Neural Graph Collaborative Filtering[*]

Xiang Wang
National University of Singapore
xiangwang@u.nus.edu

Xiangnan He[†]
University of Science and Technology of China
xiangnanhe@gmail.com

Meng Wang
Hefei University of Technology
eric.mengwang@gmail.com

Fuli Feng
National University of Singapore
fulifeng93@gmail.com

Tat-Seng Chua
National University of Singapore
dcscts@nus.edu.sg

ABSTRACT

Learning vector representations (*aka.* embeddings) of users and items lies at the core of modern recommender systems. Ranging from early matrix factorization to recently emerged deep learning based methods, existing efforts typically obtain a user's (or an item's) embedding by mapping from pre-existing features that describe the user (or the item), such as ID and attributes. We argue that an inherent drawback of such methods is that, the **collaborative signal**, which is latent in user-item interactions, is not encoded in the embedding process. As such, the resultant embeddings may not be sufficient to capture the collaborative filtering effect.

In this work, we propose to integrate the user-item interactions — more specifically the bipartite graph structure — into the embedding process. We develop a new recommendation framework *Neural Graph Collaborative Filtering* (NGCF), which exploits the user-item graph structure by propagating embeddings on it. This leads to the expressive modeling of **high-order connectivity** in user-item graph, effectively injecting the collaborative signal into the embedding process in an explicit manner. We conduct extensive experiments on three public benchmarks, demonstrating significant improvements over several state-of-the-art models like HOP-Rec [40] and Collaborative Memory Network [5]. Further analysis verifies the importance of embedding propagation for learning better user and item representations, justifying the rationality and effectiveness of NGCF. Codes are available at https://github.com/xiangwang1223/neural_graph_collaborative_filtering.

CCS CONCEPTS

• **Information systems → Recommender systems**.

[*]In the version published in ACM Digital Library, we find some small bugs; the bugs do not change the comparison results and the empirical findings. In this latest version, we update and correct the experimental results (*i.e.,* the preprocessing of Yelp2018 dataset and the ndcg metric). All updates are highlighted in footnotes.
[†]Xiangnan He is the corresponding author.

KEYWORDS

Collaborative Filtering, Recommendation, High-order Connectivity, Embedding Propagation, Graph Neural Network

ACM Reference Format:
Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. 2019. Neural Graph Collaborative Filtering. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '19), July 21–25, 2019, Paris, France.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3331184.3331267

1 INTRODUCTION

Personalized recommendation is ubiquitous, having been applied to many online services such as E-commerce, advertising, and social media. At its core is estimating how likely a user will adopt an item based on the historical interactions like purchases and clicks. Collaborative filtering (CF) addresses it by assuming that behaviorally similar users would exhibit similar preference on items. To implement the assumption, a common paradigm is to parameterize users and items for reconstructing historical interactions, and predict user preference based on the parameters [1, 14].

Generally speaking, there are two key components in learnable CF models — 1) *embedding*, which transforms users and items to vectorized representations, and 2) *interaction modeling*, which reconstructs historical interactions based on the embeddings. For example, matrix factorization (MF) directly embeds user/item ID as an vector and models user-item interaction with inner product [20]; collaborative deep learning extends the MF embedding function by integrating the deep representations learned from rich side information of items [30]; neural collaborative filtering models replace the MF interaction function of inner product with nonlinear neural networks [14]; and translation-based CF models instead use Euclidean distance metric as the interaction function [28], among others.

Despite their effectiveness, we argue that these methods are not sufficient to yield satisfactory embeddings for CF. The key reason is that the embedding function lacks an explicit encoding of the crucial **collaborative signal**, which is latent in user-item interactions to reveal the behavioral similarity between users (or items). To be more specific, most existing methods build the embedding function with the descriptive features only (*e.g.,* ID and attributes), without considering the *user-item interactions* — which are only used to define the objective function for model training [26, 28]. As a result,

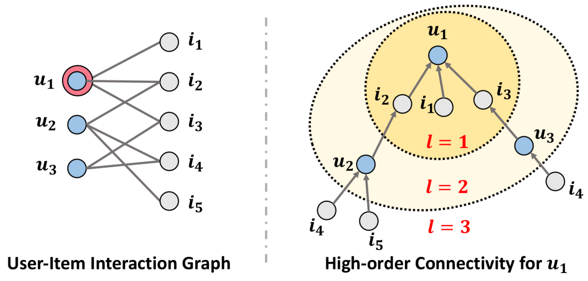


Figure 1: An illustration of the user-item interaction graph and the high-order connectivity. The node u_1 is the target user to provide recommendations for.

when the embeddings are insufficient in capturing CF, the methods have to rely on the interaction function to make up for the deficiency of suboptimal embeddings [14].

While intuitively useful to integrate user-item interactions into the embedding function, it is non-trivial to do it well. In particular, the scale of interactions can easily reach millions or even larger in real applications, making it difficult to distill the desired collaborative signal. In this work, we tackle the challenge by exploiting the **high-order connectivity** from user-item interactions, a natural way that encodes collaborative signal in the interaction graph structure.

Running Example. Figure 1 illustrates the concept of high-order connectivity. The user of interest for recommendation is u_1, labeled with the double circle in the left subfigure of user-item interaction graph. The right subfigure shows the tree structure that is expanded from u_1. The high-order connectivity denotes the path that reaches u_1 from any node with the path length l larger than 1. Such high-order connectivity contains rich semantics that carry collaborative signal. For example, the path $u_1 \leftarrow i_2 \leftarrow u_2$ indicates the behavior similarity between u_1 and u_2, as both users have interacted with i_2; the longer path $u_1 \leftarrow i_2 \leftarrow u_2 \leftarrow i_4$ suggests that u_1 is likely to adopt i_4, since her similar user u_2 has consumed i_4 before. Moreover, from the holistic view of $l = 3$, item i_4 is more likely to be of interest to u_1 than item i_5, since there are two paths connecting $<i_4, u_1>$, while only one path connects $<i_5, u_1>$.

Present Work. We propose to model the high-order connectivity information in the embedding function. Instead of expanding the interaction graph as a tree which is complex to implement, we design a neural network method to propagate embeddings recursively on the graph. This is inspired by the recent developments of graph neural networks [8, 32, 38], which can be seen as constructing information flows in the embedding space. Specifically, we devise an **embedding propagation** layer, which refines a user's (or an item's) embedding by aggregating the embeddings of the interacted items (or users). By stacking multiple embedding propagation layers, we can enforce the embeddings to capture the collaborative signal in high-order connectivities. Taking Figure 1 as an example, stacking two layers captures the behavior similarity of $u_1 \leftarrow i_2 \leftarrow u_2$, stacking three layers captures the potential recommendations of $u_1 \leftarrow i_2 \leftarrow u_2 \leftarrow i_4$, and the strength of the information flow (which is estimated by the trainable weights between layers) determines the recommendation priority of i_4 and i_5. We conduct extensive experiments on three public benchmarks to verify the rationality and effectiveness of our *Neural Graph Collaborative Filtering* (NGCF) method.

Lastly, it is worth mentioning that although the high-order connectivity information has been considered in a very recent method named HOP-Rec [40], it is only exploited to enrich the training data. Specifically, the prediction model of HOP-Rec remains to be MF, while it is trained by optimizing a loss that is augmented with high-order connectivities. Distinct from HOP-Rec, we contribute a new technique to integrate high-order connectivities into the prediction model, which empirically yields better embeddings than HOP-Rec for CF.

To summarize, this work makes the following main contributions:

- We highlight the critical importance of explicitly exploiting the collaborative signal in the embedding function of model-based CF methods.
- We propose NGCF, a new recommendation framework based on graph neural network, which explicitly encodes the collaborative signal in the form of high-order connectivities by performing embedding propagation.
- We conduct empirical studies on three million-size datasets. Extensive results demonstrate the state-of-the-art performance of NGCF and its effectiveness in improving the embedding quality with neural embedding propagation.

