Simplifying Graph-based Collaborative Filtering for Recommendation

Anonymous Author(s)**

ABSTRACT

Graph Convolutional Networks (GCNs) are a popular type of machine learning models that use multiple layers of convolutional aggregation operations and non-linear activations to represent data. Recent studies apply GCNs to Collaborative Filtering (CF)-based recommender systems (RSs) by modeling user-item interactions as a bipartite graph and achieve superior performance. However, these models face difficulty in training with non-linear activations on large graphs. Besides, most GCN-based models could not model deeper layers due to the over-smoothing effect with the graph convolution operation. In this paper, we improve the GCN-based CF models from two aspects. First, we remove non-linearities to enhance recommendation performance, which is consistent with the theories in simple graph convolutional networks. Second, we obtain the initialization of the embedding for each node in the graph by computing the network embedding on the condensed graph, which alleviates the over smoothing problem in graph convolution aggregation operation with sparse interaction data. The proposed model is a linear model that is easy to train, scalable to large datasets, and shown to yield better efficiency and effectiveness on four real datasets.

CCS CONCEPTS

• **Information systems** → **Data mining**.

KEYWORDS

Collaborative Filtering, Recommendation, Embedding Propagation, Graph Convolutional Network

ACM Reference Format:

Anonymous Author(s). 2023. Simplifying Graph-based Collaborative Filtering for Recommendation. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

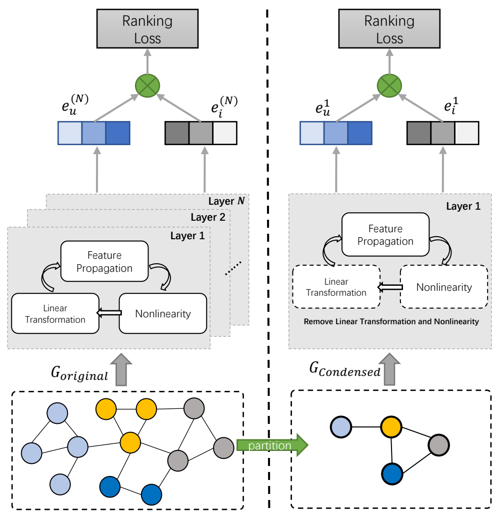

Figure 1: Illustrations of training of standard GCN (left) and SGCF (right). Standard GCN needs to recurrently perform N-layers message passing to get the final embeddings for training with a large-scale graph structure $G_{original}$. At the same time, SGCF only has one layer with a condensed graph $G_{condensed}$ and removes other operations like self-connection, feature transformation, and nonlinear activation, largely improving training efficiency and helping real deployment.

1 INTRODUCTION

Recommendation systems conduct personalized information to assist users in finding information of their interests and alleviate information overload. Collaborative filtering (CF) represents the techniques that learn user/item embeddings from their historical interactions and has been widely applied in various domains, such as online shopping and social media. Since the interactions can naturally be modeled as graphs, recent studies have leveraged Graph Convolutional Networks (GCNs) to learn node representations. GCN-based models can exploit higher-order connectivity between users and items and have achieved impressive recommendation performance. PinSage and M2GRL are examples of successful applications of GCNs in industrial applications.

Despite the promising performance, existing GCN-based CF models are becoming more sophisticated than ever, aiming to capture higher-order collaborative signals. Such complicated models are difficult to train with large graphs and bring efficiency and scalability challenges, which hinder their adoption in broad applications.

Moreover, it can be time-consuming for CF to train GCN-based models through message passing (i.e., neighborhood aggregation) on large graphs; and simplifications done by LightGCN [13] and SGC [34] do not help much. Until now, how to improve the efficiency of GCN models while retaining their effectiveness on recommendation is still an open problem.

We address the necessity of feature transformation and nonlinear activation in GCN-based recommendation, aiming to accelerate GCNs in propagation on large-scale datasets. Given that GCN-based CF models are burdensome with many operations unjustified, we derive the simplest linear model that could precede GCNs. To this end, we reduce the excess complexity of GCNs by repeatedly removing the non-linearities between GCN layers and collapsing the resulting function into a single linear transformation. Specifically, we devise a graph partition-based algorithm to generate a model that is easy to implement, train and aggregate the multi-layer node information efficiently on large graphs. We empirically show that the final linear model exhibits comparable or superior performance to GCNs on various tasks while being more computationally efficient and fitting significantly fewer parameters. We illustrate the above idea using a toy example in Figure 1.

We make the following contributions in this paper:

- We empirically reduce the excessive complexity of GCNs by repeatedly removing the nonlinearities between GCN layers and collapsing the resulting function into a single linear transformation.
- We propose SGCF, which largely simplifies the model design by including only the most essential components in GCN for more efficient recommendations. We offer an effective partition technique for reducing the scale of input graph structure to avoid infinite layers of explicit message passing for efficient recommendations.
- Our extensive experiments on four benchmark datasets show that SGCF achieves significant improvements over state-of-the-art GCN-based CF model. Notably, SGCF attains up to 10% improvement in NDCG@20 and more than 10x speed-up in training over our baselines on the Amazon-Books dataset. Due to anonymous requirements, the code link is invisible until paper acceptance.

2 PRELIMINARIES

Following SemiGCN [19], We define a graph as $\mathcal{G}$=< $\mathcal{V}$, **E** >, where $\mathcal{V}$ denotes the set of nodes and **E** denotes the edge e_{ij}between node i and node j. We use **A** to denote the adjacency matrix—$a_{ij} = 1$ if an edge exists from node i to node j; and $a_{ij} = 0$ otherwise. To ease Illustration, we use $\mathbf{A} = [i|a_{ij} = 1]$ to denote the one-hop set of nodes, $\tilde{\mathbf{A}} = \tilde{D}^{-\frac{1}{2}}\mathbf{A}\tilde{D}^{-\frac{1}{2}}$ the normalized adjacency matrix with added self loops, where $\tilde{D}$ is the degree matrix of $\tilde{\mathbf{A}}$. $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}_N$ is the adjacency matrix of the graph with added self-connections, where **I** is the identity matrix.

