Embedding Disentanglement in Graph Convolutional Networks for Recommendation

Tianyu Zhu, Leilei Sun, and Guoqing Chen

Abstract—Recent years have witnessed the rapid development of recommender systems. To improve recommendation performance, many efforts have been made to study how to equip the conventional methods with auxiliary information such as item relations. Meanwhile, a growing body of work has focused on applying graph convolutional networks to recommendation tasks. Thus, it is promising to use graph convolution to model multi-order relations among users and items for improving recommendation performance. However, the existing graph convolution-based recommendation methods may suffer from structural design problems: for methods with embedding transformations in graph convolutional layers, the MLP makes the updated embedding dimensions coupled, hurting the embedding expressivity. While for methods based on simplified graph convolution, removing the parameter matrices makes the model attach the same weight to embeddings in different layers, which may be over-simplified and limit the model expressivity. In this paper, we propose a novel graph convolution-based recommendation method, namely Channel-Independent Graph Convolutional Network (CIGCN). To learn disentangled embeddings, CIGCN uses diagonal parameter matrices as filters in graph convolution, keeping the updated embedding dimensions independent. In addition, with layer-aggregation strategies, the parameters in the diagonal matrices act as trainable weights that attach different importance to the embeddings in each layer and each dimension, enhancing the model expressivity. Results of extensive experiments on four real-world datasets show that CIGCN significantly outperforms baseline methods in recommendation accuracy and could learn better representations for users and items.

Index Terms—Recommender systems, graph convolutional networks, collaborative filtering, item relations

1 Introduction

Recently, recommender systems have become a core technology for online services, such as e-commerce platforms and social media [1], [2], [3]. It has been widely deployed to perform personalized information filtering and predict user preferences according to their historical actions. Many dot-com companies have benefited from recommender systems. For instance, it was reported that in Amazon, 35 percent of product sales came from recommendations [4]. In Netflix, about 80 percent of the content watched by subscribers was suggested by its recommender system [5].

However, recommendation methods often suffer from data sparsity and cold-start problems. To enhance recommendation performance, item relations have been considered in recent efforts [6], [7], [8]. These relations include rich semantic information, e.g., the "also-view" relations of products in e-commerce platforms, or the geographic relations of Point-Of-Interests (POIs) in location-based social platforms. Introducing these item relations into recommendation methods is considered meaningful and necessary for better representation learning for items, hence improving recommendation accuracy [6], [8].

The ways to introduce item relations into recommendation methods have two schools of thought: multi-task learning and graph convolution. For example, Collaborative Knowledge Base Embedding (CKE) [7] and Matrix Co-Factorization (MCF) [6] are factorization-based methods designed under the multi-task learning framework to regularize the item representations with item relation modeling. Meanwhile, a growing body of work focuses on applying graph convolution to recommendation tasks to explicitly model the multi-order relations among users and items [9], [11]. We can easily adapt these methods for item relation-aware recommendation by incorporating item relations into the embedding propagation matrix of graph convolution.

According to the parameter matrices used in graph convolution, these methods fall within two categories: methods with full parameter matrices in graph convolutional layers, and methods with no parameters in graph convolutional layers. For the former group of methods, such as GCMC [12], SpectralCF [13], and NGCF [11], the form of graph convolution in these methods follows the structure of Graph Convolutional Network (GCN) [10] that contains a standard MLP in each layer for embedding transformation, while this operation would leave the updated embedding dimensions coupled. Considering the trainable ID embeddings of users and items in recommendation, this form of graph convolution may hurt the learning process of embeddings. Since each embedding dimension represents a latent feature in latent factor models, we hope to learn disentangled embeddings to enhance the model expressivity. However, the MLP layer would make the embedding dimensions highly correlated, which may weaken the expressivity of embeddings and also the recommendation performance. For the

• *Tianyu Zhu and Guoqing Chen are with the Department of Management Science and Engineering, Tsinghua University, Beijing 100084, China. E-mail: {zhuty.16, chengq}@sem.tsinghua.edu.cn.*
• *Leilei Sun is with the SKLSDE Lab, School of Computer Science, Beihang University, Beijing 100191, China. E-mail: leileisun@buaa.edu.cn.*

Manuscript received 9 Jan. 2020; revised 30 Mar. 2021; accepted 3 June 2021. Date of publication 9 June 2021; date of current version 7 Dec. 2022. (Corresponding author: Leilei Sun.) Recommended for acceptance by G. Koutrika. Digital Object Identifier no. 10.1109/TKDE.2021.3087791

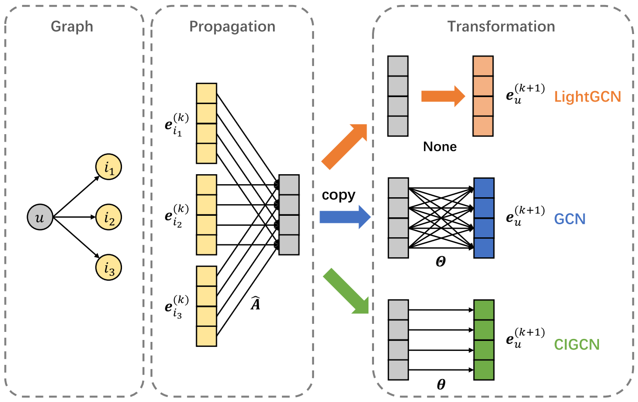


Fig. 1. An illustration of structural comparison among three graph convolution-based models. After embedding propagation, LightGCN [9] does not use any embedding transformations; GCN [10] uses an MLP for embedding transformation; our CIGCN adopts channel-independent embedding transformations.

latter group of methods, such as LR-GCCF [14] and LightGCN [9], the parameter matrices in graph convolutional layers are simply discarded, following the form of Simple Graph Convolution (SGC) [15]. Unfortunately, this form of graph convolution may be over-simplified and limit the model expressivity, since the importance of the embeddings in each layer may vary with the layers.

To address the above problems, we propose a novel graph convolution-based recommendation method, namely Channel-Independent Graph Convolutional Network (CIGCN). It adopts diagonal parameter matrices as filters in graph convolution and uses layer-aggregation strategies to obtain the final embeddings of users and items. The advantages of these designs are twofold. 1) The diagonal parameter matrices keep the embedding dimensions independent in graph convolutional layers (as shown in Fig. 1), making the model learn disentangled representations for users and items. 2) With layer-aggregation strategies, the parameters in the diagonal matrices act as trainable weights that attach different importance to the embeddings in each layer and each dimension, enhancing the model expressivity. We conduct comprehensive experiments on four public datasets, showing that CIGCN significantly outperforms the state-of-the-art baseline methods and could generate channel-independent embeddings for users and items.

We summarize our contributions as follows:

- A Channel-Independent Graph Convolutional Network (CIGCN) is provided. It is a novel graph convolution-based recommendation method with diagonal parameter matrices for disentangled embedding learning.
- A unifying framework for graph convolution-based recommendation methods is presented. We show that most of the existing methods could be covered by this framework, and we discuss the structural rationality of CIGCN by comparing it with other methods under this framework.
- Empirical studies are conducted on four public datasets to show the state-of-the-art performance of CIGCN and its effectiveness in learning better representations for users and items.

The rest of this paper is organized as follows. Section 2 introduces the preliminaries, Section 3 presents CIGCN in detail, Section 4 provides theoretical analyses on CIGCN, Section 5 evaluates the performances of CIGCN and the compared baselines, and Section 6 reviews the related work. Finally, Section 7 concludes this work with possible future directions.

2 Preliminaries

This section introduces the research problem and the graph convolution operation.

2.1 Problem Formulation

For recommendation with item relations, let $\mathcal{U} = \{u_1, u_2, \ldots, u_{|U|}\}$ be a set of users and $\mathcal{I} = \{i_1, i_2, \ldots, i_{|I|}\}$ be a set of items. Let $\mathbf{R} \in \mathbb{R}^{|U|\times|I|}$ be the binary user-item interaction matrix where $r_{ui} = 1$ if user u has an interaction with item i (e.g., click, purchase...), otherwise $r_{ui} = 0$. In addition, item relations are available, which are represented by an adjacency matrix $\mathbf{S} \in \mathbb{R}^{|I|\times|I|}$, where $s_{ij} = 1$ if item i is related to item j, otherwise $s_{ij} = 0$.

Given the user-item interaction matrix $\mathbf{R}$ and the item-item relation matrix $\mathbf{S}$, the recommendation problem aims at predicting whether user u has a potential interest in item i that he has never interacted with. Specifically, top-N items will be recommended to user u according to the predicted preference scores in descending order.

