathsf{T}\mathbf{E}^{(l-1)}.$$

We can get $L+1$ embeddings $\{\mathbf{E}^{(l)}\}_{l=0,\cdots,L}$, where $\mathbf{E}^{(0)}$ indicates embeddings of the embedding layer. Next, we pool the $L+1$ embeddings to get the predictive embeddings, and then make prediction by inputting them into a predictive function. In this subsection, we explore pooling and predictive functions.

Pooling We explore five pooling methods: concatenation, weighted sum (He et al. 2020), element-wise product, max pooling, and **M**ulti-**L**ayer **P**erceptron (**MLP**). Moreover, we try 1-layer and 3-layer MLP, denoted as MLP_1L and MLP_3L respectively. Table 3 shows that the performance of sum pooling is the best, so our model adopts sum pooling.

Pooling	F_1-score@2	Pooling	F_1-score@2
Concatenation	0.01854	Sum	0.01995
Product	0.01775	Maximum	0.01741
MLP_1L	0.01785	MLP_3L	0.01524

Table 3: Effectiveness of different pooling methods

Predictive Function After getting the predictive embeddings $\mathbf{E}$, we split it into user embeddings $\mathbf{U} = \mathbf{E}_{1:M}$ and item embeddings $\mathbf{V} = \mathbf{E}_{M+1:M+N}$. For a pair of user u and item i, the preference score $\hat{\mathbf{R}}_{ui}$ is predicted based on $\mathbf{U}_u$ and $\mathbf{V}_i$. Commonly-used predictive functions are inner product and MLP. For inner product, $\hat{\mathbf{R}}_{ui} = \mathbf{U}_u\mathbf{V}_i^\mathsf{T}$, and for MLP, $\hat{\mathbf{R}}_{ui} = MLP([\mathbf{U}_u, \mathbf{V}_i, \mathbf{U}_u \odot \mathbf{V}_i])$. Table 4 shows that inner product performs best, thus we adopt inner product in LGCN. MLP_3L performs much worse than the others and we can also conclude that light network performs best.

Functions	Inner product	MLP_1L	MLP_3L
F_1-score@2	0.01995	0.01910	0.01616

Table 4: Effectiveness of different predictive functions

Loss functions	BPR	Cross-entropy	MSE
F_1-score@2	0.01995	0.01982	0.01795

Table 5: Effectiveness of different loss functions

ρ	1	2	3	4	5
F_1-score@2	0.01995	0.02018	0.02027	0.02003	0.02011

Table 6: Performances with different sampling rates ρ

Generalization	F_1-score@2	Generalization	F_1-score@2
reg_1	0.01995	reg_2	0.01976
Dropout	0.01919	reg_1+dropout	0.01983
Normalization	0.01528		

Table 7: Effectiveness of different generalization methods

Optimization Settings

In this subsection, we design optimization strategies for LGCN, including loss functions, sample rate, generalization strategy and optimization methods.

Loss Function We explore three loss functions: **B**ayesian **P**ersonalized **R**anking (**BPR**) (Rendle et al. 2009), cross-entropy, and **M**ean **S**quare **E**rror (**MSE**). Table 5 shows that pairwise loss BPR performs better than point-wise losses cross-entropy and MSE. The reason is that the classification quality and the rank quality are both important in recommendation tasks with implicit feedbacks.

Sample Rate The impact of sampling rate ρ is shown in Table 6. As we can see, the model performs best when $\rho = 3$. By tuning the model with respect to ρ, we achieve 1.6% improvement. However, the improvement is very marginal (1.6%) and the training efficiency is impacted. To balance the effectiveness and efficiency, we set $\rho = 1$ in our experiments, and in real-world application, we can set a large ρ to get the best performance.

Generalization Strategy To prevent overfitting, we try three generalization strategies: L2 regularization (reg_1 indicates performing regularization only on embeddings and reg_2 indicates regularization on all parameters), dropout and L2 normalization (normalize each row of $\mathbf{E}$). Table 7 illustrates that regularization is the best generalization strategy for LGCN, and only embeddings need to be decayed.

Optimizer We explore four optimizers: **S**tochastic **G**radient **D**escent (**SGD**), **Ada**ptive **Grad**ient Algorithm (**AdaGrad**) (Duchi, Hazan, and Singer 2010), **R**oot **M**ean **S**quare **Prop**agation (**RMSProp**) (Tieleman and Hinton 2012) and **Ada**ptive **M**oment Estimation (**Adam**) (Kingma and Ba 2015). Table 8 shows that optimizers with momentum perform better than vanilla SGD and RMSProp performs best. Therefore, we adopt RMSProp in LGCN.

From experimental results shown above, we draw the conclusion that simple model settings usually bring better performance in recommendation tasks. Thus, the structure of our LGCN is a very simple and only the indispensable parts are reserved (please see Figure 2).

Experiment

Experiments are conducted in this section to demonstrate the effectiveness of LGCN by comparing it against several state-of-the-art models on two real-world datasets. We also demonstrate the two key novelties of this paper: 1D low-pass graph convolution and model structure by experiments.