2.1 Graph Convolutional Networks

For each node $v \in \mathcal{V}$, we use e_i^0 to denote the node initial embedding, which is usually the feature vector x_i of node i, in which $e^0 = x_i$. In a graph $\mathcal{G}$, the main idea of GCNs is to stack L steps in a recursive message passing or feature propagation operation to learn node embedding [17]. Specifically, for each node i at the step, it is computed recursively with following three steps: feature propagation, feature transformation and non-linear transition.

Feature propagation For each node i, the feature aggregation operation aggregate the embeddings from graph neighbors $\mathcal{N}_i$ and its own embedding e_i^k at previous layer l. As the focus of this work is not to design more sophisticated feature aggregation function, we follow the widely used feature aggregation function proposed in Kipf et al. [19], which is empirically effective and has been adopted by many GCN variants:

$$\overline{\mathbf{H}}^{(k+1)} = \tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1/2}\mathbf{H}^k \tag{1}$$

where the features **H** at k-th layer, feature propagation output **H** layer can be regarded as the Laplacian smoothing on the features at previous layer.

Feature transformation and nonlinear transition After the local smoothing, a GCN layer is identical to a standard multi-layer perceptron (MLP). Each layer is associated with learned weight $\mathbf{W}^{(k)}$, and the smoothed hidden feature representations are transformed linearly. Finally, a nonlinear activation function such as ReLU(.) = max(0, ·) is applied pointwisely before outputting feature representation $\mathbf{H}^{(}k)$. In totally, the representation updating rule of the k-th layer is:

$$\mathbf{H}^{(k)} \leftarrow \mathrm{ReLU}\left(\overline{\mathbf{H}}^{(k)}\mathbf{W}^{(k)}\right) \tag{2}$$

The pointwise nonlinear transformation of the k-th layer is followed by the feature propagation of the $(k+1)$-th layer.

2.2 Graph Convolutional based Recommendation

In a recommender system, there are two sets of entities: a user set **U** with M users and an item set **I** with N items. As implicit feedback is the most common form in many recommender systems, we focus on implicit feedback based CF in this work, and it is easy to extend the proposed model for rating prediction in CF. Users show ratings to the items with a rating matrix $\mathbf{R} \in \mathbb{R}^{M\times N}$, with r_{ui}=1 denotes user u likes item i, otherwise it equals 0. The rating matrix is a key to the success of recommendation performance. With the huge success of GCNs, researchers attempted to formulate recommendation as a user-item bipartite graph, and adapted GCNs for recommendation. NGCF [32] are specifically designed under the CF settings. Given ratings of users to items, the user-item bipartite graph is denoted as $\mathcal{G}$=<**U** ∪ **I**, **A**>, with **A** is constructed from the rating matrix **R** as:

$$\mathbf{A} = \begin{bmatrix} \mathbf{R} & \mathbf{0}^{N\times M} \\ \mathbf{0}^{M\times N} & \mathbf{R}^{\mathrm{T}} \end{bmatrix} \tag{3}$$

Let $\mathbf{E} \in \mathbb{R}^{(M+N)\times D}$ denote the free embedding matrix of users and items. By feeding the free embedding matrix **E** into GCNs with bipartite graph $\mathcal{G}$, i.e., $\forall i \in \mathcal{U} \cup \mathcal{I}$, $h_i^0 = e_i$. Then, GCNs iteratively perform with embedding propagation step in Eq.(1) and nonlinear transformation with Eq.(2), each user's or item's embeddings can be updated in the iterative process. Therefore, the final embedding $\mathbf{H}^k$ explicitly injects the up to K-th order collective connections between users and items. All the parameters can be learned in an end-to-end framework.

2.3 Graph Partition Technique

A naive approach for the initialization of network embedding is by random, which assigns random numbers in $\mathbb{R}$ for the initial embedding of each node in the graph. However, this approach disregards the structure of the input graph, rendering it unsuitable for network embedding. Inspired by the graph partition base algorithm, we aim to describe the sketch of the input graph $\mathcal{G}$ = < $\mathcal{V}$, **E** > using the partitioning of $\mathcal{G}$, which are then processed as the initial embedding of each node in $\mathcal{V}$. A partitioning $\mathcal{P}$ of $\mathcal{G}$ divides $\mathcal{V}$ into k disjoint subsets, denoted by $\mathcal{P}$=$\mathcal{V}_1$, $\mathcal{V}_2$, ..., $\mathcal{V}_k$, where k is a user defined number. Given a node $v \in \mathcal{V}$, let $\mathcal{V}' \in \mathcal{P}$ be the partition where v resides, denoted by $p(v) = \mathcal{V}'$. We call the neighbors in the same partition are internal nodes, while the others are external nodes. Moreover, a node $v \in \mathcal{V}$ is a border node of $\mathcal{G}$, if v has at least one neighbor $n \in N(v)$ whose partition is different from the one of v, namely $p(v) \neq p(n)$. Let $\mathcal{V}_b$ be the set of border nodes of $\mathcal{G}$. The border sub-graph $\mathcal{G}_b$ with respect to $\mathcal{P}$ is the induced sub-graph of $\mathcal{G}$ constructed on $\mathcal{V}_b$.

3 METHOD

3.1 Overall Structure of Our Model

In this part, we propose Simple Graph Convolutional Collaborative Filtering with graph partition techniques which is a general GCN-based CF model for recommendation. The overall architecture of SGCF is shown in Figure 2. SGCF advances current GCN-based model with two characteristics: (a) At each layer of the feature propagation step, we use a simplified linear embedding propagation without any nonlinear activation and linear transformations; (b) for accelerating the network embedding and improve the performance of the algorithms on both effectiveness and efficiency, we propose a graph resizing technique to recursively partition a graph into several small-sized sub-graphs to capture the internal and external structural information of nodes, and then compute the network embedding with low-order propagation process in a condensed graph.