2 METHODOLOGY

We now present the proposed NGCF model, the architecture of which is illustrated in Figure 2. There are three components in the framework: (1) an embedding layer that offers and initialization of user embeddings and item embeddings; (2) multiple embedding propagation layers that refine the embeddings by injecting high-order connectivity relations; and (3) the prediction layer that aggregates the refined embeddings from different propagation layers and outputs the affinity score of a user-item pair. Finally, we discuss the time complexity of NGCF and the connections with existing methods.

2.1 Embedding Layer

Following mainstream recommender models [1, 14, 26], we describe a user u (an item i) with an embedding vector $\mathbf{e}_u \in \mathbb{R}^d$ ($\mathbf{e}_i \in \mathbb{R}^d$), where d denotes the embedding size. This can be seen as building a parameter matrix as an embedding look-up table:

$$\mathbf{E} = [\underbrace{\mathbf{e}_{u_1}, \cdots, \mathbf{e}_{u_N}}_{\text{users embeddings}}, \underbrace{\mathbf{e}_{i_1}, \cdots, \mathbf{e}_{i_M}}_{\text{item embeddings}}]. \quad (1)$$

It is worth noting that this embedding table serves as an initial state for user embeddings and item embeddings, to be optimized in an end-to-end fashion. In traditional recommender models like MF and neural collaborative filtering [14], these ID embeddings are directly fed into an interaction layer (or operator) to achieve the prediction score. In contrast, in our NGCF framework, we refine the embeddings by propagating them on the user-item interaction graph. This leads to more effective embeddings for recommendation, since the embedding refinement step explicitly injects collaborative signal into embeddings.

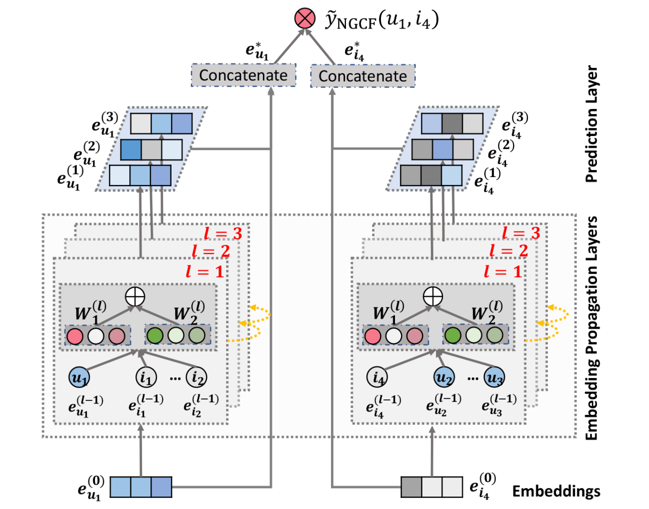


Figure 2: An illustration of NGCF model architecture (the arrowed lines present the flow of information). The representations of user u_1 (left) and item i_4 (right) are refined with multiple embedding propagation layers, whose outputs are concatenated to make the final prediction.

2.2 Embedding Propagation Layers

Next we build upon the message-passing architecture of GNNs [8, 38] in order to capture CF signal along the graph structure and refine the embeddings of users and items. We first illustrate the design of one-layer propagation, and then generalize it to multiple successive layers.

2.2.1 *First-order Propagation*.

Intuitively, the interacted items provide direct evidence on a user's preference [16, 39]; analogously, the users that consume an item can be treated as the item's features and used to measure the collaborative similarity of two items. We build upon this basis to perform embedding propagation between the connected users and items, formulating the process with two major operations: *message construction* and *message aggregation*.

Message Construction. For a connected user-item pair (u, i), we define the message from i to u as:

$$\mathbf{m}_{u \leftarrow i} = f(\mathbf{e}_i, \mathbf{e}_u, p_{ui}), \tag{2}$$

where $\mathbf{m}_{u \leftarrow i}$ is the message embedding (*i.e.,* the information to be propagated). $f(\cdot)$ is the message encoding function, which takes embeddings $\mathbf{e}_i$ and $\mathbf{e}_u$ as input, and uses the coefficient p_{ui} to control the decay factor on each propagation on edge (u, i).

In this work, we implement $f(\cdot)$ as:

$$\mathbf{m}_{u \leftarrow i} = \frac{1}{\sqrt{|\mathcal{N}_u||\mathcal{N}_i|}} \Big(\mathbf{W}_1 \mathbf{e}_i + \mathbf{W}_2 (\mathbf{e}_i \odot \mathbf{e}_u) \Big), \tag{3}$$

where $\mathbf{W}_1, \mathbf{W}_2 \in \mathbb{R}^{d' \times d}$ are the trainable weight matrices to distill useful information for propagation, and d' is the transformation size. Distinct from conventional graph convolution networks [4, 18, 29, 42] that consider the contribution of $\mathbf{e}_i$ only, here we additionally encode the interaction between $\mathbf{e}_i$ and $\mathbf{e}_u$ into the message being passed via $\mathbf{e}_i \odot \mathbf{e}_u$, where $\odot$ denotes the element-wise product. This makes the message dependent on the affinity between $\mathbf{e}_i$ and $\mathbf{e}_u$, *e.g.,* passing more messages from the similar items. This not

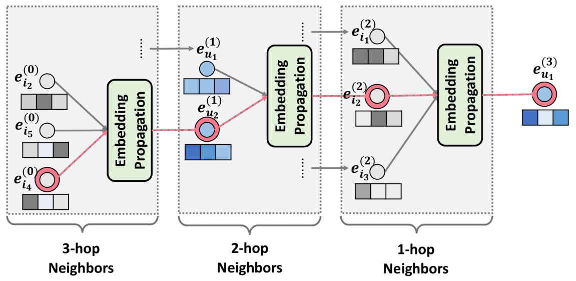


Figure 3: Illustration of third-order embedding propagation for user u_1. Best view in color.

only increases the model representation ability, but also boosts the performance for recommendation (evidences in our experiments Section 4.4.2).

Following the graph convolutional network [18], we set p_{ui} as the graph Laplacian norm $1/\sqrt{|\mathcal{N}_u||\mathcal{N}_i|}$, where $\mathcal{N}_u$ and $\mathcal{N}_i$ denote the first-hop neighbors of user u and item i. From the viewpoint of representation learning, p_{ui} reflects how much the historical item contributes the user preference. From the viewpoint of message passing, p_{ui} can be interpreted as a discount factor, considering the messages being propagated should decay with the path length.

Message Aggregation. In this stage, we aggregate the messages propagated from u's neighborhood to refine u's representation. Specifically, we define the aggregation function as:

$$\mathbf{e}_u^{(1)} = \text{LeakyReLU}\Big(\mathbf{m}_{u \leftarrow u} + \sum_{i \in \mathcal{N}_u} \mathbf{m}_{u \leftarrow i} \Big), \tag{4}$$

where $\mathbf{e}_u^{(1)}$ denotes the representation of user u obtained after the first embedding propagation layer. The activation function of LeakyReLU [23] allows messages to encode both positive and small negative signals. Note that in addition to the messages propagated from neighbors $\mathcal{N}_u$, we take the self-connection of u into consideration: $\mathbf{m}_{u \leftarrow u} = \mathbf{W}_1 \mathbf{e}_u$, which retains the information of original features ($\mathbf{W}_1$ is the weight matrix shared with the one used in Equation (3)). Analogously, we can obtain the representation $\mathbf{e}_i^{(1)}$ for item i by propagating information from its connected users. To summarize, the advantage of the embedding propagation layer lies in explicitly exploiting the first-order connectivity information to relate user and item representations.

2.2.2 *High-order Propagation*.

