Low-pass Graph Convolutional Network for Recommendation

Wenhui Yu, Zixin Zhang, Zheng Qin

1Alibaba Group, 2Tsinghua University
jianlin.ywh@alibaba-inc.com, zxx20@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, thus the structure 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.

Introduction
Convolutional Neural Networks (CNNs) 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 Graph Convolutional Networks (GCNs) (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 Low-pass Collaborative Filter (LCF). Graph convolution is defined in the frequency domain (Shuman et al. 2013) and all-frequency components of the signal participate in the 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 Network (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. What’s more, since the feature maps are large and dense matrices, they are factorized in LCFN for storage, making 2D convolution of interaction matrix degenerate 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 following 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 of the model. To solve these issues, 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 Low-pass Graph Convolutional Network (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.
• 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

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 Neural Network (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.

### 1D Low-pass Graph Convolution

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

Figure 1 shows the comparison of 2D and 1D low-pass graph convolution. We use dotted lines to indicate hyper-edges 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 \( E \) proposed in this paper. Although \( E \) is a 2D matrix, it is indeed 1D graph data since only one dimension of \( E \) is in the graph space and the other dimension is in the Euclidean space. 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 \( e = E_{ek} \) 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:

\[
F(\omega) = \int_{\mathbb{R}} f(t) e^{-j\omega t} \, dt = \langle f(t), e^{-j\omega t} \rangle, \tag{1}
\]

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 sig-
nal on the frequency, \( e^{-j\omega t} \) is the base of the frequency domain, where \( j \) is the imaginary unit and \( \omega \) is the angular frequency. \( e^{-j\omega t} \) satisfies: \( \nabla t e^{-j\omega t} = -j\omega e^{-j\omega t} \) 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:
\[
L = I - D^{-\frac{1}{2}}AD^{-\frac{1}{2}},
\]
where \( A = \begin{bmatrix} 0 & R \\ R^T & 0 \end{bmatrix} \) is adjacency matrix of the bipartite graph and \( D \) is the diagonal degree matrix: \( D_{ii} = \sum_j A_{ij} \). For graph data \( e \) (i.e., \( E_{\phi,k} \)), the result of Laplacian matrix performing on it is:
\[
(L e)_i = \frac{1}{\sqrt{D_{ii}}} \sum_{j \in N_i} \left( \frac{e_i}{\sqrt{D_{ii}}} - \frac{e_j}{\sqrt{D_{jj}}} \right)
\]
where \( N_i \) is the set of connected nodes of \( i \). Equation (2) shows that the effect of Laplacian matrix \( L \) 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: \( L = P \Lambda P^T \), where \( P \) is the eigen-vector matrix. Columns of \( P \) can form a set of normalized orthogonal bases, which defines the frequency domain. \( \Lambda = \text{diag}(\lambda_1, \ldots, \lambda_{M+N}) \) 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 \( e \) and the bases \( \{P_{\phi} \}_{\phi=1, \ldots, M+N} \):
\[
\tilde{e}_\phi = \langle e, P_{\phi} \rangle = \sum_{i=1}^{M+N} e_i P_{i,\phi}.
\]
For each column of \( E \), we calculate the inner product. The graph Fourier transform for entire embeddings \( E \) written in a closed form is: \( \tilde{E} = F_g(E) = P^T E \), where \( \tilde{E} \in \mathbb{R}^{(M+N) \times K} \) is the frequency-domain waveform of \( E \). In this paper, we use the tilde to mark the signal in frequency domain. Similarly, we can get the inverse graph Fourier transform: \( E = F_g^{-1}(\tilde{E}) = \tilde{P} \tilde{E} \), where \( F_g(\cdot) \) and \( F_g^{-1}(\cdot) \) 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: \( [\lambda_1, \ldots, \lambda_{M+N}] \). The \( \phi \)-th row \( \tilde{E}_\phi \) is the component of the signal \( E \) at the frequency \( \lambda_\phi \). Thus, \( \tilde{E}_\phi \) can be denoted as \( \tilde{E}(\lambda_\phi) \).