Experimental Setup

We strictly follow the experiment settings of Yu and Qin (2020). For details about the datasets, baselines, and tuning strategies, please see Yu and Qin (2020).

Performance of LGCN

The performance of all models on two datasets is shown in Table 9. F_1-score and NDCG@{2,5,10,20,50,100} are in Appendix. Considering longer predictive embeddings mean stronger representation ability and more time consumption for inference, we follow (Yu and Qin 2020) to set the same dimensionality K of $\mathbf{E}$ for all models in this experiments. Since GCMC, NGCF, LCFN leverage concatenation for pooling yet LightGCN, LGCN leverage sum pooling, the dimensionality of $\mathbf{E}^{(0)}$ is $\frac{K}{L+1}$ in GCMC, NGCF, LCFN, and is K in MF, LightGCN, LGCN, i.e., there are approximately $L+1$ times learnable parameters in MF, LightGCN, LGCN of those in GCMC, NGCF, and LCFN. As a result, GCMC and NGCF cannot outperform MF significantly. However, even in this case, LCFN achieves significant improvement over MF (22.00% for the best case) due to the strong representation power from learning kernels.

Comparing MF and LightGCN, we can see that by propagating embeddings through the graph, GCNs learn better embeddings and outperform the basic MF dramatically. By learning kernels to extract high-level features, LGCN gets further improvement and outperforms LightGCN significantly (9.38%). By utilizing 1-hop connections in constructing graph convolution and exploring better model structure, LGCN outperforms LCFN (12.57% for the best case).

Model Tuning

The result of model tuning is shown in Figure 3. For a fair comparison, we tune all models with the same strategy. To save space, we only report the results on *Amazon* dataset on account of the similar performance 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 μ for each K, L, and F in Figure 3.

Figure 3(a) illustrates how the dimensionality K impacts the models. As shown in the figure, we gain better

Optimizer	SGD	AdaGrad	RMSProp	Adam
F_1-score@2	0.01938	0.01989	0.02005	0.01995

Table 8: Effectiveness of different optimizer

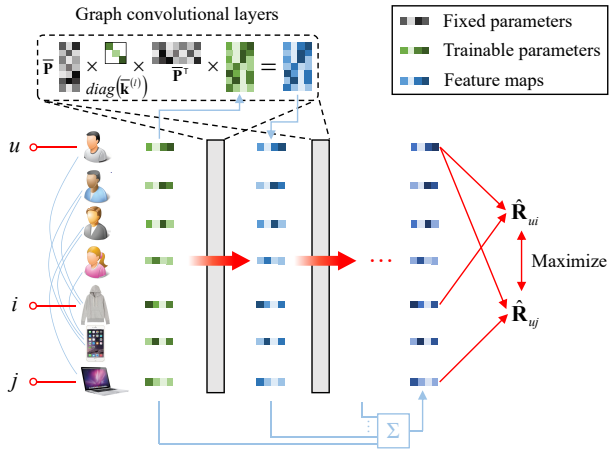


Figure 2: Illustration of LGCN

performance by increasing the representation ability. The number of layers L is analyzed in Figure 3(b). Obviously, with the increase of L, the performance of LightGCN increases first and then decreases slightly due to the over-smooth issue. Since there is no activation and transformation in LGCN, performing convolution L times equals to one time: $\mathbf{E}\bar{*}_g\mathbf{k}^{(1)}\bar{*}_g\cdots\bar{*}_g\mathbf{k}^{(L)} = \mathbf{E}\bar{*}_g\mathbf{k}$, where $\mathbf{k} = \mathbf{k}^{(1)}\bar{*}_g\cdots\bar{*}_g\mathbf{k}^{(L)}$. This is also supported by the experiments: the performance of LGCN keeps constant with the increase of L. Figure 3(c) shows the sensitivity analysis of the cut-off frequencies Φ. When Φ is too small, embeddings are over-filtered and much useful information is corrupted. Comparing LGCN with LCFN, we can see that the 1D low-pass graph convolution requires high pass band cut-off frequency.

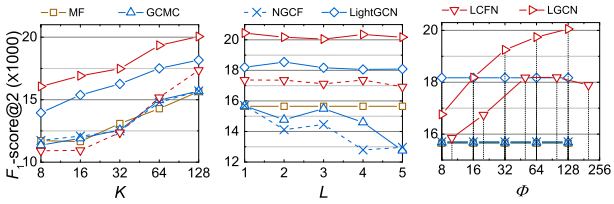
Effectiveness of 1D Low-pass Graph Convolution

There are two key novelties of our paper: (1) 1D low-pass graph convolution and (2) model structure. Since the experimental results of the model structure have been reported, in this section, we design experiments to validate the effectiveness of 1D low-pass graph convolution by comparing the three low-pass graph convolution methods. For fair comparison, they are all injected into the model structure introduced above and tuned by the same strategy.

- **LGCN_1D: LGCN** with **1D** graph convolution, which is the LGCN proposed in this paper.
- **LGCN_2D: LGCN** with **2D** graph convolution. User and item dimensions are defined by a user graph and an item graph. In the user graph, users are connected if they have interacted with the same item. The item graph is constructed in the same way.
- **LGCN_2Dh: LGCN** with **2D h**ypergraph convolution as Yu and Qin (2020) proposed.