With the representations augmented by first-order connectivity modeling, we can stack more embedding propagation layers to explore the high-order connectivity information. Such high-order connectivities are crucial to encode the collaborative signal to estimate the relevance score between a user and item.

By stacking l embedding propagation layers, a user (and an item) is capable of receiving the messages propagated from its l-hop neighbors. As Figure 2 displays, in the l-th step, the representation of user u is recursively formulated as:

$$\mathbf{e}_u^{(l)} = \text{LeakyReLU}\Big(\mathbf{m}_{u \leftarrow u}^{(l)} + \sum_{i \in \mathcal{N}_u} \mathbf{m}_{u \leftarrow i}^{(l)} \Big), \tag{5}$$

wherein the messages being propagated are defined as follows,

$$\begin{cases} \mathbf{m}_{u\leftarrow i}^{(l)} = p_{ui}\Big(\mathbf{W}_1^{(l)}\mathbf{e}_i^{(l-1)} + \mathbf{W}_2^{(l)}(\mathbf{e}_i^{(l-1)} \odot \mathbf{e}_u^{(l-1)})\Big), \\ \mathbf{m}_{u\leftarrow u}^{(l)} = \mathbf{W}_1^{(l)}\mathbf{e}_u^{(l-1)}, \end{cases} \tag{6}$$

where $\mathbf{W}_1^{(l)}, \mathbf{W}_2^{(l)}, \in \mathbb{R}^{d_l \times d_{l-1}}$ are the trainable transformation matrices, and d_l is the transformation size; $\mathbf{e}_i^{(l-1)}$ is the item representation generated from the previous message-passing steps, memorizing the messages from its (l-1)-hop neighbors. It further contributes to the representation of user u at layer l. Analogously, we can obtain the representation for item i at the layer l.

As Figure 3 shows, the collaborative signal like $u_1 \leftarrow i_2 \leftarrow u_2 \leftarrow i_4$ can be captured in the embedding propagation process. Furthermore, the message from i_4 is explicitly encoded in $\mathbf{e}_{u_1}^{(3)}$ (indicated by the red line). As such, stacking multiple embedding propagation layers seamlessly injects collaborative signal into the representation learning process.

Propagation Rule in Matrix Form. To offer a holistic view of embedding propagation and facilitate batch implementation, we provide the matrix form of the layer-wise propagation rule (equivalent to Equations (5) and (6)):

$$\mathbf{E}^{(l)} = \text{LeakyReLU}\Big((\mathcal{L} + \mathbf{I})\mathbf{E}^{(l-1)}\mathbf{W}_1^{(l)} + \mathcal{L}\mathbf{E}^{(l-1)} \odot \mathbf{E}^{(l-1)}\mathbf{W}_2^{(l)}\Big), \tag{7}$$

where $\mathbf{E}^{(l)} \in \mathbb{R}^{(N+M)\times d_l}$ are the representations for users and items obtained after l steps of embedding propagation. $\mathbf{E}^{(0)}$ is set as $\mathbf{E}$ at the initial message-passing iteration, that is $\mathbf{e}_u^{(0)} = \mathbf{e}_u$ and $\mathbf{e}_i^{(0)} = \mathbf{e}_i$; and $\mathbf{I}$ denote an identity matrix. $\mathcal{L}$ represents the Laplacian matrix for the user-item graph, which is formulated as:

$$\mathcal{L} = \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}} \text{ and } \mathbf{A} = \begin{bmatrix} \mathbf{0} & \mathbf{R} \\ \mathbf{R}^\top & \mathbf{0} \end{bmatrix}, \tag{8}$$

where $\mathbf{R} \in R^{N\times M}$ is the user-item interaction matrix, and $\mathbf{0}$ is all-zero matrix; $\mathbf{A}$ is the adjacency matrix and $\mathbf{D}$ is the diagonal degree matrix, where the t-th diagonal element $D_{tt} = |\mathcal{N}_t|$; as such, the nonzero off-diagonal entry $\mathcal{L}_{ui} = 1/\sqrt{|\mathcal{N}_u||\mathcal{N}_i|}$, which is equal to p_{ui} used in Equation (3).

By implementing the matrix-form propagation rule, we can simultaneously update the representations for all users and items in a rather efficient way. It allows us to discard the node sampling procedure, which is commonly used to make graph convolution network runnable on large-scale graph [25]. We will analyze the complexity in Section 2.5.2.

2.3 Model Prediction

After propagating with L layers, we obtain multiple representations for user u, namely $\{\mathbf{e}_u^{(1)}, \cdots, \mathbf{e}_u^{(L)}\}$. Since the representations obtained in different layers emphasize the messages passed over different connections, they have different contributions in reflecting user preference. As such, we concatenate them to constitute the final embedding for a user; we do the same operation on items, concatenating the item representations $\{\mathbf{e}_i^{(1)}, \cdots, \mathbf{e}_i^{(L)}\}$ learned by different layers to get the final item embedding:

$$\mathbf{e}_u^* = \mathbf{e}_u^{(0)} \| \cdots \| \mathbf{e}_u^{(L)}, \quad \mathbf{e}_i^* = \mathbf{e}_i^{(0)} \| \cdots \| \mathbf{e}_i^{(L)}, \tag{9}$$

where $\|$ is the concatenation operation. By doing so, we not only enrich the initial embeddings with embedding propagation layers, but also allow controlling the range of propagation by adjusting L. Note that besides concatenation, other aggregators can also be applied, such as weighted average, max pooling, LSTM, *etc.*, which imply different assumptions in combining the connectivities of different orders. The advantage of using concatenation lies in its simplicity, since it involves no additional parameters to learn, and it has been shown quite effectively in a recent work of graph neural networks [38], which refers to layer-aggregation mechanism.

Finally, we conduct the inner product to estimate the user's preference towards the target item:

$$\hat{y}_{\text{NGCF}}(u, i) = \mathbf{e}_u^{*\top}\mathbf{e}_i^*. \tag{10}$$

In this work, we emphasize the embedding function learning thus only employ the simple interaction function of inner product. Other more complicated choices, such as neural network-based interaction functions [14], are left to explore in the future work.

2.4 Optimization

To learn model parameters, we optimize the pairwise BPR loss [26], which has been intensively used in recommender systems [2, 13]. It considers the relative order between observed and unobserved user-item interactions. Specifically, BPR assumes that the observed interactions, which are more reflective of a user's preferences, should be assigned higher prediction values than unobserved ones. The objective function is as follows,

$$Loss = \sum_{(u,i,j)\in O} -\ln\sigma(\hat{y}_{ui} - \hat{y}_{uj}) + \lambda\,\|\Theta\|_2^2, \tag{11}$$

where $O = \{(u, i, j)|(u, i) \in \mathcal{R}^+, (u, j) \in \mathcal{R}^-\}$ denotes the pairwise training data, $\mathcal{R}^+$ indicates the observed interactions, and $\mathcal{R}^-$ is the unobserved interactions; $\sigma(\cdot)$ is the sigmoid function; $\Theta = \{\mathbf{E}, \{\mathbf{W}_1^{(l)}, \mathbf{W}_2^{(l)}\}_{l=1}^L\}$ denotes all trainable model parameters, and λ controls the L_2 regularization strength to prevent overfitting. We adopt mini-batch Adam [17] to optimize the prediction model and update the model parameters. In particular, for a batch of randomly sampled triples $(u, i, j) \in O$, we establish their representations $[\mathbf{e}^{(0)}, \cdots, \mathbf{e}^{(L)}]$ after L steps of propagation, and then update model parameters by using the gradients of the loss function.