3.2 Simplified Embedding Propagation

In traditional MLP's, deeper layers allow for more expressive features because they allow for feature hierarchies, such as features in the second layer building on top of features in the first layer. In GCNs, the layers have another important function: in each layer the hidden representations are averaged among neighbors that are one hop away. This implies that after k-layers a node obtains feature information from all nodes that are k-hops away in the graph. This effect is similar to convolutional neural networks, where depth increases the receptive field of internal features [8]. Although convolutional networks can benefit substantially from the increased depth [15], typically MLPs obtain little benefit beyond 3 or 4 layers.

We hypothesize that the non-linearity between GCN layers is not critical - but that the majority of the benefit arises from the local averaging. We therefore remove the nonlinear transition functions between each layer.

Given the user-item bipartite graph as formulated in Eq.(3), let $\mathbf{E} \in \mathbb{R}^{(M+N)\times D}$ denote the free embeddings of users and items, with the first M rows of the matrix, i.e., $\mathbf{E}_{1:M}$ is the user embedding sub-matrix, and $\mathbf{E}_{M+1:M+N}$ is the item embedding sub-matrix. Then, our model takes the embedding matrix as input:

$$\mathbf{E}^0 = \mathbf{E} \tag{4}$$

which resembles the embedding based models in CF. Notably, different from GCN based tasks with node features as fixed input data, the embedding matrix is unknown and needs to be trained our model. Following the theoretical elegance with graph spectral connections and empirical competing results of SGC, at each iteration step $k+1$, we assume the embedding $\mathbf{E}^{k+1}$ is a nonlinear aggregation of the embedding matrix $\mathbf{E}^k$ at the previous layer k as:

$$\mathbf{E}^{k+1} = \mathbf{S}\mathbf{E}^k\mathbf{W}^k \tag{5}$$

where $\mathbf{S} = \tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1/2}$ denotes the normalized adjacency matrix with added self loop, $\mathbf{W}^k$ is the nonlinear transformation. Further, Eq.(5) with matrix form is equivalent to modeling each user u's and each item i's update embedding as:

$$\left[\mathbf{E}^{k+1}\right]_u = \mathbf{e}_u^{k+1} = \left[\frac{1}{d_u}\mathbf{e}_u^k + \sum_{j\in R_u}\frac{1}{d_j \times d_u}\mathbf{e}_j^k\right]\mathbf{W}^k \tag{6}$$

$$\left[\mathbf{E}^{k+1}\right]_i = \mathbf{e}_i^{k+1} = \left[\frac{1}{d_i}\mathbf{e}_i^k + \sum_{u\in R_i}\frac{1}{d_i \times d_u}\mathbf{e}_u^k\right]\mathbf{W}^k \tag{7}$$

which $d_i(d_u)$ is the diagonal degree of item i (user u) in the user-item bipartite graph $\mathcal{G}$. R_u (and R_i) is neighbors of node user or item in graph $\mathcal{G}$.

3.3 Model Prediction with Condensed Graph

With a predefined depth K, the nonlinear embedding propagation would stop at the K-th layer with output of the embedding matrix $\mathbf{E}^K$. For each user (item), $e_u^K (e_i^K)$ captures the up to K-th order bipartite graph similarity. Then, many embedding based recommendation models would predict the preference $\hat{y}_{ui}$ as the inner product between user and item latent vectors as:

$$\hat{y}_{ui} =< \mathbf{e}_u^K, \mathbf{e}_i^K > \tag{8}$$

which <,> denotes the vector inner product operation.

Most existing GCN based variants, as well as GCN based recommendation models, achieve the best performance with K=2 [9]. The overall trend for these GCN variants is that: (1) the performance increases as K increases from 0 to 1, (2) and drops quickly as K continues to increase. In fact, most recommended scenarios have large-scale input networks and the user-item graph will become more complicated. It will cause each node e_u or e_i has multiple neighbor hops ($K >= 2$). However, as k increases from 0 to K, the node embeddings at deeper layers tend to be over smoothed, i.e., they are more similar with less distinctive information. Meanwhile, stacking multiple layers of message passing likely introduces uninformative, noisy, or ambiguous relationships, which could largely affect the training efficiency and effectiveness. This problem not only exists in GCNs, but is much more severe in CF with very sparse user behavior data for model learning. To alleviate the problem, we utilize the graph partition techniques to reduce the scale of the input network and construct the condensed graph.

To construct the condensed graph $\mathcal{G}_c$=($\mathcal{V}_c$, $\mathcal{E}_c$), we first obtain a partitioning $\mathcal{P}$ of $\mathcal{G}$, denoted by $\mathcal{P} = \{\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_k\}$ where k is a user-defined number. The goal of graph partition is (k, σ)-balanced where $0 < \sigma < 1$, and it satisfies the constraint:

$$\max_{1\le i\le k} |\mathcal{V}_i| \le (1+\sigma)\left\lceil\frac{|\mathcal{V}|}{k}\right\rceil \tag{9}$$

and minimizes the size of edge-cut as:

$$\bigcup_{1\le i,j\le k}\left\{(v,u)\in \mathbf{E} \mid v\in\mathcal{V}_i, u\in\mathcal{V}_j\right\} \tag{10}$$