**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: \( \tilde{f} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \), where \( \tilde{f} \in \{0, 1\}^{M+N} \), \( 1 \in \{1\}^\phi \) is an all-one column vector; \( 0 \in \{0\}^{M+N-\phi} \) is an all-zero column vector; and \( \phi \) (strictly speaking, \( \lambda_\phi \)) is the passband cutoff frequency. To remove the high-frequency components, we need to transform \( e \) into the frequency domain, and conduct element-wise product with the filter \( \tilde{f} \), and finally transform it back to the time domain: \( LCF(e) = F_g^{-1}(F_g(e) \odot \tilde{f}) \), where LCF stands for low-pass graph (collaborative) filter. 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 \( E \), we have:
\[
LCF(E) = F_g^{-1} \left( \text{diag}(\tilde{f}) \cdot F_g(E) \right) = \tilde{P} \tilde{P}^T E,
\]
where \( \tilde{P} = P_{\star,1:\phi} \) are the first \( \phi \) eigen-vectors. We can find that when \( \Phi = M + N \), the filter becomes an all-pass filter and do not change the embeddings \( E \). By tuning LCF with respect to \( \Phi \), 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: \( e *_g k = F_g^{-1} \left( F_g(e) \circ F_g(k) \right) \), where \( *_g \) represents graph convolution; column vector \( k \in \mathbb{R}^{M+N} \) is convolution kernel and \( k = F_g(k) \) is its frequency-domain waveform. In our model, we define and train the kernel directly in frequency domain. For embeddings \( E \), the convolution result is:
\[
E *_g k = F_g^{-1} \left( \text{diag}(\tilde{k}) \cdot F_g(E) \right)
= \tilde{P} \text{diag}(\tilde{k}) \tilde{P}^T E,
\]
In our graph convolutional neural network, the convolution kernel \( k \) 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:
\[
E *_{\tilde{g}} k = F_g^{-1} \left( \text{diag}(\tilde{f}) \cdot \text{diag}(\tilde{k}) \cdot F_g(E) \right)
= \tilde{P} \text{diag}(\tilde{k}) \tilde{P}^T E,
\]
where \( *_{\tilde{g}} \) indicates low-pass graph convolution; \( \tilde{P} = P_{\star,1:\phi} \) contains the first \( \phi \) eigen-vectors and \( k = k_{\lambda,1:\phi} \) 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 $\Phi$ eigen-vectors are required. The time cost of constructing $P$ is $O((M+N)^3)$. However, since $L$ is sparse, $P$ can be constructed efficiently by Lanczos algorithm (Grimes, Lewis, and Simon 1994) with $O(np^2)$ time, where $n$ is the number of non-zero elements in $L$. Experiments show that $\Phi \ll M + N$, so our low-pass graph convolution is much efficient than original graph convolution.

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.

**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 low-pass graph filter from the perspective of removing the high-frequency noise in the user-item interaction matrix. In this paper, we provide another perspective: smoothing embeddings. Of course, they are equivalent since filtering the interaction matrix can be converted into filtering embeddings. However, 1D LCF works better 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.

In graph learning task, a basic assumption is that connected nodes should show commonality, i.e., similarity between their embeddings, thus embeddings should be low-frequency (smooth) signals. However, since user can only browse a small part of items (insufficient exposure), a large proportion of links are missed, causing unsmoothness of embeddings. For example, assuming nodes $i$ and $j$ are adjacent in the graph space, i.e., with a direct connections and may with several multi-hop connections. If the direct link between them is missed in the observed graph, the pair $(i, j)$ will be sampled as a negative sample, and embeddings of them are trained to be different. However, $i$ and $j$ are still connected by strong multi-hop connections, therefore embeddings become unsmooth graph signal. To address this issue, 1D LCF filters embeddings to smooth them. As we can see, exposure noise in the interaction matrix (Yu and Qin 2020) can be removed by 1D LCF in embedding level with no connection information loss. For the same reason, quantization noise can also be removed by 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 $D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$ is used for propagation. Embeddings after 1-hop propagation is $D^{-\frac{1}{2}}AD^{-\frac{1}{2}}E$. 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 $\frac{1}{\sqrt{D_{ii}D_{jj}}}$, 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 $P\bar{P}^T$. The similarity of $i$ and $j$ is determined by the inner product of the vector: $(P\bar{P}^T)_{ij} = P_iP_j^T$. Ng, Jordan, and Weiss (2002) pointed out that $P$ encodes the graph structure and can be used to measure the similarity between nodes in node clustering. $P\bar{P}^T$ is a similarity metric that takes multi-hop connections into account. Compared to the normalized adjacency matrix, $P\bar{P}^T$ 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 $(P\bar{P}^T)_{ij}$ 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:

$$D^{-\frac{1}{2}}AD^{-\frac{1}{2}}E = (I - L)E = P(I - A)P^TE = F_g^{-1} \cdot (diag([1 - \lambda_1, \cdots, 1 - \lambda_{M+N}]) \cdot F_g(E)) \quad (6)$$

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 $\lambda_1$ is attenuated by $1 - \lambda_1$. 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 $\lambda_1$ is 0) is all preserved, and with the increasing of the frequency
λ_0, the attenuation coefficient |1 - λ_0| decreases first and then increases (noting that a signal attenuated by a negative attenuation coefficient 1 - λ_0 equals to attenuated by |1 - λ_0| 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_0 is determined by estimating the frequency range of the signal and noise. By separating the signal and noise with λ_0, the signal can be completely reserved and the noise can be completely removed.