2.4.1 *Model Size.*

It is worth pointing out that although NGCF obtains an embedding matrix ($\mathbf{E}^{(l)}$) at each propagation layer l, it only introduces very few parameters — two weight matrices of size $d_l \times d_{l-1}$. Specifically, these embedding matrices are derived from the embedding look-up table $\mathbf{E}^{(0)}$, with the transformation based on the user-item graph structure and weight matrices. As such, compared to MF — the most concise embedding-based recommender model, our NGCF uses only $2Ld_ld_{l-1}$ more parameters. Such additional cost on model parameters is almost negligible, considering that L is usually a number smaller than 5, and d_l is typically set as the embedding size, which is much smaller than the number of users and items. For example, on our experimented Gowalla dataset (20K users and 40K items), when the embedding size is 64 and we use 3 propagation layers of size 64×64, MF has 4.5 million parameters, while our NGCF uses only 0.024 million additional parameters. To summarize, NGCF uses very few additional model parameters to achieve the high-order connectivity modeling.

2.4.2 ***Message and Node Dropout.*** Although deep learning models have strong representation ability, they usually suffer from overfitting. Dropout is an effective solution to prevent neural networks from overfitting. Following the prior work on graph convolutional network [29], we propose to adopt two dropout techniques in NGCF: *message dropout* and *node dropout*. Message dropout randomly drops out the outgoing messages. Specifically, we drop out the messages being propagated in Equation (6), with a probability p_1. As such, in the l-th propagation layer, only partial messages contribute to the refined representations. We also conduct node dropout to randomly block a particular node and discard all its outgoing messages. For the l-th propagation layer, we randomly drop $(M+N)p_2$ nodes of the Laplacian matrix, where p_2 is the dropout ratio.

Note that dropout is only used in training, and must be disabled during testing. The message dropout endows the representations more robustness against the presence or absence of single connections between users and items, and the node dropout focuses on reducing the influences of particular users or items. We perform experiments to investigate the impact of message dropout and node dropout on NGCF in Section 4.4.3.

2.5 Discussions

In the subsection, we first show how NGCF generalizes SVD++ [19]. In what follows, we analyze the time complexity of NGCF.

2.5.1 ***NGCF Generalizes SVD++.*** SVD++ can be viewed as a special case of NGCF with no high-order propagation layer. In particular, we set L to one. Within the propagation layer, we disable the transformation matrix and nonlinear activation function. Thereafter, $\mathbf{e}_u^{(1)}$ and $\mathbf{e}_i^{(1)}$ are treated as the final representations for user u and item i, respectively. We term this simplified model as NGCF-SVD, which can be formulated as:

$$\hat{y}_{\text{NGCF-SVD}} = \Big(\mathbf{e}_u + \sum_{i' \in \mathcal{N}_u} p_{ui'}\mathbf{e}_{i'}\Big)^{\top}\Big(\mathbf{e}_i + \sum_{u' \in \mathcal{N}_i} p_{iu'}\mathbf{e}_i\Big). \tag{12}$$

Clearly, by setting $p_{ui'}$ and $p_{u'i}$ as $1/\sqrt{|\mathcal{N}_u|}$ and 0 separately, we can exactly recover SVD++ model. Moreover, another widely-used item-based CF model, FISM [16], can be also seen as a special case of NGCF, wherein $p_{iu'}$ in Equation (12) is set as 0.

2.5.2 ***Time Complexity Analysis.*** As we can see, the layer-wise propagation rule is the main operation. For the l-th propagation layer, the matrix multiplication has computational complexity $O(|\mathcal{R}^+|d_l d_{l-1})$, where $|\mathcal{R}^+|$ denotes the number of nonzero entires in the Laplacian matrix; and d_l and d_{l-1} are the current and previous transformation size. For the prediction layer, only the inner product is involved, for which the time complexity of the whole training epoch is $O(\sum_{l=1}^{L}|\mathcal{R}^+|d_l)$. Therefore, the overall complexity for evaluating NGCF is $O(\sum_{l=1}^{L}|\mathcal{R}^+|d_l d_{l-1} + \sum_{l=1}^{L}|\mathcal{R}^+|d_l)$. Empirically, under the same experimental settings (as explained in Section 4), MF and NGCF cost around $20s$ and $80s$ per epoch on Gowalla dataset for training, respectively; during inference, the time costs of MF and NGCF are $80s$ and $260s$ for all testing instances, respectively.

3 RELATED WORK

We review existing work on model-based CF, graph-based CF, and graph neural network-based methods, which are most relevant with this work. Here we highlight the differences with our NGCF.

3.1 Model-based CF Methods

Modern recommender systems [5, 14, 33] parameterize users and items by vectorized representations and reconstruct user-item interaction data based on model parameters. For example, MF [20, 26] projects the ID of each user and item as an embedding vector, and conducts inner product between them to predict an interaction. To enhance the embedding function, much effort has been devoted to incorporate side information like item content [2, 30], social relations [34], item relations [37], user reviews [3], and external knowledge graph [32, 35]. While inner product can force user and item embeddings of an observed interaction close to each other, its linearity makes it insufficient to reveal the complex and nonlinear relationships between users and items [14, 15]. Towards this end, recent efforts [11, 14, 15, 36] focus on exploiting deep learning techniques to enhance the interaction function, so as to capture the nonlinear feature interactions between users and items. For instance, neural CF models, such as NeuMF [14], employ nonlinear neural networks as the interaction function; meanwhile, translation-based CF models, such as LRML [28], instead model the interaction strength with Euclidean distance metrics.

Despite great success, we argue that the design of the embedding function is insufficient to yield optimal embeddings for CF, since the CF signals are only implicitly captured. Summarizing these methods, the embedding function transforms the descriptive features (*e.g.*, ID and attributes) to vectors, while the interaction function serves as a similarity measure on the vectors. Ideally, when user-item interactions are perfectly reconstructed, the transitivity property of behavior similarity could be captured. However, such transitivity effect showed in the Running Example is not explicitly encoded, thus there is no guarantee that the indirectly connected users and items are close in the embedding space. Without an explicit encoding of the CF signals, it is hard to obtain embeddings that meet the desired properties.

3.2 Graph-Based CF Methods

Another line of research [12, 24, 40] exploits the user-item interaction graph to infer user preference. Early efforts, such as ItemRank [7] and BiRank [12], adopt the idea of label propagation to capture the CF effect. To score items for a user, these methods define the labels as her interacted items, and propagate the labels on the graph. As the recommendation scores are obtained based on the structural reachness (which can be seen as a kind of similarity) between the historical items and the target item, these methods essentially belong to neighbor-based methods. However, these methods are conceptually inferior to model-based CF methods, since there lacks model parameters to optimize the objective function of recommendation.

The recently proposed method HOP-Rec [40] alleviates the problem by combining graph-based with embedding-based method. It first performs random walks to enrich the interactions of a user with multi-hop connected items. Then it trains MF with BPR

Table 1: Statistics of the datasets.

Dataset	#Users	#Items	#Interactions	Density
Gowalla	29, 858	40, 981	1, 027, 370	0.00084
Yelp2018*	31, 668	38, 048	1, 561, 406	0.00130
Amazon-Book	52, 643	91, 599	2, 984, 108	0.00062

objective based on the enriched user-item interaction data to build the recommender model. The superior performance of HOP-Rec over MF provides evidence that incorporating the connectivity information is beneficial to obtain better embeddings in capturing the CF effect. However, we argue that HOP-Rec does not fully explore the high-order connectivity, which is only utilized to enrich the training data[1], rather than directly contributing to the model's embedding function. Moreover, the performance of HOP-Rec depends heavily on the random walks, which require careful tuning efforts such as a proper setting of decay factor.

3.3 Graph Convolutional Networks

By devising a specialized graph convolution operation on user-item interaction graph (*cf.* Equation (3)), we make NGCF effective in exploiting the CF signal in high-order connectivities. Here we discuss existing recommendation methods that also employ graph convolution operations [29, 42, 43].

