Group 2: Learning-Based Hashing with GNN for Recommender System

WENHE ZHANG, The University of California, USA
JIAYING LI, The University of California, USA
YU HOU, The University of California, USA
YUHAN SHAO, The University of California, USA

Recommender System plays a key role to alleviate explosive growth of online information. The key issue in recommender systems is to develop effective user and item representations from their interactions. Since many information in recommender systems is fundamentally structured as graphs, GNN has dominance and become widely used in graph representation learning for recommender systems. For collaborative filtering, the Graph Convolution Network (GCN) has become the new state-of-the-art method. LightGCN further simplify the design of GCN to make it more concise and appropriate for recommendation. Despite its correctness in learning embedding vectors for items in the user-item interaction network, inferring users' preferences in continuous embedding space has a high computational cost.

In this work, we introduced a new learning-based Hashing with GNN method - HashLGN which consists of two parts: a lightGCN encoder for learning node representations, a neural-network-based hash function for converting continuous vectors to binary hashcodes, as well as an unique novel decoding function for efficiently restoring inner product. Comprehensive experiments over publicly available MovieLens one million dataset demonstrate that our model not only outperforms the baseline methods, but also runs several times faster during inference.

Authors' addresses: Wenhe Zhang, The University of California, Los Angeles, USA, zwh990119@g.ucla.edu; Jiaying Li, The University of California, Los Angeles, USA, jl64@ucla.edu; Yu Hou, The University of California, Los Angeles, USA, yuhou316@g.ucla.edu; Yuhan Shao, The University of California, Los Angeles, USA, yuhan17@g.ucla.edu.

1 INTRODUCTION

Recommender systems have become an essential tool in our everyday lives, supporting a variety of online services such as web search and movie recommendation. Recommender systems infer users' preferences from user-item interactions or static features, and further recommend items that users might be interested in. The recommendation engine is meant to return a limited group of things from the database that users like in response to a query. A successful recommender system should not only generate effective recommendations, but also generate recommendations with high throughput. Some research on recommender systems nowadays focus more on generating accurate recommendations with high performance rather than high speed. However, high efficiency is also important in designing a successful recommendation system since recommendation systems sometimes need to fulfill a great deal of requests from users at the same time. In our work, by experimenting with GCN, hash and decoding functions, we designed a method that could give highly effective recommendations with high efficiency.

Graph neural network techniques have been utilized widely in recommendation systems. The most obvious reason is that GNN algorithms have been shown to be effective at learning representations for graph data in a variety of disciplines[1], and most data in recommendation has a graph structure. In our project, we propose to use GNN in the recommendation system and we use a more simple

and light-weight GNN design such as LightGCN[2] to achieve the efficiency requirement. GCN[4] approximates the first-order eigendecomposition of the graph Laplacian to iteratively aggregate information from neighbors. LightGCN simplify the design of GCN: it only includes the most essential component in GCN — neighborhood aggregation — for collaborative filtering.

Existing recommender systems typically generate high-dimensional, continuous embeddings for users and products and use their inner products to estimate their relevance and generate recommendations. However, finding top-K maximum inner products between two continuous high-dimension embeddings is quite challenging. To make the recommendations more efficient, we also use learning-to-hash functions in our project, which aims to learn a hash function to transform continuous vectors into binary and low-dimensional continuous vectors to accelerate the inner product search.

2 RELATED WORK

Many papers on GNN-based recommendation have been proposed in recent years. We divide the related works into three categories: User-Item Collaborative Filtering, Learning-to-hash, and Searching Method. The first one investigates the graph representation learning. The second part draws upon research in encoding objects into binary hash codes, and the last part presents the related work in similarity search.

User-Item Collaborative Filtering Besides the explorations on hashing-based representations, some researchers also delved into the utilization of subgraph structures of users to improve the embedding learning. The main concept of user-item collaborative filtering is to employ the items interacted by users to enhance user representations and utilize the users once engaged with items to enrich item representations, given the user-item interaction data. DGCF[9] pay special attention to user-item relationships at the finer granularity of user intents to disentangle these factors and yield disentangled representations. Inspired by Graph Convolution Network(GCN), some researchers proposed Neural Graph Collaborative Filtering(NGCF) model[8], which involves feature transformation, neighborhood aggregation, and nonlinear activation to refine graph embeddings. LightGCN[2] further simplifies the structure of NGCF and only incorporates neighborhood aggregation, the essential component in GCN, to learn the user and item embeddings by linearly propagating on user-item interaction graphs. It is more light-weight, but achieves better empirical performance than NGCF. Besides, HashGNN uses the equal weight in the neighbors aggregation. LightGCN uses degree normalization, which assigns weights to nodes based on the graph structure.