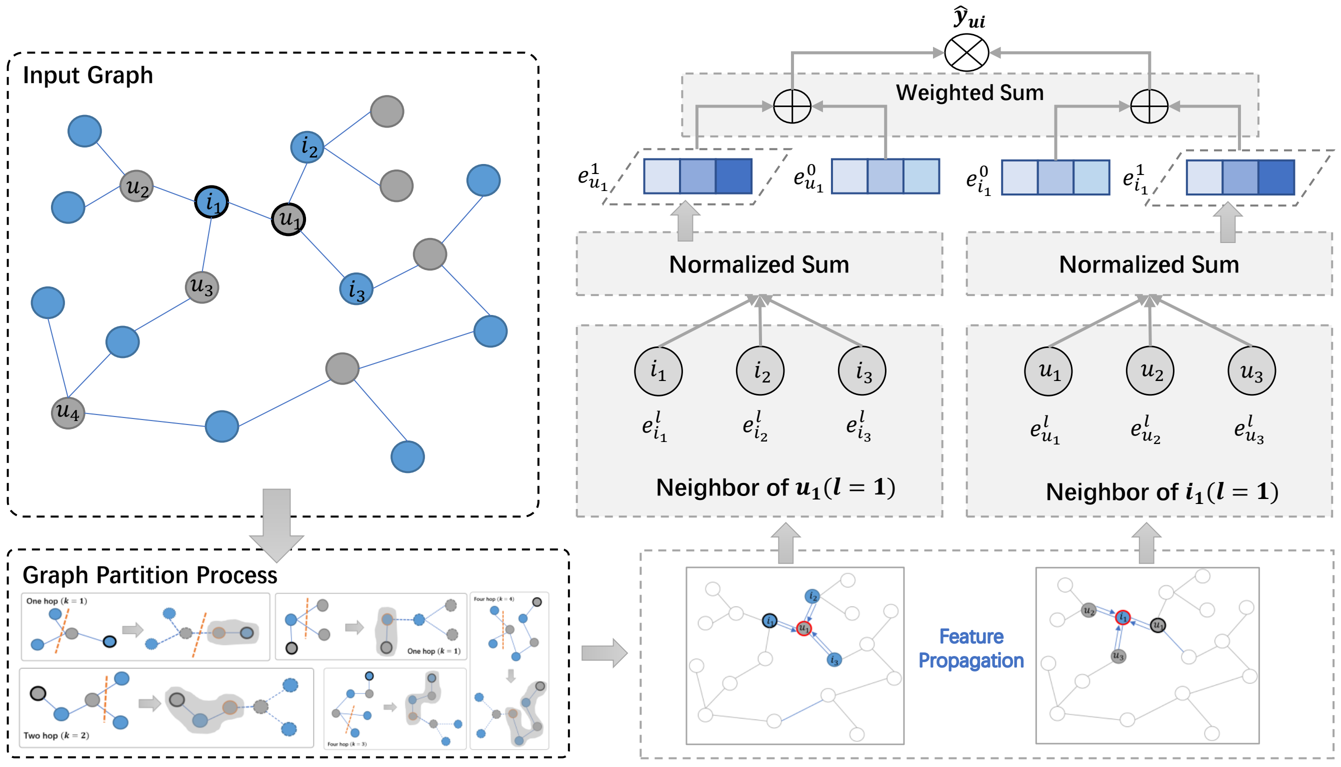

Figure 2: The overall architecture of our proposed mode. The graph process illustrates the procedure of embedding propagation with different hop. The partition algorithm works in several iterations with different hops k (left bottom). In each iteration the updating of the embedding of each node can be achieved in a k-layer computing framework. The final condensed graph feed into our simplified GCF model.

However, the (k, σ)-balanced graph partition is a NP-hard problem [2]. To deal with this issue, we are motivated by the GPA algorithm [20] for graph partitioning, which has adopted in practice and costs a running time complexity $O(|\mathcal{V}| + |\mathbf{E}| + klogk|)$. Based on $\mathcal{P}$, we construct the condensed graph $\mathcal{G}_c$ of $\mathcal{G}$ by creating an condensed node v_a for each sub-graph $\mathcal{V}' \in \mathcal{P}$ and connecting two condensed nodes v_a and u_a with an condensed edge (v_a, u_a) of a weight $w(v_a, u_a)$. Then, the number of condensed nodes in $\mathcal{G}_c$ is k, i.e., the number of partitions of $\mathcal{G}$. Besides, the number of condensed edges of $\mathcal{G}_c$ is bounded by the size of edge cut.

One crucial issue remaining is how to decide k. On one hand, if k is small, then one condensed node would be pertinent to a lot of nodes in the input graph $\mathcal{G}$. As such, the initial embedding of each node in $\mathcal{G}$ inherited from the corresponding abstract node would lose the power of effectiveness. On the other hand, if k is large, then the condensed graph $\mathcal{G}_c$ would be large too. Therefore, it would be highly expensive to compute the network embedding on $\mathcal{G}_c$, which increase the overall cost of the initialization phase. To reach a good balance, we set $k = \lceil \sqrt{|\mathcal{V}|} \rceil$, which is a sufficiently large number but much smaller than $|\mathcal{V}|$, that works well in practise.

In addition, to compute the condensed graph embedding of $\mathcal{G}_c$, a naive approach is to let the initial embedding of each node v equal the embedding of the corresponding condensed node $c(v)$. However, this approach would suffer from the issue where the nodes pertinent to the same condensed node have the same initial embeddings, rendering this method ineffective. For addressing this issue, we utilize a iterative approach where each node update its own embedding based on the embeddings of its neighbors until the convergence is reached. This means specifically, in each iteration, each node $v \in \mathcal{V}$ first aggregates the embeddings of v's neighbors, which results in the average embedding $e_{avg}(v)$. Then, we update v's embeddings as the aggregation of e_{avg} and its own embedding $e_i v$. The reason is that the embedding of each node should be close to its neighbors in the graph. Moreover, Algorithm 1 shows the procedure of embedding propagation. Consider a graph $\mathcal{G}$=< $\mathcal{V}, \mathbf{E}$ >, the condensed graph $\mathcal{G}_c$ of $\mathcal{G}$, and the network embedding e_c of $\mathcal{G}_c$.