2.2 Graph Convolution

Since graph convolution is one of the essential components in our proposed method, we introduce it briefly here. Early work defined graph convolution in the Fourier domain. First, the eigendecomposition of the normalized graph Laplacian is computed [16]

$$\mathbf{L} = \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T, \tag{1}$$

where $\mathbf{I}$ is the identity matrix, $\mathbf{A}$ is the adjacency matrix of the graph, $\mathbf{D}$ is the degree matrix, $\mathbf{U}$ is the matrix of eigenvectors of $\mathbf{L}$, and $\mathbf{\Lambda}$ is the diagonal matrix of eigenvalues of $\mathbf{L}$. Then, graph convolution is defined as the multiplication of a signal $\mathbf{x} \in \mathbb{R}^N$ (N is the number of nodes in the graph) with a filter $\mathbf{g}_{\boldsymbol{\theta}} = \mathrm{diag}(\boldsymbol{\theta})$ parameterized by $\boldsymbol{\theta} \in \mathbb{R}^N$

$$\mathbf{g}_{\boldsymbol{\theta}} * \mathbf{x} = \mathbf{U}\mathbf{g}_{\boldsymbol{\theta}}\mathbf{U}^T\mathbf{x}. \tag{2}$$

Considering computing the eigendecomposition of $\mathbf{L}$ is costly, Defferrard *et al.* [17] proposed ChebNet, which approximates the filter $\mathbf{g}_{\boldsymbol{\theta}}$ by the Chebyshev polynomials of $\mathbf{\Lambda}$, i.e., $\mathbf{g}_{\boldsymbol{\theta}} = \sum_{k=0}^{K}\theta_k T_k(\mathbf{\Lambda})$, where $T(\cdot)$ is the Chebyshev polynomial, $\tilde{\mathbf{\Lambda}} = 2\mathbf{\Lambda}/\lambda_{\max} - \mathbf{I}$ and $\lambda_{\max}$ is the largest eigenvalue of $\mathbf{L}$. Then, graph convolution can be approximated as

$$\mathbf{g}_{\boldsymbol{\theta}} * \mathbf{x} \approx \sum_{k=0}^{K}\theta_k T_k(\tilde{\mathbf{L}})\mathbf{x}, \tag{3}$$

where $\tilde{\mathbf{L}} = 2\mathbf{L}/\lambda_{\max} - \mathbf{I}$. By using a Kth-order Chebyshev polynomial in the Laplacian matrix, features of the nodes that are at maximum K-step away from the central node are aggregated. The widely used first-order ChebNet can be written as

$$\mathbf{g}_{\boldsymbol{\theta}} * \mathbf{x} \approx (\theta_0 T_0(\tilde{\mathbf{L}}) + \theta_1 T_1(\tilde{\mathbf{L}}))\mathbf{x} = (\theta_0 \mathbf{I} + \theta_1 \tilde{\mathbf{L}})\mathbf{x}. \tag{4}$$

The eigenvalues of the normalized graph Laplacian lie between 0 and 2. If we approximate $\lambda_{\max} = 2$, the first-order ChebNet can be simplified to

$$\mathbf{g}_{\boldsymbol{\theta}} * \mathbf{x} \approx (\theta_0 \mathbf{I} - \theta_1 \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2})\mathbf{x}, \tag{5}$$

which is commonly used in graph convolution-based collaborative filtering methods.

3 Methodology

In this section, we propose a novel graph convolution-based recommendation method, namely Channel-Independent Graph Convolutional Network (CIGCN). We first describe the embedding propagation process of CIGCN, and then we introduce the layer aggregation strategies used for obtaining the final embeddings of users and items. Lastly, the loss function and the training process of CIGCN are presented.

3.1 Embedding Propagation

Recall that the first-order ChebNet has two free parameters: θ_0 for controlling how much information derives from the node itself, and θ_1 for controlling how much information comes from the node's first-order neighbors. In practice, we can constrain the number of parameters to address overfitting by setting $\theta = \theta_0 = -\theta_1$ in Equation (5) [10]. Then we can use a unified form to represent some variants of the first-order ChebNet as

$$\mathbf{x}' = \theta \hat{\mathbf{A}} \mathbf{x}, \tag{6}$$

where $\mathbf{x}'$ denotes the signal after graph convolution. The variants mainly differ in the propagation matrix $\hat{\mathbf{A}}$. For example, the first-order ChebNet uses the form $\hat{\mathbf{A}} = \mathbf{I} + \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$, the Graph Convolutional Network (GCN) [10] takes the renormalized form $\hat{\mathbf{A}} = (\mathbf{D}+\mathbf{I})^{-1/2}(\mathbf{A}+\mathbf{I})(\mathbf{D}+\mathbf{I})^{-1/2}$, and the Light Graph Convolutional Network (LightGCN) [9] uses the form without self-loops $\hat{\mathbf{A}} = \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$.

Additionally, for item relation-aware recommendation, we can construct a heterogeneous graph with both user-item interactions and item-item relations for embedding propagation as

$$\mathbf{A} = \begin{pmatrix} \mathbf{0} & \mathbf{R} \\ \mathbf{R}^T & \alpha \mathbf{S} \end{pmatrix}, \tag{7}$$

where $\mathbf{S}$ denotes the item-item adjacency matrix, and α is a hyperparameter that controls the weight of item relations. Here we set α to 1 as default and refer to the proposed method with this heterogeneous graph as CIGCN+.

Note that Equation (6) applies graph convolution to a single-channel feature.[1] To allow multi-channel features, a simple way is to use a unified filter parameter for all channels as

$$\mathbf{X}' = \theta \hat{\mathbf{A}} \mathbf{X}, \tag{8}$$

where $\mathbf{X} \in \mathbb{R}^{N \times d}$ denotes the feature matrix with d channels. However, such a strategy may limit the model expressivity since different feature dimensions may make different contributions for node representations. To cope with this problem, we can use channel-specific parameters for filters by generalizing θ to $\operatorname{diag}(\boldsymbol{\theta})$ as

$$\mathbf{X}' = \hat{\mathbf{A}} \mathbf{X} \operatorname{diag}(\boldsymbol{\theta}), \tag{9}$$

where $\boldsymbol{\theta} \in \mathbb{R}^d$ and $\operatorname{diag}(\boldsymbol{\theta}) = \operatorname{diag}(\theta_1, \ldots, \theta_d)$.

To obtain multi-order convolutional functions, we can stack multiple convolutional layers. Let $\mathbf{E}^{(k)}$ be the user/item embedding on the kth layer and $\mathbf{E}^{(0)} = \mathbf{X}$ denotes the trainable ID embedding, we have

$$\mathbf{E}^{(k)} = \hat{\mathbf{A}} \mathbf{E}^{(k-1)} \operatorname{diag}(\boldsymbol{\theta}). \tag{10}$$

Moreover, since the embeddings in different layers may have different contributions, we use layer-specific parameters for θ. The nonlinear activation functions are also used to further enhance the model expressivity. Therefore, the formulation of our proposed CIGCN is

$$\mathbf{E}^{(k)} = \sigma(\hat{\mathbf{A}} \mathbf{E}^{(k-1)} \operatorname{diag}(\boldsymbol{\theta}^{(k)})), \tag{11}$$

where $\sigma(\cdot)$ is the activation function, such as $\tanh(\cdot)$.

It is worth mentioning that different from the structure of GCN, CIGCN uses the diagonal parameter matrices as filters in graph convolution. Such a strategy enables the model to learn channel-independent embeddings for users and items. We will show the advantages of this design by comparing it with other graph convolutional structures in the next section.

3.2 Layer Aggregation

After performing graph convolution for K layers, we adopt layer-aggregation strategies (also called the Jumping Knowledge Network [18]) to combine the embeddings in each layer for obtaining the final embeddings of users and items [9], [11]

$$\mathbf{E} = \operatorname{AGG}(\mathbf{E}^{(0)}, \mathbf{E}^{(1)}, \ldots, \mathbf{E}^{(K)}), \tag{12}$$

where $\operatorname{AGG}(\cdot)$ denotes a layer-aggregation function. The advantages of using layer aggregations to obtain the final embeddings are twofold. First, with the increase of the graph convolutional layers, the embeddings tend to be over-smoothed [19], [20], which may hurt recommendation performance. Layer aggregation strategies combine the embeddings in different layers to alleviate the over-smoothing problem. Second, since the embeddings in different layers focus on messages passed over different receptive fields, they could enrich the varieties of representations by aggregating multi-level semantics.

Various layer-aggregation functions are optional here, e.g., the sum-pooling [21]

$$\mathbf{E} = \sum_{k=0}^{K} \mathbf{E}^{(k)}, \tag{13}$$

1. Here we follow the literature and use the term "channel" to refer to the feature (embedding) dimension.

or the concatenation [11]

$$\mathbf{E} = [\mathbf{E}^{(0)}, \mathbf{E}^{(1)}, \ldots, \mathbf{E}^{(K)}]. \tag{14}$$

The advantage of concatenation is that the information loss is minimized, while it would lead to a large-dimensional feature space when stacking GCN layers, resulting in high computation cost and slow convergence. In contrast, sum-pooling could keep the embedding size unchanged, but it may lose some information during aggregation.

3.3 Model Training

The predicted preference score of user u to item i is defined as the dot product of their final embeddings, i.e.,

$$\hat{r}_{ui} = \mathbf{e}_u^T \mathbf{e}_i, \tag{15}$$

which is also used as the ranking score for generating recommendations.