Learning-to-hash The major tasks of recommender systems are predicting whether users would interact with an item, and returning recommendations based on predicted user behavior. User-item interaction graphs are extensively used in this process. Traditional approaches represent the objects in user-item interaction graphs as continuous embedding vectors, but suffer from the computational costs of searching for similar continuous vectors. In recent years, more and more researchers are working on the hashing-based representation of graphs for recommender systems. For example, GHashing[5], a graphical neural network(GNN) based semantic hashing method, could produce hash indexes that enable graph lookup in constant time. HashGNN[7] proposed an effective framework for jointly learning continuous vectors and hash codes for graphs.

Searching Method Recommender systems not only need fit-for-purpose graph representations, but also involve similarity search for the top recommendations. Faiss[3], a library for efficient similarity search and clustering of dense vectors, was created to facilitate this process. Accelerating similarity search also helps the efficiency of recommender systems.

3 METHODS DESCRIPTION

3.1 LightGCN

As a subset of GNN, LightGCN is a simplfied version of GCN for implementing the recommendation system. Therefore, in our project, we used the light graph convolution (LGC) which is proposed in the LightGCN paper. [2] In the LightGCN model, they defined the graph convolution operation using the formula (1) (2).

$$e_u^{(k+1)} = \sum_{i \in N_u} \frac{1}{\sqrt{|N_u|}\sqrt{|N_i|}} e_i^{(k)} \tag{1}$$

$$e_i^{(k+1)} = \sum_{i \in N_i} \frac{1}{\sqrt{|N_i|}\sqrt{|N_u|}} e_u^{(k)} \tag{2}$$

where $\frac{1}{\sqrt{|N_u|}\sqrt{|N_i|}}$ is a symmetric normalization term which is designed for avoiding the embeddings increasing scale and this term has been proved to have a good performance. Hence, in our project, we would continue using this term. The Figure 1 would give a concrete illustration of the LightGCN model. In LightGCN, only the embeddings, e_u^0 (embeddings for all users) and e_i^0 (embeddings for all items) at the 0-th layer are trainable model parameters. After we get the embeddings from the 0-layer, we can follow the formula (3) (4) to calculate the embeddings at the higher layers.

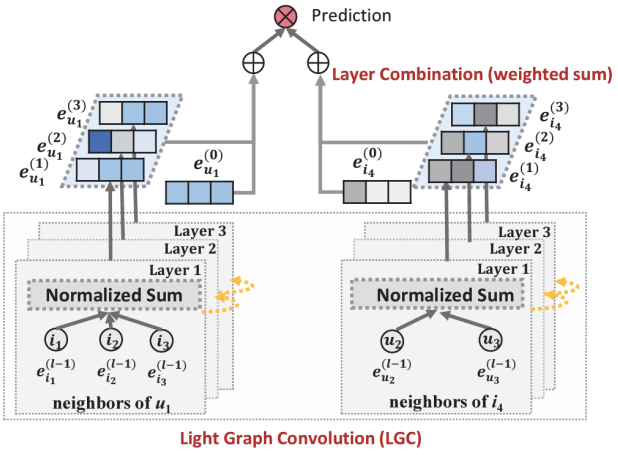

Fig. 1. LightGCN Architecture

$$e_u = \sum_{k=0}^{K} \alpha_k e_u^{(k)} \tag{3}$$

$$e_i = \sum_{k=0}^{K} \alpha_k e_i^{(k)} \tag{4}$$

where $\alpha_k \geq 0$ denotes the importance of the k-th layer embedding. From the authors' experience, setting $a_k = \frac{1}{(K+1)}$ would lead to the good performance. LightGCN used the inner product $\hat{y}_u^i = e_u^T e_i$ of user and item final representations as the prediction of the model. This inner product was also used as the ranking score of our recommendation system.