Based on the above condensed input graph, we argue that: instead of directly utilizing the original user-item bipartite network, we perform the preference learning with condensed graph as: $\hat{y}_{ui} =< \mathbf{e}^k_u, \mathbf{e}^k_i >$. We hypothesize that it is easier to optimize the condensed rating, and the condensed graph learning could help to alleviate the over smoothing effect with deeper layers. Based on the condensed

Algorithm 1: Condensed Graph Propagate

```
  Data: Input graph $\mathcal{G} =< \mathcal{V}, \mathbf{E} >$, the embeddings $e_c$ of $\mathcal{G}_c$
        condensed graph, and the threshold $\delta$
  Result: The set $e_i$ of initial embedding of each node $v \in \mathcal{V}$
1 initialization: Let $e_i(v) = e_c((c_v))$ for each node $v \in \mathcal{V}$;
  while $\triangle > \delta$ do
2     for each node $v \in \mathcal{V}$ do
3         Let $e_{avg}(v) = \frac{1}{|N(v)|} \sum_{u \in N(v)} e_i(u)$ Compute
          $e_i'(v) = \frac{1}{2}\left(e_i(v) + e_{avg}(v)\right)$
4     end
5     Let $\triangle = \frac{1}{|V|} \sum_{v \in V} \left\| e_i'(v) - e_i(v) \right\|$;
6     For each node $v \in \mathcal{V}$, let $e_i(v) = e_i'(v)$
7 end
8 return $e_i$
```

preference prediction in above, we have:

$$\begin{aligned}\hat{y}_{ui} &= \hat{y}_{ui}^{k-1} + < \mathbf{e}_u^k, \mathbf{e}_i^k > \\ &= \hat{y}_{ui}^{k-2} + < \mathbf{e}_u^{k-1}, \mathbf{e}_i^{k-1} > + < \mathbf{e}_u^k, \mathbf{e}_i^k > \\ &= \hat{y}_{ui}^{0} + < \mathbf{e}_u^1, \mathbf{e}_i^1 > + \ldots + < \mathbf{e}_u^k, \mathbf{e}_i^k > \\ &=< \mathbf{e}_u^0 \left\|\mathbf{e}_u^1\right\| \ldots \left\|\mathbf{e}_u^k, \quad \mathbf{e}_i^0\right\| \mathbf{e}_i^1 \| \ldots \| \mathbf{e}_i^k > .\end{aligned} \tag{11}$$

The above equation is equivalent to concatenate embedding of each layer to form the final embedding of each node. This is quite reasonable as each node's sub-graph varies, and recording each layer's representation to form the final embedding of each node is more informative.

3.4 Model Learning

The trainable parameters of our model are only the embeddings of the first-order layer, such as $\mathbf{W} = \mathbf{E}^{(0)}$. In other words, the model complexity is same as the standard matrix factorization (MF). We adopt the ranking based loss function in Bayesian Personalized Ranking (BPR) [28], which a pairwise loss that encourages the prediction of an observed entry to be higher than its unobserved counterparts:

$$\min_{\mathbf{W}} L(\mathbf{R}, \hat{\mathbf{R}}) = \sum_{a=1}^{M} \sum_{(i,j) \in D_a} - \ln(s(\hat{r}_{ai} - \hat{r}_{aj})) + \lambda \left\|\mathbf{W}\right\|^2 \tag{12}$$

where λ controls the L_2 regularization strength. We employ the Adam SGD [18] optimizer and use it in a mini-batch manner. We are aware of other advanced negative sampling strategies which might improve the SGCF training, such as the hard negative sampling [27] and adversarial sampling [5]. We leave this extension in the future since it is not the focus of this work. Note that we do not introduce dropout mechanisms, which are commonly used in GCNs and NGCF. The reason is that we do not have feature transformation weight matrices in SGCF, thus enforcing L_2 regularization on the embedding layer is sufficient to prevent over fitting. This showcases SGCF's advantages of being simple — it is easier to train and tune than NGCF which additionally requires to tune two dropout ratios, such as node dropout and message dropout, and normalize the embedding of each layer to unit length. Moreover, there is one crucial issue remaining in the network embedding learning on the condensed graph $\mathcal{G}_c$ which is the configuration of hyperparameters in the random walk based algorithm, i.e., the number of random walks and the length of a random walk. To cope with this issue, we utilize a pre-processing phase which trains a regression model that takes into account both the hyperparameters and the statistics of the condensed graphs. As such, given an condensed graph $\mathcal{G}_c$, we are able to infer from the model the suitable hyperparameters for $\mathcal{G}_c$ with a slight cost, as explained shortly.

3.5 Model Analysis

Detailed Analysis of Model Based on the prediction function in Eq.(11), we observe that SGCF is not a deep neural network but a wide linear model. The linearization has several advantages: First, as SGCF is built on the recent progress of SGC, it is theoretically connected as a low pass filter of graph on the spectral domain [34]. Second, with the linear embedding propagation and partition graph learning, SGCF is much easier to train compared to nonlinear GCN based models. Last but not least, we obtain the initialization of the embedding for each node in the graph by computing the network embedding on the condensed graph, which is much smaller than the input graph, and then propagating the embedding among the nodes in the input graph. Instead, we could resort to stochastic gradient descent for model learning. Therefore, SGCF is much more time efficient compared to classical GCN based models.

Connections with Existing Work We compare the key characteristics of our proposed model with three closely related GCN based recommendation models: PinSage [24], NGCF, and LightGCN. NGCF and LightGCN are both the first few attempts that also use a residual prediction function by taking each user (item)'s embedding as a concatenation of all layers' embeddings. However, the authors simply use this "trick" without any detailed explanation. We empirically show the reason why taking the output of the last layer embedding fails for CF, and show using residual prediction is equivalent to concatenate all the layer's embeddings as the final embedding of each node in the user-item bipartite graph. For PinSage, it has a lower time complexity compared to its deep learning based counterparts (e.g., NGCF) as this model designed a sampling technique in feature aggregation process.

4 EXPERIMENTS

We first compare SGCF with various state-of-the-art CF methods to demonstrate its effectiveness and high efficiency. We also perform detailed parameter studies to justify the rationality and effectiveness of the design choice of SGCF.

4.1 Experimental Setup

4.1.1 Datasets. We utilize four publicly available datasets, including Yelp2018, Amazon-Books, Gowalla [1], and MovieLens to conduct our experiments, as many recent GCN-based CF models [3, 10, 11, 13, 29, 32, 33, 35] are evaluated on these four datasets. We closely follow these GCN-based CF studies and use the same data split as them. Table 1 shows the statistics of the used datasets.