To learn a top-N recommendation model, the pairwise ranking loss is adopted [22], which ranks the positive items (items that the user has interacted with) higher than the negative ones (items that the user has not interacted with)

$$\mathcal{L} = -\sum_{(u,i,j)\in\mathcal{D}} \log \sigma(\hat{r}_{ui} - \hat{r}_{uj}) + \lambda ||\Theta||^2, \tag{16}$$

where $\sigma(\cdot)$ denotes the *sigmoid* function. The hyperparameter λ controls the strength of l_2 regularization to prevent overfitting, and $\Theta = \{\mathbf{E}^{(0)}, \boldsymbol{\theta}^{(1)}, \ldots, \boldsymbol{\theta}^{(K)}\}$ denotes the set of trainable parameters. We use $\mathcal{D}$ to represent the set of training triples, where

$$\mathcal{D} = \{(u,i,j) | u \in \mathcal{U} \wedge i \in \mathcal{R}_u^+ \wedge j \in \mathcal{I} \setminus \mathcal{R}_u^+\}, \tag{17}$$

where $\mathcal{R}_u^+$ denotes the set of positive items. By minimizing the loss in Equation (16), the relative orders between items for each user can be predicted, which is tailored for the personalized ranking task.

We adopt the mini-batch *Adaptive Moment Estimation* (Adam) [23] to optimize the loss function, which is a variant of *Stochastic Gradient Descent* (SGD) with adaptive moment estimation. Specifically, in each epoch, all the observed interactions are shuffled first. Then, for each observed user-item interaction, a negative item is randomly sampled from a uniform distribution. After that, in each iteration, a mini-batch of training triples is put into the model and the gradient of the loss function is calculated for each trainable parameter. The parameters are updated according to the Adam algorithm until convergence.

Note that we also use the node dropout strategy [12] for CIGCN in the training process to prevent overfitting, which randomly blocks part of the nodes and discards their outgoing messages to obtain robust embeddings.

4 Theoretical Analyses

In this section, we conduct theoretical analyses to demonstrate the motivation behind the design of CIGCN. We first present a unified framework to summarize the existing graph convolution-based recommendation methods. We then compare the structure of CIGCN with two typical graph convolution architectures, i.e., the Graph Convolutional Network (GCN) [10] and the Light Graph Convolutional Network (LightGCN) [9], to present the advantages of CIGCN for recommendation tasks. Finally, the model size of CIGCN is analyzed and compared with other methods.

TABLE 1
Comparisons Among Several Graph Convolution-Based Models in Their Convolutional Forms

Model	Convolutional Form	θ_0	θ_1
First-order ChebNet	$(\theta_0\mathbf{I} - \theta_1\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2})\mathbf{x}$	-	-
GCMC	$\theta(\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2})\mathbf{x}$	0	$-\theta$
SpectralCF[2]	$\theta(2\mathbf{I} - \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2})\mathbf{x}$	2θ	θ
NGCF-SVD++	$\theta(\mathbf{I} + \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2})\mathbf{x}$	θ	$-\theta$
LightGCN	$(\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2})\mathbf{x}$	0	-1

4.1 A Unified Framework for Graph Convolution-Based Models

Most of the existing graph convolution-based recommendation methods are built upon the first-order ChebNet. We use a unified form to summarize their graph convolutional structures, as shown in Table 1.

4.1.1 GCMC

Graph Convolutional Matrix Completion (GCMC) [12] is a graph autoencoder for rating prediction. The message passing process of GCMC is

$$\boldsymbol{\mu}_{j\to i,r} = \frac{1}{c_{ij}} \mathbf{W}_r \mathbf{x}_j^v, \tag{18}$$

where $\boldsymbol{\mu}_{j\to i,r}$ denotes the messages from item j to user i of rating level r, c_{ij} is a normalization constant. $\mathbf{W}_r$ is the parameter matrix of rating level r and $\mathbf{x}_j^v$ is the feature vector of item j. Messages from users to items are processed in an analogous way. Then, the messages for each rating level are accumulated into a single vector representation and a dense layer is used to obtain the final embeddings of users and items.

To tailor GCMC for recommendation with implicit feedback, we can set $\mathbf{W}_r = \mathbf{I}$ and discard the message accumulation step. Then, the message passing process of GCMC can be written as

$$\mathbf{x}' = \theta(\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2})\mathbf{x}. \tag{19}$$

Thus, GCMC uses a first-order ChebNet with $\theta_0 = 0$ and $\theta_1 = -\theta$ for graph convolution.

4.1.2 SpectralCF

Spectral Collaborative Filtering (SpectralCF) [13] proposes a spectral convolutional operation to learn user and item embeddings as

$$\mathbf{x}' = \theta(\mathbf{U}\mathbf{U}^T + \mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^T)\mathbf{x}, \tag{20}$$

2. Here we consider a variant of SpectralCF that replaces the random walk normalized graph Laplacian with the symmetric normalized graph Laplacian.

where $\mathbf{U}$ is the matrix of eigenvectors of the random walk normalized graph Laplacian $\mathbf{L} = \mathbf{I} - \mathbf{D}^{-1}\mathbf{A}$, and $\mathbf{\Lambda}$ is the diagonal matrix of eigenvalues of $\mathbf{L}$.

Since SpectralCF needs to compute the eigendecomposition of $\mathbf{L}$, which is not applicable to large graphs, we can approximate it by replacing the random walk normalized graph Laplacian $\mathbf{L} = \mathbf{I} - \mathbf{D}^{-1}\mathbf{A}$ with the symmetric normalized graph Laplacian $\mathbf{L} = \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$. Then the eigenvectors become orthonormal and the graph convolution process can be simplified to

$$\mathbf{x}' = \theta(\mathbf{I} + \mathbf{L})\mathbf{x} = \theta(2\mathbf{I} - \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2})\mathbf{x}. \tag{21}$$

Hence, SpectralCF approximately uses a first-order ChebNet with $\theta_0 = 2\theta$ and $\theta_1 = \theta$ for graph convolution.

4.1.3 NGCF

Neural Graph Collaborative Filtering (NGCF) [11] propagates embeddings on the user-item interaction graph to model the multi-order connectivity for item recommendation. The embedding propagation rule of NGCF is

$$\begin{aligned}\mathbf{E}^{(k)} = \mathrm{LeakyReLU}((&\mathbf{I} + \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2})\mathbf{E}^{(k-1)}\mathbf{W}_1^{(k)} \\ &+ \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}\mathbf{E}^{(k-1)} \odot \mathbf{E}^{(k-1)}\mathbf{W}_2^{(k)}),\end{aligned} \tag{22}$$

where the latter term makes messages being propagated dependent on the affinity between the user embedding and the item embedding. For easy analysis, we consider a simplified variant of NGCF that omits the latter term in Equation (22), i.e., the NGCF-SVD++ model [11], which can be formulated as

$$\mathbf{E}^{(k)} = \mathrm{LeakyReLU}((\mathbf{I} + \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2})\mathbf{E}^{(k-1)}\mathbf{W}_1^{(k)}), \tag{23}$$

and its embedding propagation rule can be written as

$$\mathbf{x}' = \theta(\mathbf{I} + \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2})\mathbf{x}. \tag{24}$$

Thus, NGCF-SVD++ uses a first-order ChebNet with $\theta_0 = \theta$ and $\theta_1 = -\theta$ for graph convolution.

4.1.4 LightGCN

Light Graph Convolutional Network (LightGCN) [9] learns user and item embeddings by linearly propagation on the user-item interaction graph. Its embedding propagation process can be formulated as

$$\mathbf{E}^{(k)} = (\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2})\mathbf{E}^{(k-1)}, \tag{25}$$

which can be written in the vector form as

$$\mathbf{x}' = (\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2})\mathbf{x}. \tag{26}$$

Therefore, LightGCN uses a first-order ChebNet with $\theta_0 = 0$ and $\theta_1 = -1$ for graph convolution.

4.2 Comparison With GCN

Since the form of graph convolution in GCMC, SpectralCF, and NGCF are based on the Graph Convolutional Network (GCN) [10], here we compare the structure of CIGCN with that of GCN.

GCN updates the node representations in two steps: embedding propagation and embedding transformation. First, the embedding of each node is averaged with that in its local neighborhood. After the local smoothing, a standard MLP is used for embedding transformation

$$\mathbf{E}^{(k)} = \sigma(\hat{\mathbf{A}}\mathbf{E}^{(k-1)}\mathbf{\Theta}^{(k)}), \tag{27}$$

where $\mathbf{\Theta}^{(k)} \in \mathbb{R}^{d\times d}$ denotes the parameter matrix in the kth layer

$$\mathbf{\Theta}^{(k)} = \begin{pmatrix} \theta_{11}^{(k)} & \theta_{12}^{(k)} & \cdots & \theta_{1d}^{(k)} \\ \theta_{21}^{(k)} & \theta_{22}^{(k)} & \cdots & \theta_{2d}^{(k)} \\ \vdots & \vdots & \ddots & \vdots \\ \theta_{d1}^{(k)} & \theta_{d2}^{(k)} & \cdots & \theta_{dd}^{(k)} \end{pmatrix}.$$

With the MLP layer, the dimensions of the updated embeddings become no longer independent. Specifically, for the jth embedding dimension of node i, after calculating $\mathbf{X\Theta}$, $x'_{ij} = \theta_{1j}x_{i1} + \cdots + \theta_{dj}x_{id}$. Thus, the updated embedding dimensions become highly correlated.