In order to better describe the LightGCN, we provide the following matrix representation of the model. Let matrix $\mathbf{R} \in \mathbb{R}^{M \times N}$ where M denotes number of users and N denotes number of items. And each entry of matrix R would be:

$$R_{ui} = \begin{cases} 1, & \text{if u interacts with item i} \\ 0, & \text{otherwise} \end{cases} \tag{5}$$

The adjacency matrix of the user-item graph would be $A = \begin{pmatrix} 0 & R \\ R^T & 0 \end{pmatrix}$. Denoting 0-th layer embedding matrix as $E^{(0)} \in \mathbb{R}^{(M+N) \times T}$ and T as the embedding size, we formulate the LightGCN as the formula 6.

$$E^{(k+1)} = (D^{-\frac{1}{2}} A D^{-\frac{1}{2}}) E^{(k)} \tag{6}$$

where D is a $(M+N) \times (M+N)$ diagonal matrix where each entry D_{ii} represents number of non-zero entries in the i-th row vector of the degree matrix A. Finally, we can formulate the final embedding matrix for the model prediction as formula 7.

$$\begin{aligned} E &= \alpha_0 E^0 + \alpha_1 E^1 + \alpha_2 E^2 + \ldots + \alpha_K E^K \\ &= \alpha_0 E^0 + \alpha_1 \tilde{A} E^1 + \alpha_2 \tilde{A}^2 E^2 + \ldots + \alpha_K \tilde{A}^K E^K \end{aligned} \tag{7}$$

where $\tilde{A} = D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$ is a symmetrically normalized matrix.

3.2 Learning-based Hashing

In our project, we reused the idea of learnable hash and decoding function [7] by defining a hash function $h(\mathbf{.}) : R^M \longrightarrow \{\pm 1\}^d$ and a decoding function $\Omega : \{\pm 1\}^d \times R^m \longrightarrow R$ where $\Omega(h(o); q) \approx q^T o$. **o** denotes the data (item) vector and **q** denotes the query (user) vector. [6]

Hash Function We implemented our hashing function by a multi-layer perceptron (MLP) which consists of 3 fully connected layers. In the last layer, we used sign function as an activation function for generating the binary outputs.

Decoding Function We found that the most common way to use the embeddings is using their inner product (IP) to predict the relevance objects. However, this implementation has some limitations. First of all, a d-dimensional hashcode can only represent 2^d distinct codes at most. Secondly, the norms of all the hashcodes are identical, which are always $\sqrt{d}$. Next, the IP of two hashcodes are integers in [-d, d] and each hashcode always similar to itself at most. Therefore, we want to find a decoding function to estimate $q^T o$ using q and o's hashcode $c = h(o)$. In order to find such a decoding function, we need a reweighting function for hashcodes and a compression function for query vectors. Firstly, we defined a learnable reweighting function $g(c) = \alpha \odot max(c, 0) + \beta \odot min(c, 0)$ where $\alpha, \beta \in R_+^d$ are learnable with all entries non-negative. Notation $\odot$ is an element-wise multiplication. The function $g(c)$ would map a discrete code c into a continuous d-dimensional vector. Finally, we defined a compression function $\Omega(c; q) = f(q)^T g(c)$ where $f(q)$ is the compressed vector of the query (user) vector q and $g(c)$ is the reweighted vector for o's hashcode c. This compression function would approximate the inner product of query (user) vector q and data (item) vector o.

3.3 Combined Model: HashLGN

In this work, we aim to combine the strength of LightGCN in learning continuous embeddings and the power of hash code in facilitating similarity search. To achieve such a combination, we fed the continuous embeddings learned by LightGCN into the aforementioned hash and decoding function, as shown in Figure 2. The continuous embeddings of items were converted into hash codes by the hash function, and then processed by the decoding function for reweighting. The continuous embeddings of users were directly fed into the decoding function for compression. Finally, the decoded embeddings of items and users were passed into Faiss for similarity search. The LightGCN model, hash function, and decoding function were optimized jointly during training.