GC-MC [29] applies the graph convolution network (GCN) [18] on user-item graph, however it only employs one convolutional layer to exploit the direct connections between users and items. Hence it fails to reveal collaborative signal in high-order connectivities. PinSage [42] is an industrial solution that employs multiple graph convolution layers on item-item graph for Pinterest image recommendation. As such, the CF effect is captured on the level of item relations, rather than the collective user behaviors. SpectralCF [43] proposes a spectral convolution operation to discover all possible connectivity between users and items in the spectral domain. Through the eigen-decomposition of graph adjacency matrix, it can discover the connections between a user-item pair. However, the eigen-decomposition causes a high computational complexity, which is very time-consuming and difficult to support large-scale recommendation scenarios.

4 EXPERIMENTS

We perform experiments on three real-world datasets to evaluate our proposed method, especially the embedding propagation layer. We aim to answer the following research questions:

- **RQ1**: How does NGCF perform as compared with state-of-the-art CF methods?
- **RQ2**: How do different hyper-parameter settings (*e.g.,* depth of layer, embedding propagation layer, layer-aggregation mechanism, message dropout, and node dropout) affect NGCF?
- **RQ3**: How do the representations benefit from the high-order connectivity?

4.1 Dataset Description

To evaluate the effectiveness of NGCF, we conduct experiments on three benchmark datasets: Gowalla, Yelp2018*[2], and Amazon-book, which are publicly accessible and vary in terms of domain, size, and sparsity. We summarize the statistics of three datasets in Table 1.

Gowalla: This is the check-in dataset [21] obtained from Gowalla, where users share their locations by checking-in. To ensure the quality of the dataset, we use the 10-core setting [10], *i.e.,* retaining users and items with at least ten interactions.

Yelp2018*: This dataset is adopted from the 2018 edition of the Yelp challenge. Wherein, the local businesses like restaurants and bars are viewed as the items. We use the same 10-core setting in order to ensure data quality.

Amazon-book: Amazon-review is a widely used dataset for product recommendation [9]. We select Amazon-book from the collection. Similarly, we use the 10-core setting to ensure that each user and item have at least ten interactions.

For each dataset, we randomly select 80% of historical interactions of each user to constitute the training set, and treat the remaining as the test set. From the training set, we randomly select 10% of interactions as validation set to tune hyper-parameters. For each observed user-item interaction, we treat it as a positive instance, and then conduct the negative sampling strategy to pair it with one negative item that the user did not consume before.

4.2 Experimental Settings

4.2.1 ***Evaluation Metrics.*** For each user in the test set, we treat all the items that the user has not interacted with as the negative items. Then each method outputs the user's preference scores over all the items, except the positive ones used in the training set. To evaluate the effectiveness of top-K recommendation and preference ranking, we adopt two widely-used evaluation protocols [14, 40]: recall@K and ndcg@K[3]. By default, we set $K = 20$. We report the average metrics for all users in the test set.

4.2.2 ***Baselines.*** To demonstrate the effectiveness, we compare our proposed NGCF with the following methods:

- **MF** [26]: This is matrix factorization optimized by the Bayesian personalized ranking (BPR) loss, which exploits the user-item direct interactions only as the target value of interaction function.
- **NeuMF** [14]: The method is a state-of-the-art neural CF model which uses multiple hidden layers above the element-wise and concatenation of user and item embeddings to capture their non-linear feature interactions. Especially, we employ two-layered plain architecture, where the dimension of each hidden layer keeps the same.
- **CMN** [5]: It is a state-of-the-art memory-based model, where the user representation attentively combines the memory slots of neighboring users via the memory layers. Note that the first-order connections are used to find similar users who interacted with the same items.
- **HOP-Rec** [40]: This is a state-of-the-art graph-based model, where the high-order neighbors derived from random walks are exploited to enrich the user-item interaction data.
- **PinSage** [42]: PinSage is designed to employ GraphSAGE [8] on item-item graph. In this work, we apply it on user-item

[1]The enriched trained data can be seen as a regularizer to the original training.
[2]In the previous version of yelp2018, we did not filter out cold-start items in the testing set, and hence we rerun all methods.

[3]The previous implementation of ndcg metric in NGCF follows the codes [31], but we find it is slightly different from the standard definition, although reflecting the similar trends. We re-implement the ndcg metric and rerun all methods

Table 2: Overall Performance Comparison.

	Gowalla		Yelp2018*		Amazon-Book	
	recall	ndcg	recall	ndcg	recall	ndcg
MF	0.1291	0.1109	0.0433	0.0354	0.0250	0.0196
NeuMF	0.1399	0.1212	0.0451	0.0363	0.0258	0.0200
CMN	<u>0.1405</u>	<u>0.1221</u>	0.0457	0.0369	0.0267	0.0218
HOP-Rec	0.1399	0.1214	<u>0.0517</u>	<u>0.0428</u>	<u>0.0309</u>	<u>0.0232</u>
GC-MC	0.1395	0.1204	0.0462	0.0379	0.0288	0.0224
PinSage	0.1380	0.1196	0.0471	0.0393	0.0282	0.0219
NGCF-3	**0.1569***	**0.1327***	**0.0579***	**0.0477***	**0.0337***	**0.0261***
%Improv.	11.68%	8.64%	11.97%	11.29%	9.61%	12.50%
p-value	2.01e-7	3.03e-3	5.34e-3	4.62e-4	3.48e-5	1.26e-4

interaction graph. Especially, we employ two graph convolution layers as suggested in [42], and the hidden dimension is set equal to the embedding size.

- **GC-MC** [29]: This model adopts GCN [18] encoder to generate the representations for users and items, where only the first-order neighbors are considered. Hence one graph convolution layer, where the hidden dimension is set as the embedding size, is used as suggested in [29].

We also tried SpectralCF [43] but found that the eigen-decomposition leads to high time cost and resource cost, especially when the number of users and items is large. Hence, although it achieved promising performance in small datasets, we did not select it for comparison. For fair comparison, all methods optimize the BPR loss as shown in Equation (11).

4.2.3 ***Parameter Settings.*** We implement our NGCF model in Tensorflow. The embedding size is fixed to 64 for all models. For HOP-Rec, we search the steps of random walks in $\{1, 2, 3\}$ and tune the learning rate in $\{0.025, 0.020, 0.015, 0.010\}$. We optimize all models except HOP-Rec with the Adam optimizer, where the batch size is fixed at 1024. In terms of hyperparameters, we apply a grid search for hyperparameters: the learning rate is tuned amongst $\{0.0001, 0.0005, 0.001, 0.005\}$, the coefficient of L_2 normalization is searched in $\{10^{-5}, 10^{-4}, \cdots, 10^{1}, 10^{2}\}$, and the dropout ratio in $\{0.0, 0.1, \cdots, 0.8\}$. Besides, we employ the node dropout technique for GC-MC and NGCF, where the ratio is tuned in $\{0.0, 0.1, \cdots, 0.8\}$. We use the Xavier initializer [6] to initialize the model parameters[4]. Moreover, early stopping strategy is performed, *i.e.,* premature stopping if recall@20 on the validation data does not increase for 50 successive epochs. To model the CF signal encoded in third-order connectivity, we set the depth of NGCF L as three. Without specification, we show the results of three embedding propagation layers, node dropout ratio of 0.0, and message dropout ratio of 0.1.

4.3 Performance Comparison (RQ1)

We start by comparing the performance of all the methods, and then explore how the modeling of high-order connectivity improves under the sparse settings.