Considering the trainable ID embeddings $\mathbf{E}^{(0)}$ of users and items in recommendation, this form of graph convolution may hurt the learning process of embeddings. Since each embedding dimension represents a latent feature in latent factor models, we hope to learn disentangled embeddings to enhance the model expressivity. However, the MLP layer would make the embedding dimensions coupled, which may weaken the expressivity of embeddings and also the recommendation performance.

To learn disentangled embeddings, CIGCN uses a diagonal parameter matrix in each graph convolutional layer, keeping the embedding dimensions independent. The diagonal parameter matrix can be formulated as

$$\mathrm{diag}(\theta^{(k)}) = \begin{pmatrix} \theta_1^{(k)} & 0 & \cdots & 0 \\ 0 & \theta_2^{(k)} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \theta_d^{(k)} \end{pmatrix}.$$

For the jth embedding dimension of node i, after calculating $\mathbf{X}\mathrm{diag}(\boldsymbol{\theta})$, $x'_{ij} = \theta_j x_{ij}$. That is, the updated embedding dimensions only deal with the information from their own dimensions, and the parameters are used to control how much information should be retained after embedding propagation. In this way, CIGCN can keep the embedding dimensions independent after graph convolution and learn disentangled embeddings for users and items, enhancing the model expressivity.

4.3 Comparison With LightGCN

Another category of graph convolution-based recommendation methods apply linear embedding propagation without embedding transformation, following the structure of Simple Graph Convolution (SGC) [15]. Here we take LightGCN [9] as an example and discuss its connection with CIGCN. If we omit the non-linear activation function in CIGCN, we can bridge the embedding in the kth layer of CIGCN with that of LightGCN as

$$\begin{aligned}\mathbf{E}^{(k)}_{CIGCN} &= \hat{\mathbf{A}}\mathbf{E}^{(k-1)}_{CIGCN}\mathrm{diag}(\boldsymbol{\theta}^{(k)}) \\ &= \hat{\mathbf{A}}^k\mathbf{E}^{(0)}_{CIGCN}\mathrm{diag}(\boldsymbol{\theta}^{(1)})\cdots\mathrm{diag}(\boldsymbol{\theta}^{(k)}) \\ &= \mathbf{E}^{(k)}_{LightGCN}\mathrm{diag}(\boldsymbol{\theta}^{(1)}\odot\cdots\odot\boldsymbol{\theta}^{(k)}).\end{aligned} \tag{28}$$

Thus, if we fix $\boldsymbol{\theta}^{(k)} = \mathbf{1}$ for $k = 1, \ldots, K$, CIGCN would degenerate to LightGCN.

Compared with LightGCN, CIGCN introduces channel-specific and layer-specific filter parameters in the embedding propagation process, which act as trainable weights that attach different importance to the embeddings in each layer and each dimension, enhancing the model expressivity.

4.4 Model Size Analysis

Here we analyze the model size of CIGCN by comparing it with other graph convolution-based recommendation methods. For the user and item ID embedding $\mathbf{E}^{(0)}$, it takes $(|U| + |I|)d$ parameters, where d denotes the embedding size. It is also the number of parameters of BPR and LightGCN, which have no extra parameters in the models. NGCF introduces two transformation matrices in each layer with $2d^2K$ more parameters, where K is the number of graph convolutional layers. CIGCN uses diagonal parameter matrices in each layer with dK more parameters. Therefore, the number of parameters of CIGCN is smaller than NGCF but slightly larger than BPR and LightGCN. However, CIGCN obtains significantly better performance than LightGCN with comparable training efficiency, which will be shown in our experiments.

5 Experiments

In this section, data experiments were conducted to show the superiority of CIGCN to baseline methods. The codes have been published on GitHub.[3] The experiments were designed to answer the following research questions:

RQ1. Does CIGCN outperform the state-of-the-art recommendation methods?

RQ1. How do the hyperparameters and components affect the performance of CIGCN?

RQ3. Can CIGCN learn the importance of the embeddings in different graph convolutional layers to recommendation?

RQ4. Can CIGCN learn channel-independent embeddings in graph convolutional layers?

5.1 Datasets

We evaluated CIGCN on four real-world datasets with different domains and sparsity. The statistics of the datasets are summarized in Table 2.

TABLE 2
Statistics of the Evaluation Datasets

Dataset	#users	#items	#actions	density	#relations
Amazon Beauty	39,507	53,076	346,515	0.017%	1,472,176
Amazon Automotive	17,373	34,175	131,500	0.022%	1,162,037
Yelp	32,206	27,671	726,154	0.081%	268,402
Google Local	16,381	34,928	427,910	0.075%	687,397

Amazon.[4] This is a product-review dataset crawled from *Amazon.com*, a well-known online shopping platform [24]. The entire data were split into several subsets according to the top-level categories. Here we adopted two categories, i.e., "Beauty" and "Automotive". The datasets were preprocessed by retaining items/users with at least 5/5 records and the presence of a review was treated as implicit feedback. The "also-view", "also-buy", "buy after viewing", and "buy together" data were used as item relations.

Yelp.[5] This dataset is a subset of the businesses, reviews, and user data in *Yelp.com*, a large location-based social platform. Here we used the review and business data in 2014-2016. The datasets were preprocessed by retaining items/users with at least 20/10 records and the presence of a review was treated as implicit feedback. The nearest at most 20 businesses in the same city were regarded as semantically related items for each business.

Google Local.[6] This is a POI-based dataset crawled from *Google Local*, which contains user reviews and POI data [25]. The datasets were preprocessed by retaining items/users with at least 20/10 records and the presence of a review was treated as implicit feedback. The nearest at most 20 POIs (truncated by a distance threshold) were treated as related items for each POI.

5.2 Experimental Settings

Evaluation Protocols. To evaluate the recommendation performances of the compared methods, we followed the leave-one-out scheme that leaves the last items and the next-to-last items in user interactions for test and validation, and uses the remaining items for training [8], [26]. For time-saving evaluation, we randomly sampled 999 items that have not been interacted by a user and ranked them along with the ground-truth item. The performance of a ranking list was judged by Hit Ratio (HR), Normalized Discounted Cumulative Gain (NDCG), and Mean Reciprocal Rank (MRR)

$$\mathrm{HR}@N = \frac{1}{|U|}\sum_{u\in U}\mathbf{1}(r_{u,g_u} \leq N), \tag{29}$$

$$\mathrm{NDCG}@N = \frac{1}{|U|}\sum_{u\in U}\frac{\mathbf{1}(r_{u,g_u} \leq N)}{\log_2(r_{u,g_u}+1)}, \tag{30}$$

$$\mathrm{MRR}@N = \frac{1}{|U|}\sum_{u\in U}\frac{\mathbf{1}(r_{u,g_u} \leq N)}{r_{u,g_u}}, \tag{31}$$

where g_u is the ground-truth item for user u, r_{u,g_u} is the rank generated by a recommendation method for user u and

3. http://github.com/zhuty16/CIGCN
4. http://jmcauley.ucsd.edu/data/amazon/index_2014.html
5. https://www.yelp.com/dataset/
6. https://cseweb.ucsd.edu/jmcauley/datasets.html

TABLE 3
Performance Comparison With Embedding Size 64

Model	Amazon Beauty			Amazon Automotive			Yelp			Google Local		
	HR	NDCG	MRR	HR	NDCG	MRR	HR	NDCG	MRR	HR	NDCG	MRR
MP	0.1058	0.0566	0.0418	0.1138	0.0649	0.0500	0.0868	0.0433	0.0303	0.0673	0.0326	0.0223
BPR	0.2341	0.1377	0.1086	0.1855	0.1100	0.0870	0.3484	0.1905	0.1429	0.5224	0.3093	0.2438
Mult-DAE	0.2439	0.1519	0.1241	0.1848	0.1121	0.0901	0.3910	0.2191	0.1671	0.5666	0.3520	0.2859
CKE	0.3136	0.1861	0.1475	0.2768	0.1643	0.1300	0.3705	0.2046	0.1545	0.5685	0.3432	0.2739
MCF	0.3196	0.1913	0.1519	0.2922	0.1738	0.1382	0.3736	0.2049	0.1543	0.5725	0.3458	0.2769
GCMC+	0.2230	0.1339	0.1071	0.1688	0.0973	0.0755	0.3483	0.1901	0.1426	0.5677	0.3485	0.2816
NGCF+	0.2438	0.1426	0.1117	0.1901	0.1102	0.0858	0.3461	0.1886	0.1410	0.5275	0.3103	0.2435
LR-GCCF+	0.2990	0.1745	0.1368	0.2484	0.1480	0.1175	0.3853	0.2128	0.1608	0.5875	0.3598	0.2894
LightGCN+	0.3423	0.2087	0.1678	0.2916	0.1753	0.1405	0.4069	0.2283	0.1742	0.6177	0.3876	0.3167
CIGCN+	**0.3652**	**0.2276**	**0.1856**	**0.3265**	**0.1999**	**0.1609**	**0.4149**	**0.2334**	**0.1788**	**0.6282**	**0.3949**	**0.3229**

Best results are in boldface.

item g_u, and $\mathbf{1}(\cdot)$ is an indicator function. Without special mention, the ranking list was truncated at 10 for all metrics. As such, HR@10 intuitively measures whether the ground-truth item is presented on the top-10 list, while NDCG@10 and MRR@10 account for the position of the hit by assigning higher scores to hits at the top ranks. The average scores of all users for three metrics were reported. Due to space limitation, we only show the performances of NDCG@10 in some figures.