3.4 BPR Loss for training

We deployed Bayesian Personalized Ranking(BPR) loss for training the combined model of LightGCN, hash function, and decoding function. BPR loss is a pairwise loss which encourages the prediction of an observed entry to be higher than its unobserved part. In this work, we could formulate the BPR loss by formula 8, where $\tilde{y}_{ui}$ represents the prediction of items that a user u has interacted with, $\tilde{y}_{uj}$ represents the prediction of items that a user u has never interacted with, $E^{(0)}$ denotes the embeddings at LightGCN's 0-th layer, and λ controls the weight of L2 regularization.

$$L_{BPR} = -\sum_{u=1}^{M} \sum_{i \in N_u} \sum_{j \notin N_u} \ln \sigma(\tilde{y}_{ui} - \tilde{y}_{uj}) + \lambda ||E^{(0)}||^2 \tag{8}$$

4 EXPERIMENTS DESIGN AND EVALUATION

4.1 Experiments Design

4.1.1 Data Explanation and Preprocessing.
The datasets used in related work include the recommender system datasets from Netflix, Yahoo, Amazon, MovieLens, Gowalla, Pinterest, and Alibaba. Most of the datasets are in the form of user ratings associated with items. Considering the data availability and our computational resources, we

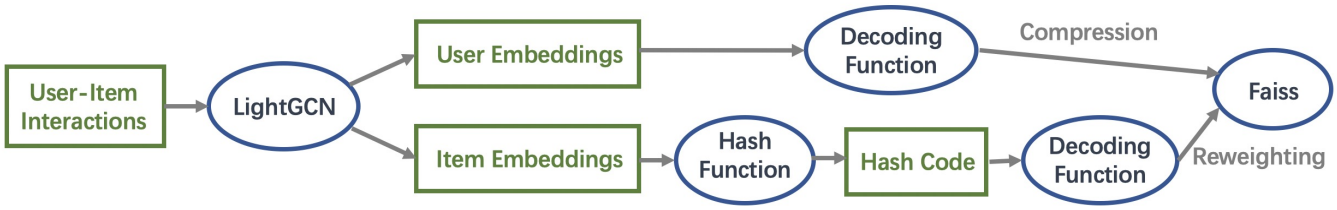

Fig. 2. The Combined Model of LightGCN, hash function and decoding function

would like to conduct our experiments on the MovieLens dataset, which include ratings of 6040 users and 1000209 user-item interactions in total.
We chose 80 percent of each user's historical interactions at random to form the training set for each dataset, and the rest is treated as the test set.

4.1.2 *Precision, Recall, F1, NDCG.*

Precision, Recall and F1 are widely used to evaluate the accuracy of top-K recommendation. Precision@K is a metric that determines what percentage of the recommended K items the user will click.

$$Precision@K(u) = \frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} \frac{|R^K(u) \cap T(u)|}{K} \quad (9)$$

Recall@K measures the proportion of the number of user clicks among the recommended K items to the entire click set.

$$Recall@K(u) = \frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} \frac{|R^K(u) \cap T(u)|}{|T(u)|} \quad (10)$$

F1@K is the combination of Precision@K and Recall@K.

$$F1@K(u) = \frac{2 \times Precision@K(u) \times Recall@K}{Precision@K(u) + Recall@K} \quad (11)$$

The contributions of the appropriately recommended items are differentiated by NDCG depending on their ranking positions.

$$NDCG@K(u) = \frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} \frac{\sum_{k=1}^{K} \frac{I(R_k^K(u) \in T(u))}{log(k+1)}}{\sum_{k=1}^{K} \frac{1}{log(k+1)}} \quad (12)$$

The average search time measures the average search time for all 6040 users.

4.1.3 *TopK and Threshold.*

There are two popular inner product search methods, which are maximum inner product search (MIPS) and threshold inner product search (TIPS). MIPS returns the vectors with the top-K largest IP values, while TIPS returns all vectors with IP values larger than a given threshold.

4.2 Evaluation

In contrast with our combined model, there are two baselines in the experiment: HashGNN and LightGCN. Both of the baseline approaches incorporated Faiss in their search phase. We tested the recommendation performance of two baselines and the combined model on the MovieLens dataset. We did the experiments using maximum inner product search (MIPS) and threshold inner product search (TIPS) separately. The performance comparison for MIPS is shown in Table 1 and the performance comparison for TIPS is showsn in Table 2.

From the experiment results in table 1 and table 2, we can see that our proposed HashLGN framework has the lowest average search time compared to the other two baselines in all the experiments.

Compared with the HashGNN baseline, our proposed HashLGN framework has better f1 score with much faster average search time. Our search time is two times faster than the HashGNN framework.