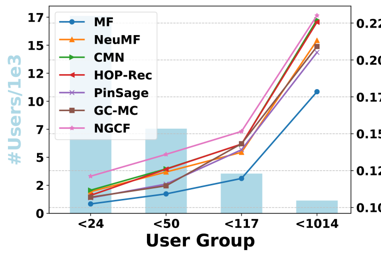
4.3.1 ***Overall Comparison.*** Table 2 reports the performance comparison results. We have the following observations:

- MF achieves poor performance on three datasets. This indicates that the inner product is insufficient to capture the complex relations between users and items, further limiting the performance. NeuMF consistently outperforms MF across all cases, demonstrating the importance of nonlinear feature interactions between user and item embeddings. However, neither MF nor NeuMF explicitly models the connectivity in the embedding learning process, which could easily lead to suboptimal representations.
- Compared to MF and NeuMF, the performance of GC-MC verifies that incorporating the first-order neighbors can improve the representation learning. However, in Yelp2018*, GC-MC underperforms NeuMF *w.r.t.* ndcg@20. The reason might be that GC-MC fails to fully explore the nonlinear feature interactions between users and items.
- CMN generally achieves better performance than GC-MC in most cases. Such improvement might be attributed to the neural attention mechanism, which can specify the attentive weight of each neighboring user, rather than the equal or heuristic weight used in GC-MC.
- PinSage slightly underperforms CMN in Gowalla and Amazon-Book, while performing much better in Yelp2018*; meanwhile, HOP-Rec generally achieves remarkable improvements in most cases. It makes sense since PinSage introduces high-order connectivity in the embedding function, and HOP-Rec exploits high-order neighbors to enrich the training data, while CMN considers the similar users only. It therefore points to the positive effect of modeling the high-order connectivity or neighbors.
- NGCF consistently yields the best performance on all the datasets. In particular, NGCF improves over the strongest baselines *w.r.t.* recall@20 by 11.68%, 11.97%, and 9.61% in Gowalla, Yelp2018*, and Amazon-Book, respectively. By stacking multiple embedding propagation layers, NGCF is capable of exploring the high-order connectivity in an explicit way, while CMN and GC-MC only utilize the first-order neighbors to guide the representation learning. This verifies the importance of capturing collaborative signal in the embedding function. Moreover, compared with PinSage, NGCF considers multi-grained representations to infer user preference, while PinSage only uses the output of the last layer. This demonstrates that different propagation layers encode different information in the representations. And the improvements over HOP-Rec indicate that explicit encoding CF in the embedding function can achieve better representations. We conduct one-sample t-tests and p-value < 0.05 indicates that the improvements of NGCF over the strongest baseline (underlined) are statistically significant.

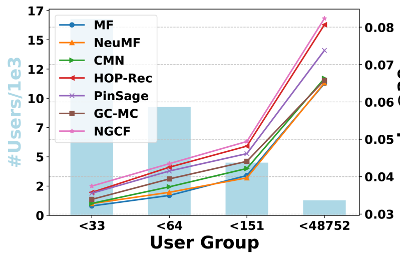
4.3.2 ***Performance Comparison w.r.t. Interaction Sparsity Levels.*** The sparsity issue usually limits the expressiveness of recommender systems, since few interactions of inactive users are insufficient to generate high-quality representations. We investigate whether exploiting connectivity information helps to alleviate this issue.

Towards this end, we perform experiments over user groups of different sparsity levels. In particular, based on interaction number per user, we divide the test set into four groups, each of which has the same total interactions. Taking Gowalla dataset as an example, the interaction numbers per user are less than 24, 50, 117, and 1014 respectively. Figure 4 illustrates the results *w.r.t.* ndcg@20 on

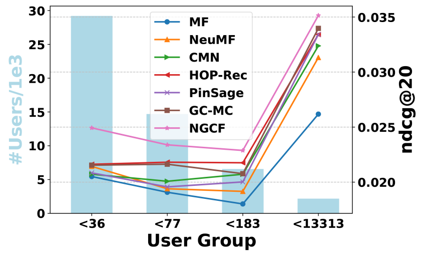
[4]We train MF from scratch, and use the trained MF embeddings to initialize NeuMF, GC-MC, PinSage, and NGCF to speed up and stabilize the training process. Since we update the implementation of BPR loss, we rerun the experiments.



(a) ndcg on Gowalla



(b) ndcg on Yelp2018*



(c) ndcg on Amazon-book

Figure 4: Performance comparison over the sparsity distribution of user groups on different datasets. Wherein, the background histograms indicate the number of users involved in each group, and the lines demonstrate the performance *w.r.t.* ndcg@20.

Table 3: Effect of embedding propagation layer numbers (L).

	Gowalla		Yelp2018*		Amazon-Book	
	recall	ndcg	recall	ndcg	recall	ndcg
NGCF-1	0.1556	0.1315	0.0543	0.0442	0.0313	0.0241
NGCF-2	0.1547	0.1307	0.0566	0.0465	0.0330	0.0254
NGCF-3	0.1569	0.1327	**0.0579**	**0.0477**	0.0337	0.0261
NGCF-4	**0.1570**	**0.1327**	0.0566	0.0461	**0.0344**	**0.0263**

different user groups in Gowalla, Yelp2018*, and Amazon-Book; we see a similar trend for performance *w.r.t.* recall@20 and omit the part due to the space limitation. We find that:

- NGCF and HOP-Rec consistently outperform all other baselines on all user groups. It demonstrates that exploiting high-order connectivity greatly facilitates the representation learning for inactive users, as the collaborative signal can be effectively captured. Hence, it might be promising to solve the sparsity issue in recommender systems, and we leave it in future work.
- Jointly analyzing Figures 4(a), 4(b), and 4(c), we observe that the improvements achieved in the first two groups (*e.g.,* 8.49% and 7.79% over the best baselines separately for < 24 and < 50 in Gowalla) are more significant than that of the others (*e.g.,* 1.29% for < 1014 Gowalla groups). It verifies that the embedding propagation is beneficial to the relatively inactive users.

4.4 Study of NGCF (RQ2)

As the embedding propagation layer plays a pivotal role in NGCF, we investigate its impact on the performance. We start by exploring the influence of layer numbers. We then study how the Laplacian matrix (*i.e.,* discounting factor p_{ui} between user u and item i) affects the performance. Moreover, we analyze the influences of key factors, such as node dropout and message dropout ratios. We also study the training process of NGCF.

4.4.1 ***Effect of Layer Numbers.*** To investigate whether NGCF can benefit from multiple embedding propagation layers, we vary the model depth. In particular, we search the layer numbers in the range of $\{1, 2, 3, 4\}$. Table 3 summarizes the experimental results, wherein NGCF-3 indicates the model with three embedding propagation layers, and similar notations for others. Jointly analyzing Tables 2 and 3, we have the following observations:

- Increasing the depth of NGCF substantially enhances the recommendation cases. Clearly, NGCF-2 and NGCF-3 achieve consistent improvement over NGCF-1 across all the board, which considers the first-order neighbors only. We attribute the improvement to the effective modeling of CF effect: collaborative user similarity and collaborative signal are carried by the second- and third-order connectivities, respectively.
- When further stacking propagation layer on the top of NGCF-3, we find that NGCF-4 leads to overfitting on Yelp2018* dataset. This might be caused by applying a too deep architecture might introduce noises to the representation learning. The marginal improvements on the other two datasets verifies that conducting three propagation layers is sufficient to capture the CF signal.
- When varying the number of propagation layers, NGCF is consistently superior to other methods across three datasets. It again verifies the effectiveness of NGCF, empirically showing that explicit modeling of high-order connectivity can greatly facilitate the recommendation task.