Baselines. We compare the performance of GIGCN with those of three groups of baseline methods. The first group contains general recommendation methods that only leverage user-item interactions:

- *Most Popular (MP).* It is a non-personalized method that simply ranks the items according to their popularity.
- *Bayesian Personalized Ranking (BPR)* [22]. It is an MF-based method with a pairwise ranking loss to generate recommendations with implicit feedback.
- *Multinomial Denoising Autoencoder (Mult-DAE)* [27]. It is a state-of-the-art item-based recommendation method built upon denoising autoencoders with multinomial conditional likelihood.

The second group contains recommendation methods that leverage item relations by multi-task learning to improve recommendation performance:

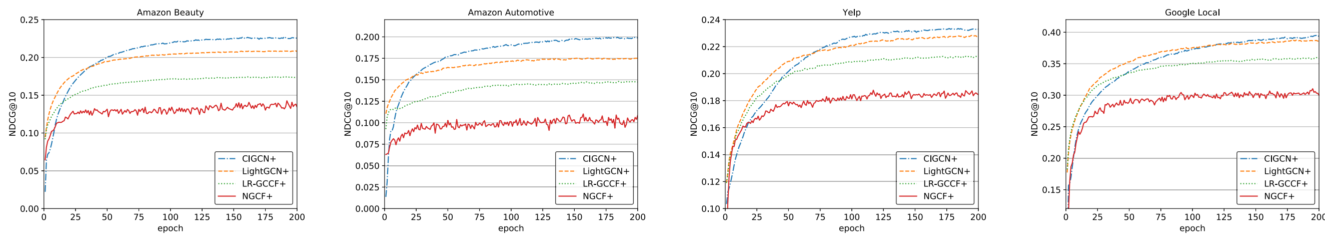
- *Collaborative Knowledge Base Embedding (CKE)*[7]. It is a factorization-based method that combines structural, textual, and visual information in an MF-based framework. Here only structural information is used for a fair comparison.
- *Matrix Co-Factorization (MCF)*[6]. It simultaneously factorizes the user-item rating matrix and the item-item relation matrix with shared item representations for rating prediction. Here the BPR loss is adopted for item recommendation.

The third group contains recommendation methods based on graph convolutions:

- *Graph Convolutional Matrix Completion (GCMC)* [12]. It is a GCN-based recommendation method that uses a one-layer graph convolution to construct user/item embedding through message passing on the user-item interaction graph.
- *Neural Graph Collaborative Filtering (NGCF)* [11]. It propagates embeddings on the user-item interaction graph to model the multi-order connectivity for item recommendation.
- *Linear Residual Graph Convolutional Collaborative Filtering (LR-GCCF)* [14]. It removes the non-linear transformations in GCN and proposes a linear embedding propagation strategy with a residual network structure for item recommendation.
- *Light Graph Convolutional Network (LightGCN)* [9]. It learns user/item embedding by linearly propagation on the user-item interaction graph and generates the final embeddings with a mean-pooling strategy for item recommendation.

For a fair comparison, we extend all the graph convolution-based methods by using an embedding propagation matrix with both user-item interactions and item-item relations like Equation (7), namely GCMC+, NGCF+, LR-GCCF+, and LightGCN+, respectively. We also tried SpectralCF [13], but found that the eigendecomposition in SpectralCF costs too much time, especially on large datasets [11]. Hence, we did not select it as a baseline method.

Hyperparameter Settings. We implemented all the model-based methods with TensorFlow. For common hyperparameters in all the model-based methods, the embedding size d was set to 64 with the random normal initializer (with a mean of 0 and standard deviation of 0.01), the l_2 regularization was tuned in {0, 1e-8, 1e-7, 1e-6, 5e-6, 1e-5, 5e-5, 1e-4}, and the learning rate was tuned in {5e-3, 2e-3, 1e-3, 5e-4, 2e-4, 1e-4, 5e-5}. All the model-based methods were trained with mini-batch Adam [28], in the batch size of 256. For graph convolution-based methods, we computed the graph convolution on the whole graph before the mini-batch loss, and we tuned the number of graph convolutional layers in $\{1, 2, 3, 4\}$. We reported the results of baseline methods under their optimal hyperparameter settings. For CIGCN+, we adopted the symmetric normalized adjacency matrix $\hat{\mathbf{A}} = \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$ as the propagation matrix and used the sum-pooling layer aggregation as default.



(a) Amazon Beauty (b) Amazon Automotive (c) Yelp (d) Google Local

Fig. 2. Test performances of the graph convolution-based methods in each epoch.

TABLE 4
Performance Comparison w.r.t. the Number of Graph Convolutional Layers

Depth	Model	Amazon Beauty			Amazon Automotive			Yelp			Google Local		
		HR	NDCG	MRR	HR	NDCG	MRR	HR	NDCG	MRR	HR	NDCG	MRR
1 Layer	LR-GCCF+	0.2746	0.1618	0.1276	0.2178	0.1318	0.1057	0.3765	0.2082	0.1574	0.5550	0.3355	0.2680
	LightGCN+	0.2884	0.1791	0.1454	0.2379	0.1453	0.1173	0.3835	0.2110	0.1590	0.5718	0.3478	0.2788
	CIGCN+	0.3430	0.2078	0.1665	0.3016	0.1833	0.1472	0.3993	0.2214	0.1673	0.6161	0.3764	0.3024
2 Layers	LR-GCCF+	0.2900	0.1704	0.1340	0.2374	0.1421	0.1131	0.3808	0.2114	0.1600	0.5763	0.3522	0.2833
	LightGCN+	0.3267	0.2021	0.1643	0.2720	0.1661	0.1337	0.3926	0.2198	0.1677	0.6045	0.3765	0.3073
	CIGCN+	0.3616	0.2244	0.1825	0.3211	0.1979	0.1602	**0.4149**	**0.2334**	**0.1788**	**0.6282**	**0.3949**	**0.3229**
3 Layers	LR-GCCF+	0.2969	0.1738	0.1364	0.2447	0.1463	0.1164	0.3839	0.2124	0.1605	0.5837	0.3568	0.2872
	LightGCN+	0.3383	0.2081	0.1682	0.2856	0.1734	0.1394	0.4026	0.2264	0.1728	0.6130	0.3824	0.3128
	CIGCN+	0.3643	0.2274	0.1855	**0.3285**	**0.2012**	**0.1626**	0.4131	0.2327	0.1781	0.6230	0.3899	0.3176
4 Layers	LR-GCCF+	0.2990	0.1745	0.1368	0.2484	0.1480	0.1175	0.3853	0.2128	0.1608	0.5875	0.3598	0.2894
	LightGCN+	0.3423	0.2087	0.1678	0.2916	0.1753	0.1405	0.4069	0.2283	0.1742	0.6177	0.3876	0.3167
	CIGCN+	**0.3652**	**0.2276**	**0.1856**	0.3265	0.1999	0.1609	0.4122	0.2325	0.1779	0.6227	0.3920	0.3208

Best results are in boldface.

5.3 Performance Comparison (RQ1)

Main Performance. The overall performances of all the compared methods are summarized in Table 3. We conclude the findings as follows:

First, item relations are beneficial to recommendation accuracy. The last two groups of methods significantly outperform the general recommendation methods owing to item relation modeling, which may result in better representation learning for users and items.

Second, for recommendation with item relations, the best graph convolution-based method (i.e., LightGCN+) achieves better results than the best multi-task learning-based method (i.e., MCF). In other words, the graph convolution-based methods that explicitly encode multi-order item relations may be more effective than the factorization-based methods.

Third, for graph convolution-based methods, GCMC+ and NGCF+, which include the MLP in graph convolutional layers, do not perform well especially on sparser datasets. Without extra parameters in graph convolutional layers, LightGCN+ and LR-GCCF+ achieve better results. Moreover, CIGCN+ that uses diagonal parameter matrices outperforms these baseline methods significantly, showing the rationality of its structural design.

Learning Curves. We also draw the learning curves (test performance of NDCG@10) of the graph convolution-based methods in Fig. 2. We have the following findings:

First, the performance of NGCF+ is the worst among the four methods and shows severe fluctuations during the training epochs. The full parameter matrices in graph convolutional layers of NGCF+ would make the training process unstable and hurt recommendation performance.