[1]http://www.gowalla.com/

- **Yelp2018**: This dataset is adopted from the 2018 edition of the Yelp challenge. Where in, 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-Books**: Amazon-books is a widely used dataset for product recommendation [12]. We select Amazon-Books from the collection. Similarly, we use the 10-core setting to ensure that each user and item have at least ten interactions.
- **Gowalla**: is a location-based social networking website where users share their locations by checking-in. The friendship network is undirected and was collected using their public API, and consists of 196,591 nodes and 950,327 edges.
- **MovieLens**: The MovieLens dataset is obtained from the MovieLens 10M Data [2]. We assume a user has an interaction with a movie if the user gives it a rating of 4 or 5.

4.1.2 Baselines. In total, we compare SGCF with three types of the stat-of-the-art models, covering MF-based methods, metric learning-based approaches and GCN-based models.

- **MF-based methods**: MF-BPR [22] a pairwise method that exploits different types of feedback with an extended sampling method. ENMF [23]) an Efficient Adaptive Transfer Neural Network (EATNN) for social-aware recommendation. Metric learning-based method - CML [14].
- **Networking embedding methods**: DeepWalk [25] learns embeddings via the prediction of the local neighborhood of nodes, sampled from random walks on the graph. LINE [31] is suitable for arbitrary types of information networks: undirected, directed, and/or weighted. Node2Vec [6] is a state of art graph representation learning method. It utilizes random walk to capture the proximity in the network and maps all the nodes into a low-dimensional representation space which preserves the proximity.
- **GCN-based methods**: NGCF achieves the target by leveraging high-order connectivities in the user-item integration graph. NIA-GCN [30] can explicitly model the relational information between neighbor nodes and exploit the heterogeneous nature of the user-item bipartite graph. LR-GCCF [3] is a general GCN based CF model for recommendation. LightGCN learns user and item embeddings by linearly propagating them on the user-item interaction graph, and uses the weighted sum of the embeddings learned at all layers as the final embedding and DGCF [36] considers user-item relationships at the finer granularity of user intents and generates disentangled user and item representations to get better recommendation performance.

4.1.3 Evaluation Metrics. Given a user, a top-K item list recommendation algorithm provides a list of ranked item lists according to the predicted preference of them. To assess the ranked lists with respect to the ground-truth lists set of what users actually interacted with, we adopt three evaluation metrics: Normalized Discounted Cumulative Gain (NDCG) [16] at 20 (NDCG@20), Hit Ratio at 20 (HR@20) and recall at 20 (Recall@20).

[2]http://files.grouplens.org/datasets/movielens/

Table 1: Statistics of the datasets.

Dataset	#Users	#Items	#Interactions	Density
Amazon-Books	52,643	91,599	2,984,108	0.062 %
MovieLens-10M	71,567	10,681	10,000,054	0.371 %
Gowalla	29,858	40,981	1,027,370	0.084 %
Yelp2018	31,668	38,048	1,561,406	0.130 %

4.1.4 Parameter Settings. We implement our SGCF model in Tensorflow[3]. There are two important parameters in our model: 1) the dimension D of the user and item embedding matrix $\mathbf{E}$, and 2) the regularization parameter λ in the objective function (Eq.12). The embedding size is fixed to 64 for all models. In our proposed SGCF model, we try the regularization parameter λ in the range $[0.0001, 0.001, 0.01, 0.1]$ and find $\lambda = 0.01$ reaches the best performance. We adopt Gaussian distribution with 0 mean 10^{-4} standard deviation to initialize embeddings. There are several parameters in the baselines, for fail comparison, all the parameters in the baselines are also tuned to achieve the best performance.

4.2 Quantitative Performance Comparison

Our experimental results are reported in Table 2. We have several observations: 1) SGCF consistently outperforms all baseline approaches across all four datasets. In particular, SGCF hugely improves over the strongest GCN-based baseline on Amazon-Books by 10.16% and 10.15% by using Recall@20 and NDCG@20 respectively. The results of significance testing indicates that our improvements over the current strongest GCN-based baseline are statistically significant. In particular, SGCF show the effectiveness of modeling the information passing of a graph. NGCF is the baseline that captures higher-order user-item bipartite graph structure. It performs better than most baselines. Our proposed SGCF model consistently outperforms NGCF, thus showing the effectiveness of modeling the user preference by the residual preference prediction and the linear embedding propagation. Compared with other baselines, SGCF can leverage powerful graph convolution to exploit useful and deeper collaborative information in graphs. These advantages jointly lead to the superiority of SGCF than compared state-of-the-art models. 2) In total, network embedding models perform worse than GCN-based models, especially on Gowalla. The reason might be that the powerful graph convolution is more effective than traditional random walk in many network embedding methods, to capture collaborative information for recommendation. 3) Since SGCF is a special fixed filter on the graph spectral domain, its architecture is orthogonal to some stat-of-the-art models (e.g., SGC). Therefore, similar to low-pass-type filters, SGCF can be deemed as an effective and efficient CF framework which is possible to be incorporated with other methods. such as enabling disentangled representation for users and items as DGCF, to achieve better performance.

4.3 Efficiency Comparison

As highlighted in Section 3.5, SGCF is endowed with high training efficiency for CF due to its concise and unified designs. In this section, we further empirically demonstrate the superiority of SGCF on

[3]https://www.tensorflow.org/

Table 2: Overall performance comparison. Improv. denotes the relative improvements over the best GNN-based baselines.

Model	Amazon-Books		Yelp2018		Gowalla		MovieLens-10M	
	Recall@20	NDCG@20	Recall@20	NDCG@20	Recall@20	NDCG@20	Recall@20	NDCG@20
ENMF	0.0334	0.0279	0.0614	0.0525	0.1532	0.1351	0.2345	0.2098
CML	0.0413	0.0313	0.0621	0.0536	0.1639	0.1298	0.1725	0.1536
MF-BPR	0.0324	0.0259	0.0539	0.0432	0.1623	0.1346	0.2134	0.2135
DeepWalk	0.0347	0.0266	0.0478	0.0382	0.1042	0.0741	0.1351	0.1047
Node2Vec	0.0412	0.0307	0.0448	0.0361	0.1020	0.0711	0.1476	0.1190
LINE	0.0412	0.0321	0.0547	0.0445	0.1336	0.1057	0.2338	0.2232
NGCF	0.0345	0.0261	0.0580	0.0478	0.1571	0.1337	0.2515	0.2513
LR-GCCF	0.0336	0.0264	0.0560	0.0345	0.1521	0.1286	0.2230	0.2131
LightGCN	0.0412	0.0314	0.0651	0.0529	0.1824	0.1548	0.2573	0.2423
NIA-GCN	0.0371	0.0289	0.0589	0.0492	0.1361	0.1116	0.2361	0.2243
DGCF	0.0423	0.0325	0.0654	0.0534	0.1843	0.1563	0.2640	0.2504
SGCF	0.0466	0.0358	0.0683	0.0561	0.1862	0.1580	0.2787	0.2642
Improv.	10.16 %	10.15 %	4.43 %	4.66 %	1.03 %	1.08 %	5.57 %	5.51 %