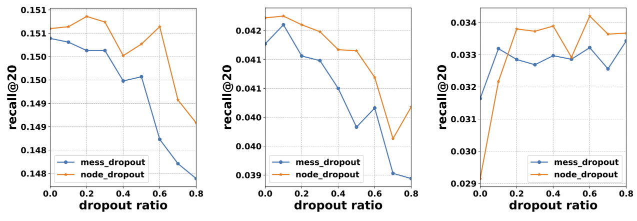
4.4.2 ***Effect of Embedding Propagation Layer and Layer-Aggregation Mechanism.*** To investigate how the embedding propagation (*i.e.,* graph convolution) layer affects the performance, we consider the variants of NGCF-1 that use different layers. In particular, we remove the representation interaction between a node and its neighbor from the message passing function (*cf.* Equation (3)) and set it as that of PinSage and GC-MC, termed NGCF-1_{PinSage} and NGCF-$1_{\text{GC-MC}}$ respectively. Moreover, following SVD++, we obtain one variant based on Equations (12), termed NGCF-$1_{\text{SVD++}}$. We show the results in Table 4 and have the following findings:

- NGCF-1 is consistently superior to all variants. We attribute the improvements to the representation interactions (*i.e.,* $\mathbf{e}_u \odot \mathbf{e}_i$), which makes messages being propagated dependent on the affinity between $\mathbf{e}_i$ and $\mathbf{e}_u$ and functions like the attention mechanism [2]. Whereas, all variants only take linear transformation into consideration. It hence verifies the rationality and effectiveness of our embedding propagation function.
- In most cases, NGCF-$1_{\text{SVD++}}$ underperforms NGCF-1_{PinSage} and NGCF-$1_{\text{GC-MC}}$. It illustrates the importance of messages passed by the nodes themselves and the nonlinear transformation.
- Jointly analyzing Tables 2 and 4, we find that, when concatenating all layers' outputs together, NGCF-1_{PinSage} and NGCF-$1_{\text{GC-MC}}$ achieve better performance than PinSage and GC-MC, respectively. This emphasizes the significance of layer-aggregation mechanism, which is consistent with [38].

4.4.3 ***Effect of Dropout.*** Following the prior work [29], we employ node dropout and message dropout techniques to prevent NGCF from overfitting. Figure 5 plots the effect of message dropout

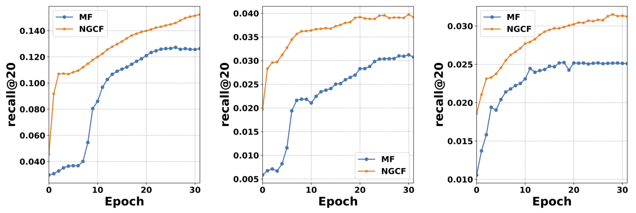
Table 4: Effect of graph convolution layers.

	Gowalla		Yelp2018*		Amazon-Book	
	recall	ndcg	recall	ndcg	recall	ndcg
NGCF-1	**0.1556**	**0.1315**	**0.0543**	**0.0442**	**0.0313**	**0.0241**
NGCF-$1_{\text{SVD++}}$	0.1517	0.1265	0.0504	0.0414	0.0297	0.0232
NGCF-$1_{\text{GC-MC}}$	0.1523	0.1307	0.0518	0.0420	0.0305	0.0234
NGCF-1_{PinSage}	0.1534	0.1308	0.0516	0.0420	0.0293	0.0231



(a) Gowalla (b) Yelp2018* (c) Amazon-Book

Figure 5: Effect of node dropout and message dropout ratios.



(a) Gowalla (b) Yelp2018* (c) Amazon-Book

Figure 6: Test performance of each epoch of MF and NGCF.

ratio p_1 and node dropout ratio p_2 against different evaluation protocols on different datasets.

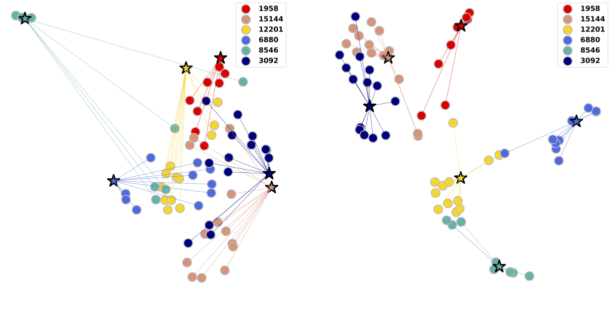
Between the two dropout strategies, node dropout offers better performance. Taking Gowalla as an example, setting p_2 as 0.2 leads to the highest recall@20 of 0.1514, which is better than that of message dropout 0.1506. One reason might be that dropping out all outgoing messages from particular users and items makes the representations robust against not only the influence of particular edges, but also the effect of nodes. Hence, node dropout is more effective than message dropout, which is consistent with the findings of prior effort [29]. We believe this is an interesting finding, which means that node dropout can be an effective strategy to address overfitting of graph neural networks.

4.4.4 *Test Performance w.r.t. Epoch*.

Figure 6 shows the test performance *w.r.t.* recall of each epoch of MF and NGCF. Due to the space limitation, we omit the performance *w.r.t.* ndcg which has the similar trend. We can see that, NGCF exhibits fast convergence than MF on three datasets. It is reasonable since indirectly connected users and items are involved when optimizing the interaction pairs in mini-batch. Such an observation demonstrates the better model capacity of NGCF and the effectiveness of performing embedding propagation in the embedding space.

4.5 Effect of High-order Connectivity (RQ3)

In this section, we attempt to understand how the embedding propagation layer facilitates the representation learning in the embedding space. Towards this end, we randomly selected six users from Gowalla dataset, as well as their relevant items. We observe how their representations are influenced *w.r.t.* the depth of NGCF.



(a) MF (NGCF-0) (b) NGCF-3

Figure 7: Visualization of the learned t-SNE transformed representations derived from MF and NGCF-3. Each star represents a user from Gowalla dataset, while the points with the same color denote the relevant items. Best view in color.

Figures 7(a) and 7(b) show the visualization of the representations derived from MF (*i.e.,* NGCF-0) and NGCF-3, respectively. Note that the items are from the test set, which are not paired with users in the training phase. There are two key observations:

- The connectivities of users and items are well reflected in the embedding space, that is, they are embedded into the near part of the space. In particular, the representations of NGCF-3 exhibit discernible clustering, meaning that the points with the same colors (*i.e.,* the items consumed by the same users) tend to form the clusters.
- Jointly analyzing the same users (*e.g.,* 12201 and 6880) across Figures 7(a) and 7(b), we find that, when stacking three embedding propagation layers, the embeddings of their historical items tend to be closer. It qualitatively verifies that the proposed embedding propagation layer is capable of injecting the explicit collaborative signal (via NGCF-3) into the representations.

5 CONCLUSION AND FUTURE WORK

In this work, we explicitly incorporated collaborative signal into the embedding function of model-based CF. We devised a new framework NGCF, which achieves the target by leveraging high-order connectivities in the user-item integration graph. The key of NGCF is the newly proposed embedding propagation layer, based on which we allow the embeddings of users and items interact with each other to harvest the collaborative signal. Extensive experiments on three real-world datasets demonstrate the rationality and effectiveness of injecting the user-item graph structure into the embedding learning process. In future, we will further improve NGCF by incorporating the attention mechanism [2] to learn variable weights for neighbors during embedding propagation and for the connectivities of different orders. This will be beneficial to model generalization and interpretability. Moreover, we are interested in exploring the adversarial learning [13] on user/item embedding and the graph structure for enhancing the robustness of NGCF.

This work represents an initial attempt to exploit structural knowledge with the message-passing mechanism in model-based CF and opens up new research possibilities. Specifically, there are many other forms of structural information can be useful for understanding user behaviors, such as the cross features [41] in context-aware and semantics-rich recommendation [22, 27], item knowledge graph [32], and social networks [34]. For example, by integrating item knowledge graph with user-item graph, we can establish knowledge-aware connectivities between users and items, which help unveil user decision-making process in choosing items. We hope the development of NGCF is beneficial to the reasoning of user online behavior towards more effective and interpretable recommendation.