Second, CIGCN+ outperforms LightGCN+ and LR-GCCF+ on the four datasets significantly. Since the parameters in CIGCN+ act as trainable weights that attach different importance to the embeddings in each layer, CIGCN+ has stronger expressivity than LightGCN+ and LR-GCCF+, leading to better recommendation performance.

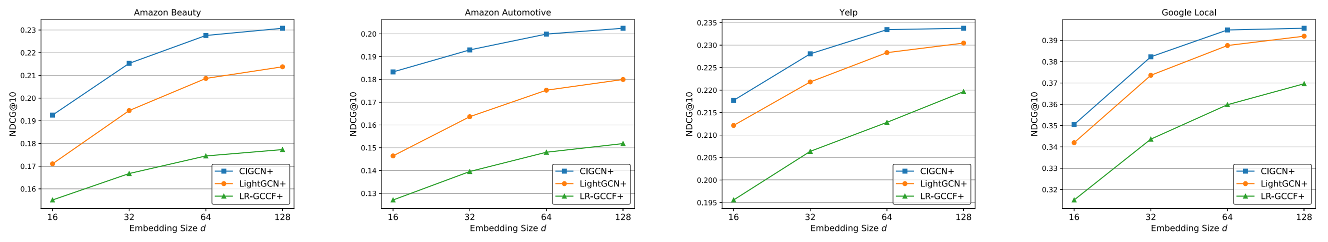
5.4 Hyperparameter and Ablation Studies (RQ2)

In this subsection, we first analyze the impact of the key hyperparameters for the graph convolution-based methods on recommendation performance. We then perform ablation studies on CIGCN+ to show how the different components affect its performance.

5.4.1 *Impact of Layer Number*

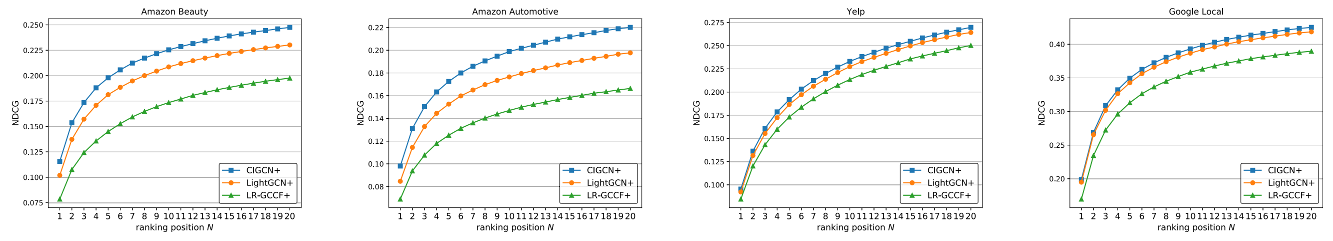
Here we study the impact of the number of graph convolutional layers of LR-GCCF+, LightGCN+, and CIGCN+ on recommendation performance. We report their performances with 1 to 4 layers in Table 4.

As can be seen, increasing the number of layers could improve the performances of LR-GCCF+ and LightGCN+ but the marginal benefits diminish. While for CIGCN+, the optimal number of layers could be 2 or 3. Since the filter parameters could enable CIGCN+ to learn properly smoothed embeddings by attaching different weights to different embedding layers, a shallow CIGCN+ could already



(a) Amazon Beauty (b) Amazon Automotive (c) Yelp (d) Google Local

Fig. 3. Performances of LR-GCCF+, LightGCN+ and CIGCN+ w.r.t. the embedding size d.



(a) Amazon Beauty (b) Amazon Automotive (c) Yelp (d) Google Local

Fig. 4. Performances of LR-GCCF+, LightGCN+ and CIGCN+ w.r.t. the ranking position N.

TABLE 5
Ablation Analysis of CIGCN+

Architecture	Amazon Beauty			Amazon Automotive			Yelp			Google Local		
	HR	NDCG	MRR	HR	NDCG	MRR	HR	NDCG	MRR	HR	NDCG	MRR
Default	0.3652	0.2276	0.1856	0.3265	0.1999	0.1609	**0.4149**	0.2334	0.1788	**0.6282**	**0.3949**	**0.3229**
Remove activation	0.3618	0.2231	0.1806	0.3254	0.1977	0.1587	0.4142	0.2324	0.1771	0.6247	0.3920	0.3200
Remove node dropout	0.3529	0.2206	0.1805	0.3163	0.1925	0.1549	0.4094	0.2313	0.1776	0.6266	0.3933	0.3213
Concatenation	**0.3675**	**0.2310**	**0.1890**	**0.3316**	**0.2032**	**0.1643**	0.4139	**0.2339**	**0.1791**	0.6254	0.3937	0.3223
Renormalization	0.3610	0.2244	0.1827	0.3253	0.1982	0.1596	0.4141	0.2333	0.1788	0.6265	0.3940	0.3223
1st-order ChebNet	0.2986	0.1871	0.1534	0.2606	0.1583	0.1272	0.3410	0.1852	0.1382	0.5769	0.3522	0.2831

Best results are in boldface.

achieve satisfactory performance, leading to higher training efficiency. Moreover, CIGCN+ outperforms LR-GCCF+ and LightGCN+ consistently with different numbers of graph convolutional layers.

5.4.2 *Impact of Embedding Size*

We now study how the embedding size d affects the performances of the graph convolution-based methods. Fig. 3 shows the performances of LR-GCCF+, LightGCN+ and CIGCN+ w.r.t. the embedding size d varying from 16 to 128.

From a macro perspective, with the increase of embedding size, the performances of the three methods improve due to stronger expressivity. However, the marginal improvement is decreasing. Specifically, CIGCN+ achieves better results than LR-GCCF+ and LightGCN+ under different embedding sizes on the four datasets.

5.4.3 *Impact of Ranking Position*

We also analyse how the performances of the graph convolution-based methods vary with the ranking position N. Fig. 4 shows the performances of LR-GCCF+, LightGCN+ and CIGCN+ w.r.t. the ranking position N ranging from 1 to 20.

We can see that the performances of the three methods improve as the ranking position increases, while the marginal improvement diminishes. In addition, CIGCN+ demonstrates consistent improvements over LR-GCCF+ and LightGCN+ across different ranking positions.

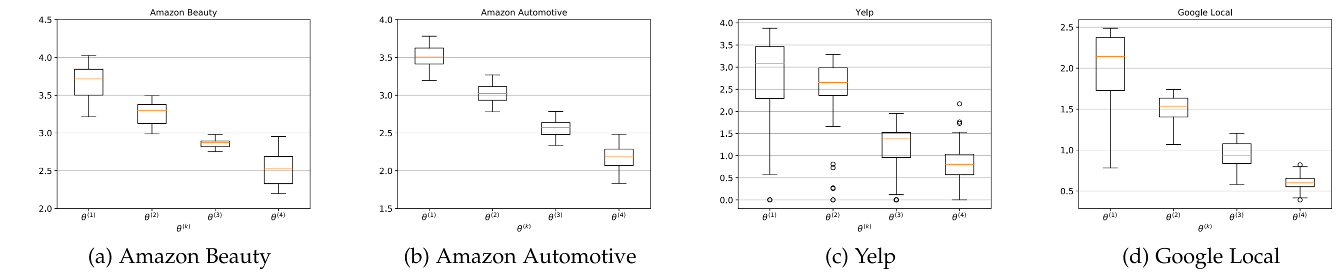
5.4.4 *Ablation Studies*

We conducted ablation experiments on the key components of CIGCN+ to better understand their impacts, including the activation function, the layer-aggregation function, the node dropout strategy, and the propagation matrix, with the results shown in Table 5. We analyze their effects respectively:

First, removing the activation function in graph convolutional layers would not affect the performance significantly. Since $\tanh(\cdot)$ is used as the activation function in CIGCN+, it normalizes the embeddings in each layer and helps the training process converge fast.

Second, removing node dropout would hurt the performance of CIGCN+, especially on sparser datasets (Amazon Beauty and Automotive). That is, the node dropout strategy could effectively regularize the model and avoid overfitting, benefiting its generalization ability.

Third, in some cases, the performance of CIGCN+ becomes better when using concatenation as the layer-aggregation function. Both sum-pooling and concatenation could help CIGCN+ leverage the information in different layers to enhance recommendation performance.



(a) Amazon Beauty (b) Amazon Automotive (c) Yelp (d) Google Local

Fig. 5. Visualization of $\boldsymbol{\theta}^{(k)}$ in CIGCN+.