Method	Precision		Recall		F1 Score		NDCG		Avg Search Time	
	Top50	Top100	Top50	Top100	Top50	Top100	Top50	Top100	Top50	Top100
HashGNN	**22.25%**	13.96%	28.91%	34.26%	0.2023	0.1634	**0.5930**	**0.6578**	$\sim 31.5s$	$\sim 28s$
LightGCN	20.87%	**15.22%**	**41.14%**	**55.58%**	**0.2762**	**0.2384**	0.3982	0.4415	$\sim 14.1734s$	$\sim 14.6762s$
HashLGN	16.76%	12.82%	31.51%	45.85%	0.2181	0.1998	0.2931	0.3384	$\sim$ **13.5427s**	$\sim$ **13.7535s**

Table 1. Performance comparison of two baselines and the HashLGN model on topK-based search.

Method	Precision	Recall	F1 Score	NDCG	Threshold	Avg Search Time
HashGNN	15.90%	29.98%	0.1594	**0.6582**	5	$\sim 24.51s$
LightGCN	**21.18%**	45.23%	**0.2866**	0.0913	6	$\sim 22.73s$
HashLGN	14.57%	**46.65%**	0.2201	0.1317	4	$\sim$ **12.07s**

Table 2. Performance comparison of two baselines and the HashLGN model on threshold-based search.

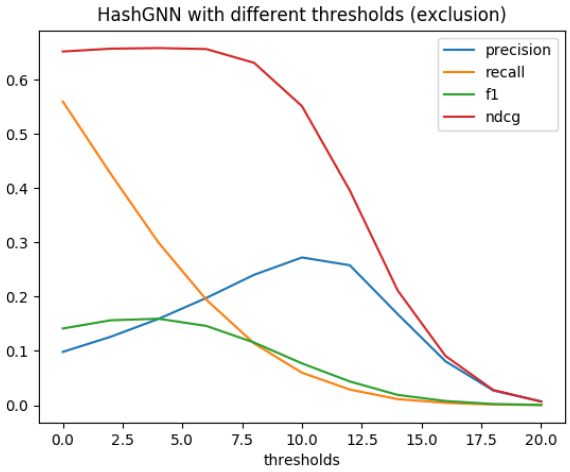

Fig. 3. HashGNN performance with different threholds

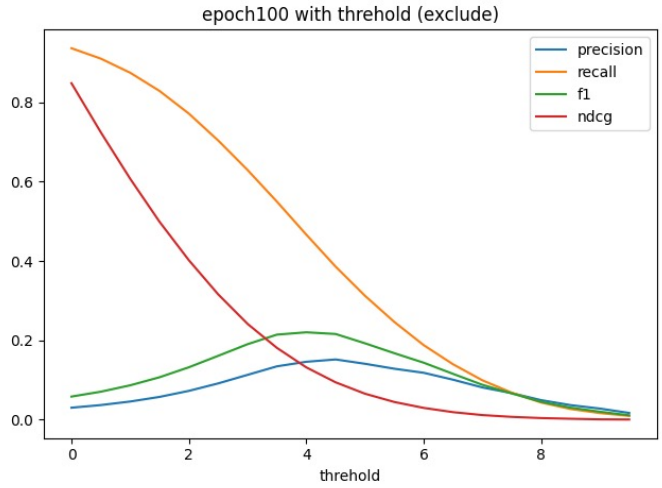

Fig. 4. HashLGN performance with different threholds

At the same time, our HashLGN has better performance than the HashGNN baseline. From the comparison in Fig 3 and Fig 4, we can see that HashLGN framework has higher f1 score and NDCG score than the HashGNN baseline.

Compared with LightGCN baseline, even though our proposed HashLGN framework have a slight downgrade in the performance, we have a much higher efficiency than the LightGCN baseline. We can see that in table 2, our HashLGN framework's average search time among all the thresholds is 10 seconds faster than the LightGCN baseline. We also provided a search time comparison between LightGCN baseline and HashLGN with different thresholds. We can see that our proposed HashLGN framework has lower average search time among all the threholds, which indicates that our HashLGN framework has much higher efficiency.