Model	*Amazon*		*Movielens*	
	F_1-score@2	NDCG@2	F_1-score@2	NDCG@2
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
LightGCN	0.01703	0.02165	0.08162	0.26817
LCFN	0.01654	0.02104	0.08151	0.26129
LGCN	**0.01862**	**0.02346**	**0.08318**	**0.26917**
Improvement over competitive baselines				
MF	37.33%	34.65%	10.82%	7.24%
LightGCN	9.38%	8.35%	1.91%	0.38%
LCFN	12.57%	11.50%	2.05%	3.02%

Table 9: Recommendation performance (testing set)



(a) Impact of K (b) Impact of L (c) Impact of Φ

Figure 3: Model tuning (*Amazon*, validation set).

Metrics	LGCN_1D	LGCN_2D	LGCN_2Dh
F_1-score@2	**0.01862**	0.01449	0.01677
NDCG@2	**0.02346**	0.01834	0.02104

Table 10: Recommendation performance (*Amazon*, test set)

The results are reported in Table 10. Comparing LGCN_1D and LGCN_2D, we can see that a large amount of topological information is lost in 2D graph convolution. Thanks to the strong representation capacity, hypergraphs alleviate this issue in some degree, thus LGCN_2Dh outperforms LGCN_2D significantly, yet still performs worse than LGCN_1D. Since all topological information of the bipartite graph is utilized in the proposed 1D low-pass graph convolution, our LGCN_1D performs best.

Conclusion and Future Work

In this paper, we solve the two issues in the low-pass graph convolutional network: (1) Only 2-hop connections are used, which causes topological information loss. (2) The model structure used is rather suboptimal. For the issue (1), we design 1D low-pass graph convolution to utilize 1-hop connections directly. For the issue (2), we design experiments to explore the best structure for the proposed model. Comprehensive experiments show the effectiveness of the model.

For future work, we are interested in validate the effectiveness of our model in other graph data learning tasks. We also want to leverage more graph data in recommendation tasks, such as knowledge graphs and social networks. Some widely-used strategies in CNN also attract our interests.

References

Barrett, L. C.; and Wilde, C. 1960. A Power booktitle Development of the Convolution Theorem. *American Mathematical Monthly*, 893–894.

Berg, R. V. D.; Kipf, T. N.; and Welling, M. 2018. Graph Convolutional Matrix Completion. In *KDD*.

Defferrard, M.; Bresson, X.; and Vandergheynst, P. 2016. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. In *NIPS*, 3837–3845.

Duchi, J. C.; Hazan, E.; and Singer, Y. 2010. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. In *COLT*, 257–269.

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

Hamilton, W.; Ying, Z.; and Leskovec, J. 2017. Inductive Representation Learning on Large Graphs. In *NIPS*, 1024–1034.

He, X.; and Chua, T.-S. 2017. Neural Factorization Machines for Sparse Predictive Analytics. In *SIGIR*, 355–364.

He, X.; Deng, K.; Wang, X.; Li, Y.; Zhang, Y.; and Wang, M. 2020. LightGCN: Simplifying and Powering Graph Convolution Network for Recommendation. In *SIGIR*, 639–648.

He, X.; Liao, L.; Zhang, H.; Nie, L.; Hu, X.; and Chua, T.-S. 2017. Neural Collaborative Filtering. In *WWW*, 173–182.

Kingma, D. P.; and Ba, J. 2015. Adam: A Method for Stochastic Optimization. In *ICLR*.

Kipf, T. N.; and Welling, M. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR*.

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

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *NIPS*, 1106–1114.

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

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

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

Tieleman, T.; and Hinton, G. 2012. Lecture 6.5-RMSProp: Divide the gradient by a running average of its recent magnitude. In *COURSERA: Neural Networks for Machine Learning*.

Wang, X.; He, X.; Wang, M.; Feng, F.; and Chua, T.-S. 2019. Neural Graph Collaborative Filtering. In *SIGIR*.

Wu, F.; Souza, A.; Zhang, T.; Fifty, C.; Yu, T.; and Weinberger, K. 2019a. Simplifying graph convolutional networks. In *ICML*, 6861–6871.

Wu, L.; Sun, P.; Hong, R.; Fu, Y.; Wang, X.; and Wang, M. 2019b. SocialGCN: An Efficient Graph Convolutional Network based Model for Social Recommendation. In *AAAI*.

Ying, R.; He, R.; Chen, K.; Eksombatchai, P.; Hamilton, W. L.; and Leskovec, J. 2018. Graph Convolutional Neural Networks for Web-Scale Recommender Systems. In *KDD*, 974–983.

Yu, W.; and Qin, Z. 2020. Graph Convolutional Network for Recommendation with Low-pass Collaborative Filters. In *ICML*, 10936–10945.

Zhou, J.; Cui, G.; Zhang, Z.; Yang, C.; Liu, Z.; and Sun, M. 2018. Graph Neural Networks: A Review of Methods and Applications. *CoRR*.