Fourth, the form of the embedding propagation matrix $\hat{\mathbf{A}}$ would affect the performance significantly. The default setting of CIGCN+ uses the symmetric normalized adjacency matrix $\hat{\mathbf{A}} = \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$ (same as LightGCN [9]). When adding self-loops and using a renormalization form $\hat{\mathbf{A}} = (\mathbf{D}+\mathbf{I})^{-1/2}(\mathbf{A}+\mathbf{I})(\mathbf{D}+\mathbf{I})^{-1/2}$ (same as GCN [10]), the results are comparable. However, when using the form of the first-order ChebNet $\hat{\mathbf{A}} = \mathbf{I} + \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$ [17], the performance becomes significantly worse. Since this propagation matrix has contained the self-loops with a relatively large weight, using layer aggregations may make the model pay too much attention to the node's own embedding. In addition, this form of propagation matrix has eigenvalues in the range [0,2], and its high powers would result in exploding filter coefficients and numerical instabilities [10], [15].

5.5 Parameter Visualization (RQ3)

Recall that the filter parameters $\theta^{(k)}$ in CIGCN+ act as the trainable weights to learn the importance of embeddings in each layer and each dimension to the final embeddings. Here we visualize the parameters through boxplots on four datasets shown in Fig. 5.

As can be seen, the values of θ of each dimension in a layer are close to each other, while the scale diminishes with the increase of layers. This indicates that the filter parameters could make CIGCN+ learn properly smoothed embeddings by attaching different weights to different embedding layers, leading to stronger expressivity than the mean-pooling aggregation strategy used in LightGCN+ or the concatenation strategy used in LR-GCCF+.

5.6 Correlations of Embedding Dimensions (RQ4)

To show that CIGCN+ learns disentangled embeddings with the design of diagonal parameter matrices, we calculate the Pearson correlation coefficients of the embedding dimensions in each layer for the three graph convolution-based methods on Amazon Beauty and Amazon Automotive and draw boxplots in Fig. 6. "C", "L", "N" represent CIGCN+, LightGCN+, and NGCF+, respectively, and the numbers represent layers.

It can be seen that the correlations of the embedding dimensions in each layer of NGCF+ are much higher than CIGCN+ and LightGCN+ owing to its full parameter matrices in MLP, hurting the expressivity of embeddings. Since LightGCN+ is essentially a linear model, the correlations of its embedding dimensions tend to overlay with the increase of layers. While for CIGCN+, the correlations of embedding dimensions are around zero, indicating that CIGCN+ learns disentangled embeddings with stronger expressivity for recommendation.

6 Related Work

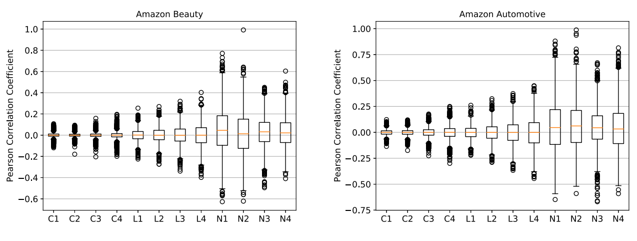
In this section, we review related work from three aspects: collaborative filtering, recommendation with item relations, and graph convolution-based recommendation methods.

6.1 Collaborative Filtering

Collaborative filtering (CF) is a group of methods that are widely used in recommender systems to generate recommendations according to user interactions. The earliest CF methods are based on memory, e.g., ItemKNN [1], [2] that computes the similarity of items according to their interacted users and recommends items to users which are most similar to their interacted ones. After that, model-based methods are proposed, which use user-item interactions to learn a model for making recommendations [3]. The most popular model-based CF method is Matrix Factorization (MF), which maps users and items to a joint latent factor space and models user-item interactions as inner products in that space [29]. Meanwhile, item-based methods are proposed to deal with the data sparsity problem. For example, SLIM [30] learns a sparse item similarity matrix by optimizing a regression-based objective function. FISM [31] represents each item with a low-dimensional embedding vector and factorizes the item-item similarity as the inner product of their embedding vectors. NAIS [32] extends FISM by using attention mechanisms to distinguish which items the user has interacted with are more important for predictions. Mult-VAE [27] is a variant of the variational autoencoder with a multinomial likelihood function for collaborative filtering on implicit feedback data.

6.2 Recommendation With Item Relations

In real-world scenarios, user feedback is often sparse, leading to poor performances of recommendation methods. To address this issue, recent efforts consider leveraging item relations to improve the representation learning of items.



(a) Amazon Beauty (b) Amazon Automotive

Fig. 6. Correlations between embedding dimensions of three graph convolution-based models. "C", "L", and "N" represent CIGCN+, LightGCN+, and NGCF+, respectively, and the numbers represent layers.

These methods fall within two categories: 1) using the straightforward item-item relations. These methods construct a homogeneous item graph according to the item relations. For example, MCF [6] jointly factorizes the user rating data and the "also-viewed" relations among items by sharing item representations. PACE [33] exploits context graphs and applies POI smoothing based on their geolocation for POI recommendation. MoHR [8] adopts translational operations to jointly model the user-item interactions and the multiple item-item relations for sequential recommendation. 2) Using the item-related knowledge graphs. These efforts build a heterogeneous information network, where some nodes represent items and others represent related entities. For example, CKE [7] uses the knowledge graph embedding techniques to automatically learn semantic item embeddings for improving recommendation accuracy. MKR [34] introduces cross&compress units to automatically share latent features and learn high-order interactions between items and related entities in the knowledge graph. RippleNet [21] propagates user preferences and explores their hierarchical interests in the knowledge graph for click-through rate prediction. RCF [35] develops a two-level hierarchical attention mechanism to model user preferences by considering both the relation types and values.

6.3 Graph Convolution-Based Recommendation

Recently, graph convolutions have been widely used in recommendation methods to deal with graph-structured data. Specifically, graph convolutions are mainly applied to four kinds of graphs: 1) user-item interaction graphs. For example, GCMC [12] uses a graph convolution layer to construct user and item embeddings through message passing on the user-item interaction graph. SpectralCF [13] proposes a spectral convolution operation to discover deep connections between users and items from the spectral domain. NGCF [11] propagates user and item embeddings on the user-item interaction graph through a well-designed embedding propagation layer to model the multi-order connections. LR-GCCF [14] removes the non-linear transformations in GCN and proposes a linear embedding propagation strategy and a residual network structure for item recommendation. LightGCN [9] learns user/item embedding by linearly propagation on the user-item interaction graph to generate the final embeddings with a mean-pooling strategy for item recommendation. 2) Knowledge graphs. For example, KGCN [36] proposes knowledge graph convolutional networks that aggregate neighborhood information biasedly to learn both structure and semantic information of the knowledge graph as well as user personalized interests. KGAT [37] proposes knowledge graph attention networks that propagate node embeddings in the knowledge graph with attention mechanisms to generate the knowledge-aware item recommendations. 3) User social graphs. For example, DGRec [38] models dynamic user behaviors with recurrent neural networks and context-dependent social influence with graph attention networks for dynamic social recommendation. DANSER [39] introduces dual graph attention networks to collaboratively learn representations for social effects that are modeled by a user-specific attention weight and are modeled by a dynamic and context-aware attention weight, respectively. GraphRec [40] captures interactions and opinions in the user-item graph with graph neural networks for social recommendation. 4) Item sequential graphs. For example, SR-GNN [41] and GC-SAN [42] aggregate session sequences together to build a graph structure data and apply gated graph neural networks on it to capture the complex transitions of items for session-based recommendation.

7 Conclusion

In this study, we have proposed a novel graph convolution-based recommendation method, namely Channel-Independent Graph Convolutional Network (CIGCN). CIGCN adopts diagonal parameter matrices as filters in graph convolution and uses layer-aggregation strategies to obtain the final embeddings of users and items. The advantages of these designs are twofold. 1) The diagonal parameter matrices could keep the embedding dimensions independent in graph convolutional layers, enabling the model to learn disentangled representations for users and items. 2) With layer-aggregation strategies, the parameters in the diagonal matrices act as trainable weights that attach different importance to the embeddings in each layer and each dimension, enhancing the model expressivity. Results of extensive experiments on four real-world datasets demonstrate the state-of-the-art performance of CIGCN and its effectiveness in learning better representations for users and items.

For future work, an effort could be made to extend CIGCN by considering other available relations, such as user-user relations in the social network and item-entity relations in the knowledge graph, to improve its flexibility and scalability and further enhance recommendation performance.

Acknowledgments

The work was supported in part by the National Natural Science Foundation of China under Grant 71901011 and in part by the MOE Project of Key Research Institute of Humanities and Social Sciences at Universities under Grant 17JJD630006.

References

[1] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Item-based collaborative filtering recommendation algorithms," in *Proc. 10th Int. Conf. World Wide Web*, 2001, pp. 285–295.
[2] G. Linden, B. Smith, and J. York, "Amazon.com recommendations: Item-to-item collaborative filtering," *IEEE Internet Comput.*, vol. 7, no. 1, pp. 76–80, Jan./Feb. 2003.
[3] G. Adomavicius and A. Tuzhilin, "Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions," *IEEE Trans. Knowl. Data Eng.*, vol. 17, no. 6, pp. 734–749, Jun. 2005.
[4] G. Adomavicius, J. C. Bockstedt, S. P. Curley, and J. Zhang, "Effects of online recommendations on consumers' willingness to pay," *Inf. Syst. Res.*, vol. 29, no. 1, pp. 84–102, 2018.
[5] C. A. Gomez-Uribe and N. Hunt, "The Netflix recommender system: Algorithms, business value, and innovation," *ACM Trans. Manage. Inf. Syst.*, vol. 6, no. 4, pp. 1–19, 2015.
[6] C. Park, D. Kim, J. Oh, and H. Yu, "Do also-viewed products help user rating prediction?," in *Proc. 26th Int. Conf. World Wide Web*, 2017, pp. 1113–1122.
[7] F. Zhang, N. J. Yuan, D. Lian, X. Xie, and W.-Y. Ma, "Collaborative knowledge base embedding for recommender systems," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2016, pp. 353–362.