5 FUTURE WORK

We believe there are many future efforts that could be taken to improve the efficiency and effectiveness of recommender system with GNN. Our work calls for a better exclusion method that could rule out

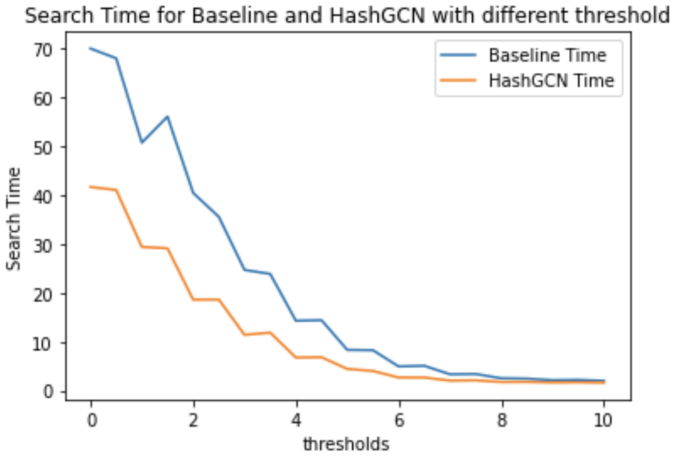

Fig. 5. search time comparison for LightGCN baseline and HashLGN

trained items during searching phase with lower time complexity. In our current work, we excluded the trained items by iterating through all the recommended items after searching. Future work may figure out a method to exclude trained items before searching and avoid iterations on all items.

Future work could also try different combinations of hash functions and GNN models, which may lead to higher efficiency and comparable effectiveness. Using different forms of loss may also help optimize the framework. Furthermore, examine the HashLGN framework on more datasets could give us a better understanding of the effect of combining hash-based representations with graph-based encoders.

6 CONCLUSION

In this work, we proposed a new framework called HashLGN which consists of two parts: a lightGCN encoder for learning node representations, and a hash layer with encoder and decoder referenced from the LH-TIPS[6] method. Comprehensive experiments demonstrate that our model can achieve comparable performance with the state-of-art models but with much higher efficiency.

7 TASK DISTRIBUTION FORM

Task	People
1. Collecting and Preprocessing Data	All Group Members
2. Run the HashGNN Baseline	All Group Members
3. Run the LightGCN Baseline	All Group Members
4. Implement HashLGN Framework	All Group Members
5. Evaluating and Comparing Algorithms	All Group Members
6. Write the Report	All Group Members

REFERENCES

[1] Qipeng Guo, Xipeng Qiu, Xiangyang Xue, and Zheng Zhang. 2021. Syntax-guided text generation via Graph Neural Network. *Science China Information Sciences* 64, 5 (2021). https://doi.org/10.1007/s11432-019-2740-1

[2] Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, YongDong Zhang, and Meng Wang. 2020. *LightGCN: Simplifying and Powering Graph Convolution Network for Recommendation.* Association for Computing Machinery, New York, NY, USA, 639–648. https://doi.org/10.1145/3397271.3401063

[3] Jeff Johnson, Matthijs Douze, and Herve Jegou. 2021. Billion-scale similarity search with gpus. *IEEE Transactions on Big Data* 7, 3 (2021), 535–547. https://doi.org/10.1109/tbdata.2019.2921572

[4] Thomas N. Kipf and Max Welling. 2017. Semi-supervised classification with graph Convolutional Networks. https://arxiv.org/abs/1609.02907

[5] Zongyue Qin, Yunsheng Bai, and Yizhou Sun. 2020. GHashing: Semantic Graph Hashing for Approximate Similarity Search in Graph Databases. 2062–2072. https://doi.org/10.1145/3394486.3403257

[6] Zongyue Qin and Yizhou Sun. [n.d.]. Learning-based Hashing for Threshold Inner Product Search.

[7] Qiaoyu Tan, Ninghao Liu, Xing Zhao, Hongxia Yang, Jingren Zhou, and Xia Hu. 2020. Learning to hash with graph neural networks for recommender systems. *Proceedings of The Web Conference 2020* (2020). https://doi.org/10.1145/3366423.3380266

[8] 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.* ACM. https://doi.org/10.1145/3331184.3331267

[9] Xiang Wang, Hongye Jin, An Zhang, Xiangnan He, Tong Xu, and Tat-Seng Chua. 2020. Disentangled graph collaborative filtering. *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval* (2020). https://doi.org/10.1145/3397271.3401137