Table 3: Efficiency comparison with full training time

Model	Epoch Count	Time per Epoch	Totally Time
MF-BPR	**25**	**33s**	**13.75 m**
LR-GCCF	170	70s	3h 30m
ENMF	85	135s	3h 11m
LightGCN	55s	850	12h 58m
SGCF	64	**36s**	**38.4 m**

training efficiency compared with other CF models, especially GCN-based models. To be specific, we select MF-BPR, ENMF, LightGCN, and LR-GCCF as the competitors, which are relatively efficient models in their respective categories. To be more convincing, we compare their training efficiency from two aspects: 1) The total training time and epochs for achieving their best performance. 2) Training them with the same epochs to see what performance they can achieve. Note that Table 3 shows that the training speed (i.e., Time per Epoch) of SGCF is close to MF-BPR, which empirically justifies our analysis that the time complexities of SGCF and MF are on the same level. SGCF needs 64 epochs to converge which is much less than LR-GCCF and LightGCN, leading to only 38.4 minutes for total training. Finally, SGCF has around 20x, 5x, 5x speedup compared with LightGCN, LR-GCCF, and ENMF respectively, demonstrating the big efficiency superiority of SGCF.

Moreover, Table 4 shows that when SGCF converges (i.e., train the fixed 64 epochs), the performances of all the other compared models are much worse than SGCF. That is to say, SGCF can achieve much better performance with less time, which further demonstrates the higher efficiency of SGCF that the other GCN-based CF models.

4.4 SGCF Model Component Analysis

To explore the effect of different components in SGCF model, we design a simplified version that removes the graph partition module in our framework. We call the simplified version model as LGCF. For LGCF and SGCF, with each predefined depth k, we calculate the cosine similarity of each pair of nodes (i.e., users and items)

Table 4: Efficiency comparison with same epochs. All models are trained with the fixed 64 epochs except MF-BPR. Since MF-BPR needs less than 64 epochs to converge, we report its actual training time.

Model	Training Time	Recall@20	NDCG@20
MF-BPR	**16m**	0.0342	0.0264
ENMF	2h45m	0.0357	0.0281
LR-GCCF	1h25m	0.0314	0.0191
LightGCN	1h41m	0.0345	0.0264
SGCF	**43m**	**0.0682**	**0.0561**

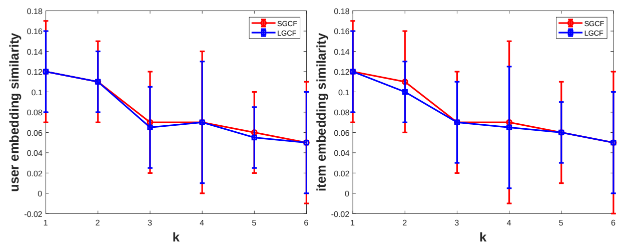

Figure 3: (left): Error-bar of user embedding similarity. (right): Error-bar of item embedding similarity. Comparisons with and without graph partition process structure under different layers depth k on Amazon-Books dataset.

between their k-layer output embedding, i.e., e^k for each node of the graph. The statistics of the mean and variance of user-user (item-item) embedding similarities are shown in Figure 3. It obviously shows our proposed model has larger variance of the user-user cosine similarity compared to its counterparts LGCF that does not perform condensed graph learning. This empirically validates that the condensed graph learning could partially alleviate the over smoothing issue, and achieves better performance. Please note that, the overall trend on the other three dataset is similar, and we do not illustrate it due to page limit.

Table 5: Performance of HR@20 and NDCG@20 with different depth k

Model	Amazon-Books		Gowalla	
	HR@20	NDCG@20	HR@20	NDCG@20
k=0	0.0284	0.0219	0.1379	0.1126
k=1	0.0317	0.0242	0.1506	0.1245
k=2	0.0327	0.0248	0.1504	0.1246
k=3	0.0337	0.0255	**0.1518**	**0.1561**
k=4	**0.0341**	**0.0324**	0.1496	0.1241
k=5	0.0340	0.0356	0.1504	0.1249

4.5 SGCF Model Parameter Study

4.5.1 Parameter Analysis. We would analyze the influence of the recursive label propagation depth k, and a detailed analysis of the learned embeddings of the preference prediction with condensed input graph in SGCF. Table 5 shows the results on SGCF with different k values. Specially, the layer-wise propagation part disappears when k=0, i.e., our proposed model degenerates to BPR. As can be observed from Table 5, when k increase from 0 to 1, the performance increase quickly on both datasets. For Amazon-Books, the best performance reaches with four propagation depth. Meanwhile, our model reaches the best performance when k=3 on Gowalla.

4.5.2 Scalability Analysis. As GCN-based networks are complex and contain such a large number of nodes in the real world application scenario, it is necessary for a model being feasible to be applied in the large-scale datasets. We investigate the scalability of SGCF model optimized by gradient descent, which deploys multiple threads for parallel model optimization. Our experiments are conducted in a computer server with 12 cores and 128GiB memory. We run experiments with different threads from 1 to 20. We depict in Figure 4 the speedup ratio vs. the number of threads. The speedup ratio is very close to linear, which indicates that the optimization algorithm of the SGCF is reasonably scalable.