[8] W.-C. Kang, M. Wan, and J. McAuley, “Recommendation through mixtures of heterogeneous item relationships,” in *Proc. 27th ACM Int. Conf. Inf. Knowl. Manage.*, 2018, pp. 1143–1152.

[9] X. He, K. Deng, X. Wang, Y. Li, Y. Zhang, and M. Wang, “LightGCN: Simplifying and powering graph convolution network for recommendation,” 2020, *arXiv:2002.02126*.

[10] T. N. Kipf and M. Welling, “Semi-supervised classification with graph convolutional networks,” 2016, *arXiv:1609.02907*.

[11] X. Wang, X. He, M. Wang, F. Feng, and T.-S. Chua, “Neural graph collaborative filtering,” in *Proc. 42nd Int. ACM SIGIR Conf. Res. Develop. Inf. Retrieval*, 2019, pp. 165–174.

[12] R. v. d. Berg, T. N. Kipf, and M. Welling, “Graph convolutional matrix completion,” 2017, *arXiv:1706.02263*.

[13] L. Zheng, C.-T. Lu, F. Jiang, J. Zhang, and P. S. Yu, “Spectral collaborative filtering,” in *Proc. 12th ACM Conf. Recommender Syst.*, 2018, pp. 311–319.

[14] L. Chen, L. Wu, R. Hong, K. Zhang, and M. Wang, “Revisiting graph based collaborative filtering: A linear residual graph convolutional network approach,” *Proc. AAAI Conf. Artif. Intell.*, vol. 34, no. 1, 2020, pp. 27–34.

[15] F. Wu, T. Zhang, A. H. d. Souza Jr., C. Fifty, T. Yu, and K. Q. Weinberger, “Simplifying graph convolutional networks,” 2019, *arXiv:1902.07153*.

[16] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, “Spectral networks and locally connected networks on graphs,” 2013, *arXiv:1312.6203*.

[17] M. Defferrard, X. Bresson, and P. Vandergheynst, “Convolutional neural networks on graphs with fast localized spectral filtering,” in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 3844–3852.

[18] K. Xu, C. Li, Y. Tian, T. Sonobe, K.-I. Kawarabayashi, and S. Jegelka, “Representation learning on graphs with jumping knowledge networks,” 2018, *arXiv:1806.03536*.

[19] Q. Li, Z. Han, and X.-M. Wu, “Deeper insights into graph convolutional networks for semi-supervised learning,” in *Proc. 30nd AAAI Conf. Artif. Intell.*, 2018, pp. 3538–3545.

[20] G. Li, M. Muller, A. Thabet, and B. Ghanem, “DeepGCNs: Can GCNs go as deep as CNNs?,” in *Proc. IEEE Int. Conf. Comput. Vis.*, 2019, pp. 9267–9276.

[21] H. Wang *et al.*, “RippleNet: Propagating user preferences on the knowledge graph for recommender systems,” in *Proc. 27th ACM Int. Conf. Inf. Knowl. Manage.*, 2018, pp. 417–426.

[22] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme, “BPR: Bayesian personalized ranking from implicit feedback,” in *Proc. 25th Conf. Uncertainty Artif. Intell.*, 2009, pp. 452–461.

[23] D. P. Kingma and J. L. Ba, “Adam: A method for stochastic optimization,” in *Proc. Int. Conf. Learn. Representations*, 2015, *arXiv:1412.6980*.

[24] R. He and J. McAuley, “Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering,” in *Proc. 25th Int. Conf. World Wide Web*, 2016, pp. 507–517.

[25] R. He, W.-C. Kang, and J. McAuley, “Translation-based recommendation,” in *Proc. 11th ACM Conf. Recommender Syst.*, 2017, pp. 161–169.

[26] W.-C. Kang and J. McAuley, “Self-attentive sequential recommendation,” in *Proc. IEEE Int. Conf. Data Mining*, 2018, pp. 197–206.

[27] D. Liang, R. G. Krishnan, M. D. Hoffman, and T. Jebara, “Variational autoencoders for collaborative filtering,” in *Proc. 27th Int. Conf. World Wide Web*, 2018, pp. 689–698.

[28] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” 2014, *arXiv:1412.6980*.

[29] Y. Koren, R. Bell, and C. Volinsky, “Matrix factorization techniques for recommender systems,” *Computer*, vol. 42, no. 8, pp. 30–37, 2009.

[30] X. Ning and G. Karypis, “Slim: Sparse linear methods for top-n recommender systems,” in *Proc. 11th IEEE Int. Conf. Data Mining*, 2011, pp. 497–506.

[31] S. Kabbur, X. Ning, and G. Karypis, “FISM: Factored item similarity models for top-n recommender systems,” in *Proc. 19nd ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2013, pp. 659–667.

[32] X. He, Z. He, J. Song, Z. Liu, Y.-G. Jiang, and T.-S. Chua, “NAIS: Neural attentive item similarity model for recommendation,” *IEEE Trans. Knowl. Data Eng.*, vol. 30, no. 12, pp. 2354–2366, Dec. 2018.

[33] C. Yang, L. Bai, C. Zhang, Q. Yuan, and J. Han, “Bridging collaborative filtering and semi-supervised learning: A neural approach for poi recommendation,” in *Proc. 23rd ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2017, pp. 1245–1254.

[34] H. Wang, F. Zhang, M. Zhao, W. Li, X. Xie, and M. Guo, “Multi-task feature learning for knowledge graph enhanced recommendation,” in *Proc. World Wide Web Conf.*, 2019, pp. 2000–2010.

[35] X. Xin, X. He, Y. Zhang, Y. Zhang, and J. Jose, “Relational collaborative filtering: Modeling multiple item relations for recommendation,” in *Proc. 42nd Int. ACM SIGIR Conf. Res. Develop. Inf. Retrieval*, 2019, pp. 125–134.

[36] H. Wang, M. Zhao, X. Xie, W. Li, and M. Guo, “Knowledge graph convolutional networks for recommender systems,” in *Proc. World Wide Web Conf.*, 2019, pp. 3307–3313.

[37] X. Wang, X. He, Y. Cao, M. Liu, and T.-S. Chua, “KGAT: Knowledge graph attention network for recommendation,” in *Proc. 25th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2019, pp. 950–958.

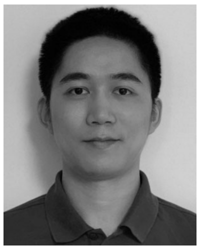
[38] W. Song, Z. Xiao, Y. Wang, L. Charlin, M. Zhang, and J. Tang, “Session-based social recommendation via dynamic graph attention networks,” in *Proc. 12th ACM Int. Conf. Web Search Data Mining*, 2019, pp. 555–563.

[39] Q. Wu *et al.*, “Dual graph attention networks for deep latent representation of multifaceted social effects in recommender systems,” in *Proc. World Wide Web Conf.*, 2019, pp. 2091–2102.

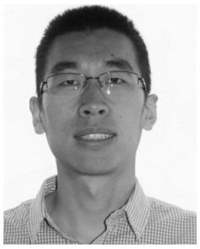
[40] W. Fan *et al.*, “Graph neural networks for social recommendation,” in *Proc. World Wide Web Conf.*, 2019, pp. 417–426.

[41] S. Wu, Y. Tang, Y. Zhu, L. Wang, X. Xie, and T. Tan, “Session-based recommendation with graph neural networks,” in *Proc. AAAI Conf. Artif. Intell.*, 2019, vol. 33, pp. 346–353.

[42] C. Xu *et al.*, “Graph contextualized self-attention network for session-based recommendation,” in *Proc. IJCAI Int. Joint Conf. Artif. Intell.*, 2019, pp. 3940–3946.



Tianyu Zhu is currently working toward the PhD degree with the Department of Management Science and Engineering, Tsinghua University, Beijing, China. His research interests include machine learning, data mining, and recommender systems.



Leilei Sun received the PhD from the Institute of Systems Engineering, Dalian University of Technology, Dalian, China, in 2017. He is currently an assistant professor with the School of Computer Science, Beihang University, Beijing, China. His research interests include machine learning and data mining.

Guoqing Chen received the PhD degree from the Catholic University of Leuven, KUL, Belgium, in 1992. He is currently the chair professor with the Department of Management Science and Engineering, Tsinghua University, Beijing China. His research interests include data mining and business analytics, recommender systems, information extraction, fuzzy logic, and data modeling.