In long-distance propagation, we face over-smooth issue. It is easy to explain in the frequency domain. After k-hop propagation, we have: \( \left(D^{-\frac{1}{2}}AD^{-\frac{1}{2}}\right)E = \mathcal{F}_g^{-1}(\text{diag}((1-\lambda_1)^k, \ldots, (1-\lambda_{M+N})^k)) \cdot \mathcal{F}_g(E) \). We can see that the filter becomes \( (1-\lambda_1)^k, \ldots, (1-\lambda_{M+N})^k \), 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 \( \hat{f} \circ \cdots \circ \hat{f} = \hat{f} \) and filtered by LCF k times equals filtered by LCF one time: \( LCF(\cdots LCF(E)) = (P^T)^k E = \bar{P}(P^T P)^{-1} P^T E = \bar{P}^{-1} P^T E = P^T E = LCF(E) \).

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 E with kernel k to obtain the feature map \( \bar{E}' \).

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.

<table>
<thead>
<tr>
<th>Activation</th>
<th>–</th>
<th>Sigmoid</th>
<th>ReLU</th>
<th>Tanh</th>
</tr>
</thead>
<tbody>
<tr>
<td>( F_1)-score@2</td>
<td>0.01995</td>
<td>0.01845</td>
<td>0.01911</td>
<td>0.01977</td>
</tr>
</tbody>
</table>

Table 2: Effectiveness of different activation functions
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 \( E^{(l-1)} \) is input in current layer and we get the new feature maps \( E^{(l)} \):

\[ E^{(l)} = \bar{P} \text{diag}(k^{(l)}) P^T E^{(l-1)}. \]

We can get \( L + 1 \) embeddings \( \{E^{(l)}\}_{l=0, \ldots, L} \), where \( E^{(0)} \) 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 pooling (He et al. 2020), element-wise product, max pooling, and Multi-Layer Perceptron (MLP). Moreover, we try 1-layer and 3-layer MLP, called MLP_1L and MLP_3L respectively. Table 3 shows that the performance of sum pooling is the best, so our model adopts sum pooling.

<table>
<thead>
<tr>
<th>Pooling</th>
<th>( F_1)-score@2</th>
<th>Pooling</th>
<th>( F_1)-score@2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Concatenation</td>
<td>0.01854</td>
<td>Sum</td>
<td>0.01995</td>
</tr>
<tr>
<td>Product</td>
<td>0.01775</td>
<td>Maximum</td>
<td>0.01741</td>
</tr>
<tr>
<td>MLP_1L</td>
<td>0.01785</td>
<td>MLP_3L</td>
<td>0.01524</td>
</tr>
</tbody>
</table>

Table 3: Effectiveness of different pooling methods

Predictive Function After getting the predictive embeddings \( E \), we split it into user embeddings \( U = E_{1:M} \) and item embeddings \( V = E_{M+1:M+N} \). For a pair of user \( u \) and item \( i \), the preference score \( \hat{R}_{ui} \) is predicted based on \( U_u \) and \( V_i \). Commonly-used predictive functions are inner product and MLP. For inner product, \( \hat{R}_{ui} = U_u V_i^T \), and for MLP, \( \hat{R}_{ui} = MLP([U_u, V_i, U_u \odot V_i]) \). Table 4 shows that

<table>
<thead>
<tr>
<th>With transformation</th>
<th>Without</th>
</tr>
</thead>
<tbody>
<tr>
<td>( F_1)-score@2</td>
<td>0.01975</td>
</tr>
</tbody>
</table>

Table 1: Effectiveness of transformation
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.

<table>
<thead>
<tr>
<th>Functions</th>
<th>Inner product</th>
<th>MLP, 1L</th>
<th>MLP, 3L</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F_1$-score@2</td>
<td>0.01995</td>
<td>0.01910</td>
<td>0.01616</td>
</tr>
</tbody>
</table>

Table 4: Effectiveness of different predictive functions

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: Bayesian Personalized Ranking (BPR) (Rendle et al. 2009), cross-entropy, and Mean Square Error (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.

<table>
<thead>
<tr>
<th>Loss functions</th>
<th>BPR</th>
<th>Cross-entropy</th>
<th>MSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F_1$-score@2</td>
<td>0.01995</td>
<td>0.01982</td>
<td>0.01795</td>
</tr>
</tbody>
</table>

Table 5: Effectiveness of different loss functions

Sample Rate The impact of sampling rate $\rho$ is shown in Table 6. As we can see, the model performs best when $\rho = 3$. By tuning the model with respect to $\rho$, 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 $\rho$ to get the best performance.

<table>
<thead>
<tr>
<th>$\rho$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F_1$-score@2</td>
<td>0.01995</td>
<td>0.02018</td>
<td>0.02027</td>
<td>0.02003</td>
<td>0.02011</td>
</tr>
</tbody>
</table>

Table 6: Performances with different sampling rates $\rho$

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.