Acknowledgement: This research is part of NExT++ research and also supported by the Thousand Youth Talents Program 2018. NExT++ is supported by the National Research Foundation, Prime Minister's Office, Singapore under its IRC@SG Funding Initiative.

REFERENCES

[1] Yixin Cao, Xiang Wang, Xiangnan He, Zikun Hu, and Tat-Seng Chua. 2019. Unifying Knowledge Graph Learning and Recommendation: Towards a Better Understanding of User Preferences. In *WWW*.

[2] Jingyuan Chen, Hanwang Zhang, Xiangnan He, Liqiang Nie, Wei Liu, and Tat-Seng Chua. 2017. Attentive Collaborative Filtering: Multimedia Recommendation with Item- and Component-Level Attention. In *SIGIR*. 335–344.

[3] Zhiyong Cheng, Ying Ding, Lei Zhu, and Mohan S. Kankanhalli. 2018. Aspect-Aware Latent Factor Model: Rating Prediction with Ratings and Reviews. In *WWW*. 639–648.

[4] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. In *NeurIPS*. 3837–3845.

[5] Travis Ebesu, Bin Shen, and Yi Fang. 2018. Collaborative Memory Network for Recommendation Systems. In *SIGIR*. 515–524.

[6] Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*. 249–256.

[7] Marco Gori and Augusto Pucci. 2007. ItemRank: A Random-Walk Based Scoring Algorithm for Recommender Engines. In *IJCAI*. 2766–2771.

[8] William L. Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *NeurIPS*. 1025–1035.

[9] Ruining He and Julian McAuley. 2016. Ups and Downs: Modeling the Visual Evolution of Fashion Trends with One-Class Collaborative Filtering. In *WWW*. 507–517.

[10] Ruining He and Julian McAuley. 2016. VBPR: Visual Bayesian Personalized Ranking from Implicit Feedback. In *AAAI*. 144–150.

[11] Xiangnan He and Tat-Seng Chua. 2017. Neural Factorization Machines for Sparse Predictive Analytics. In *SIGIR*. 355–364.

[12] Xiangnan He, Ming Gao, Min-Yen Kan, and Dingxian Wang. 2017. BiRank: Towards Ranking on Bipartite Graphs. *TKDE* 29, 1 (2017), 57–71.

[13] Xiangnan He, Zhankui He, Xiaoyu Du, and Tat-Seng Chua. 2018. Adversarial Personalized Ranking for Recommendation. In *SIGIR*. 355–364.

[14] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural Collaborative Filtering. In *WWW*. 173–182.

[15] Cheng-Kang Hsieh, Longqi Yang, Yin Cui, Tsung-Yi Lin, Serge J. Belongie, and Deborah Estrin. 2017. Collaborative Metric Learning. In *WWW*. 193–201.

[16] Santosh Kabbur, Xia Ning, and George Karypis. 2013. FISM: factored item similarity models for top-N recommender systems. In *KDD*. 659–667.

[17] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *ICLR*.

[18] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR*.

[19] Yehuda Koren. 2008. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *KDD*. 426–434.

[20] Yehuda Koren, Robert M. Bell, and Chris Volinsky. 2009. Matrix Factorization Techniques for Recommender Systems. *IEEE Computer* 42, 8 (2009), 30–37.

[21] Dawen Liang, Laurent Charlin, James McInerney, and David M. Blei. 2016. Modeling User Exposure in Recommendation. In *WWW*. 951–961.

[22] Zhenguang Liu, Zepeng Wang, Luming Zhang, Rajiv Ratn Shah, Yingjie Xia, Yi Yang, and Xuelong Li. 2017. FastShrinkage: Perceptually-aware Retargeting Toward Mobile Platforms. In *MM*. 501–509.

[23] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. 2013. Rectifier nonlinearities improve neural network acoustic models. In *ICML*.

[24] Athanasios N Nikolakopoulos and George Karypis. 2019. RecWalk: Nearly Uncoupled Random Walks for Top-N Recommendation. (2019).

[25] Jiezhong Qiu, Jian Tang, Hao Ma, Yuxiao Dong, Kuansan Wang, and Jie Tang. 2018. DeepInf: Social Influence Prediction with Deep Learning. In *KDD*. 2110–2119.

[26] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian Personalized Ranking from Implicit Feedback. In *UAI*. 452–461.

[27] Xuemeng Song, Fuli Feng, Xianjing Han, Xin Yang, Wei Liu, and Liqiang Nie. 2018. Neural Compatibility Modeling with Attentive Knowledge Distillation. In *SIGIR*. 5–14.

[28] Yi Tay, Luu Anh Tuan, and Siu Cheung Hui. 2018. Latent relational metric learning via memory-based attention for collaborative ranking. In *WWW*. 729–739.

[29] Rianne van den Berg, Thomas N. Kipf, and Max Welling. 2017. Graph Convolutional Matrix Completion. In *KDD*.

[30] Hao Wang, Naiyan Wang, and Dit-Yan Yeung. 2015. Collaborative Deep Learning for Recommender Systems. In *KDD*. 1235–1244.

[31] Jun Wang, Lantao Yu, Weinan Zhang, Yu Gong, Yinghui Xu, Benyou Wang, Peng Zhang, and Dell Zhang. 2017. IRGAN: A Minimax Game for Unifying Generative and Discriminative Information Retrieval Models. In *SIGIR*. 515–524.

[32] Xiang Wang, Xiangnan He, Yixin Cao, Meng Liu, and Tat-Seng Chua. 2019. KGAT: Knowledge Graph Attention Network for Recommendation. In *KDD*.

[33] Xiang Wang, Xiangnan He, Fuli Feng, Liqiang Nie, and Tat-Seng Chua. 2018. TEM: Tree-enhanced Embedding Model for Explainable Recommendation. In *WWW*. 1543–1552.

[34] Xiang Wang, Xiangnan He, Liqiang Nie, and Tat-Seng Chua. 2017. Item Silk Road: Recommending Items from Information Domains to Social Users. In *SIGIR*. 185–194.

[35] Xiang Wang, Dingxian Wang, Canran Xu, Xiangnan He, Yixin Cao, and Tat-Seng Chua. 2019. Explainable Reasoning over Knowledge Graphs for Recommendation. In *AAAI*.

[36] Yao Wu, Christopher DuBois, Alice X. Zheng, and Martin Ester. 2016. Collaborative Denoising Auto-Encoders for Top-N Recommender Systems. In *WSDM*. 153–162.

[37] Xin Xin, Xiangnan He, Yongfeng Zhang, Yongdong Zhang, and Joemon Jose. 2019. Relational Collaborative Filtering:Modeling Multiple Item Relations for Recommendation. In *SIGIR*.

[38] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. 2018. Representation Learning on Graphs with Jumping Knowledge Networks. In *ICML*, Vol. 80. 5449–5458.

[39] Feng Xue, Xiangnan He, Xiang Wang, Jiandong Xu, Kai Liu, and Richang Hong. 2019. Deep Item-based Collaborative Filtering for Top-N Recommendation. *TOIS* 37, 3 (2019), 33:1–33:25.

[40] Jheng-Hong Yang, Chih-Ming Chen, Chuan-Ju Wang, and Ming-Feng Tsai. 2018. HOP-rec: high-order proximity for implicit recommendation. In *RecSys*. 140–144.

[41] Xun Yang, Xiangnan He, Xiang Wang, Yunshan Ma, Fuli Feng, Meng Wang, and Tat-Seng Chua. 2019. Interpretable Fashion Matching with Rich Attributes. In *SIGIR*.

[42] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. 2018. Graph Convolutional Neural Networks for Web-Scale Recommender Systems. In *KDD (Data Science track)*. 974–983.

[43] Lei Zheng, Chun-Ta Lu, Fei Jiang, Jiawei Zhang, and Philip S. Yu. 2018. Spectral collaborative filtering. In *RecSys*. 311–319.