<table>
<thead>
<tr>
<th>Generalization</th>
<th>$F_1$-score@2</th>
<th>Generalization</th>
<th>$F_1$-score@2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$reg_1$</td>
<td>0.01995</td>
<td>$reg_2$</td>
<td>0.01976</td>
</tr>
<tr>
<td>Dropout</td>
<td>0.01919</td>
<td>$reg_1$ +dropout</td>
<td>0.01983</td>
</tr>
<tr>
<td>Normalization</td>
<td>0.01528</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 7: Effectiveness of different generalization methods

Optimizer We explore four optimizers: Stochastic Gradient Descent (SGD), Adaptive Gradient Algorithm (AdaGrad) (Duchi, Hazan, and Singer 2010), Root Mean Square Propagation (RMSProp) (Tieleman and Hinton 2012) and Adaptive Moment Estimation (Adam) (Kingma and Ba 2015). Table 8 shows that optimizers with momentum perform better than vanilla SGD and RMSProp.

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).

<table>
<thead>
<tr>
<th>Optimizer</th>
<th>SGD</th>
<th>AdaGrad</th>
<th>RMSProp</th>
<th>Adam</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F_1$-score@2</td>
<td>0.01938</td>
<td>0.01989</td>
<td>0.02005</td>
<td>0.01995</td>
</tr>
</tbody>
</table>

Table 8: Effectiveness of different optimizer

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) or Appendix.

Performance of LGCN

<table>
<thead>
<tr>
<th>Model</th>
<th>Amazon $F_1$-score@2</th>
<th>NDCG@2</th>
<th>Movielens $F_1$-score@2</th>
<th>NDCG@2</th>
</tr>
</thead>
<tbody>
<tr>
<td>MF</td>
<td>0.013256</td>
<td>0.01742</td>
<td>0.07506</td>
<td>0.25100</td>
</tr>
<tr>
<td>GCMD</td>
<td>0.01377</td>
<td>0.01742</td>
<td>0.07689</td>
<td>0.26316</td>
</tr>
<tr>
<td>NGCF</td>
<td>0.01349</td>
<td>0.01733</td>
<td>0.07453</td>
<td>0.25436</td>
</tr>
<tr>
<td>LightGCN</td>
<td>0.01703</td>
<td>0.02165</td>
<td>0.08162</td>
<td>0.26817</td>
</tr>
<tr>
<td>LCFN</td>
<td>0.01654</td>
<td>0.02104</td>
<td>0.08151</td>
<td>0.26129</td>
</tr>
<tr>
<td>LGCN</td>
<td>0.01862</td>
<td>0.02346</td>
<td>0.08318</td>
<td>0.26917</td>
</tr>
</tbody>
</table>

Improvement over competitive baselines

<table>
<thead>
<tr>
<th>Model</th>
<th>Amazon $F_1$-score@2</th>
<th>NDCG@2</th>
<th>Movielens $F_1$-score@2</th>
<th>NDCG@2</th>
</tr>
</thead>
<tbody>
<tr>
<td>MF</td>
<td>57.33%</td>
<td>34.65%</td>
<td>10.82%</td>
<td>7.24%</td>
</tr>
<tr>
<td>LightGCN</td>
<td>9.38%</td>
<td>8.35%</td>
<td>1.91%</td>
<td>0.38%</td>
</tr>
<tr>
<td>LCFN</td>
<td>12.57%</td>
<td>11.50%</td>
<td>2.05%</td>
<td>3.02%</td>
</tr>
</tbody>
</table>

Table 9: Recommendation performance (testing set)

Considering longer predictive embeddings mean stronger representation ability and large 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. 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.

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 introduced in Experimental Setup. Due to the limited space, we only report experiments 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 \( \eta \) and \( \mu \) 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 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: \( \mathbf{E}_{\Phi} \mathbf{k}^{(1)} = \mathbf{E}_{\Phi} \mathbf{k}^{(L)} \), where \( \mathbf{k} = \mathbf{k}^{(1)} = \cdots = \mathbf{k}^{(L)} \). 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 \( \Phi \). When \( \Phi \) 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 hypergraph convolution as Yu and Qin (2020) proposed.

<table>
<thead>
<tr>
<th>Metrics</th>
<th>LGCN(_{1D})</th>
<th>LGCN(_{2D})</th>
<th>LGCN(_{2Dh})</th>
</tr>
</thead>
<tbody>
<tr>
<td>F(_1)-score@2</td>
<td>0.01862</td>
<td>0.01449</td>
<td>0.01677</td>
</tr>
<tr>
<td>NDCG@2</td>
<td>0.02346</td>
<td>0.01834</td>
<td>0.02104</td>
</tr>
</tbody>
</table>

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 to 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


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