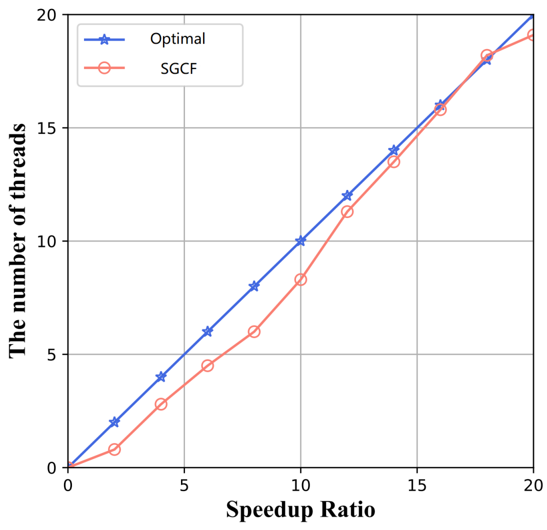

Figure 4: Scalability of SGCF

5 RELATED WORK

In this section, we briefly review some representative GCN-based methods and their efforts for model simplification toward recommendation tasks. With the development and success of GCN in various machine learning areas, there appears a lot of users and items could be naturally formed to a user-item bipartite graph and adapted GCNs for recommendation [9, 13, 32]. Earlier works on GCN based models relied on the spectral theories of graphs, and are computationally costly when applying in real-world recommendation. Some of recent works on GCN based recommendation models focused on the spatial domain [19]. PinSage was designed for similar item recommendation under the content based model, with the item feature x_v and the item-tiem correlation graph as the inputs. GC-MC [1] and NGCF are specifically designed under the CF setting. Although NGCF achieves good performance compared with previous non-GNN based methods, its heavy designs limit its efficiency and full exertion of GCN. To model the diversity of user intents on items, Wang et al. [36] devise Disentangled Graph Collaborative Filtering (DGCF) [33], which considers user-item relationships at the finer granularity of user intents and generates disentangled user and item representations to get better recommendation performance.

Although GCN-based recommendation models have achieved impressive performance, their efficiencies are still unsatisfactory when facing large-scale recommendation scenarios. How to improve the efficiency of GCNs and reserve their high performance for recommendation becomes a urgency research problem. Recently, Dai et al. [4] and Gu et al. [7] extend fixed-point theory on GNN for better representation learning. Liu et al. [21] propose UCMF that simplifies GCN for the node classification task. Wu et al. [34] find the non-necessity of nonlinear activation and feature transformation in GCN, proposing a simplified GCN (SGCN) model by removing these two parts. Inspired by SGC, He et al. [13] devise LightGCN for recommendation by removing nonlinear activation and feature transformation too. However, its efficiency is still limited by the time-consuming message passing. Qiu et al. [26] demonstrate that many network embedding algorithms with negative sampling can be unified into the MF framework which may be efficient, however,their performances still have a gap between that of GCNs. We are inspired by these instructive studies, and propose SGCF for both efficient and effective recommendation.

6 CONCLUSION

In this paper, we revisited the current GCN-based recommendation models and proposed an SGCF model for CF-based recommendation. SGCF consists of two main parts: First, with the recent progress of simple GCNs, we empirically removed the non-linear transformations in GCNs, and replaced it with linear embedding propagation. Second, to reduce the over smoothing effect introduced by higher layers of graph convolutions, we designed a condensed graph learning process for the input network. Extensive experimental results clearly showed the effectiveness and efficiency of our proposed model. In the future, we will explore better integration of different layers' representations with well-defined deep neural architectures to further enhance CF-based recommendation.

REFERENCES

[1] Rianne van den Berg, Thomas N Kipf, and Max Welling. 2017. Graph convolutional matrix completion. *arXiv preprint arXiv:1706.02263* (2017).

[2] Aydın Buluç, Henning Meyerhenke, Ilya Safro, Peter Sanders, and Christian Schulz. 2016. Recent advances in graph partitioning. *Algorithm engineering* (2016), 117–158.

[3] Lei Chen, Le Wu, Richang Hong, Kun Zhang, and Meng Wang. 2020. Revisiting graph based collaborative filtering: A linear residual graph convolutional network approach. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 34. 27–34.

[4] Hanjun Dai, Zornitsa Kozareva, Bo Dai, Alex Smola, and Le Song. 2018. Learning steady-states of iterative algorithms over graphs. In *International conference on machine learning*. PMLR, 1106–1114.

[5] Jingtao Ding, Yuhan Quan, Xiangnan He, Yong Li, and Depeng Jin. 2019. Reinforced Negative Sampling for Recommendation with Exposure Data.. In *IJCAI*. 2230–2236.

[6] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. 855–864.

[7] Fangda Gu, Heng Chang, Wenwu Zhu, Somayeh Sojoudi, and Laurent El Ghaoui. 2020. Implicit graph neural networks. *Advances in Neural Information Processing Systems* 33 (2020), 11984–11995.

[8] Bharath Hariharan, Pablo Arbeláez, Ross Girshick, and Jitendra Malik. 2015. Hypercolumns for object segmentation and fine-grained localization. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 447–456.

[9] Li He, Hongxu Chen, Dingxian Wang, Shoaib Jameel, Philip Yu, and Guandong Xu. 2021. Click-Through Rate Prediction with Multi-Modal Hypergraphs. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. 690–699.

[10] Li He, Xianzhi Wang, Hongxu Chen, and Guandong Xu. 2022. Online Spam Review Detection: A Survey of Literature. *Human-Centric Intelligent Systems* (2022), 1–17.

[11] Li He, Guandong Xu, Shoaib Jameel, Xianzhi Wang, and Hongxu Chen. 2022. Graph-Aware Deep Fusion Networks for Online Spam Review Detection. *IEEE Transactions on Computational Social Systems* (2022).

[12] Ruining He and Julian McAuley. 2016. Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. In *proceedings of the 25th international conference on world wide web*. 507–517.

[13] Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, Yongdong Zhang, and Meng Wang. 2020. Lightgcn: Simplifying and powering graph convolution network for recommendation. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*. 639–648.

[14] Cheng-Kang Hsieh, Longqi Yang, Yin Cui, Tsung-Yi Lin, Serge Belongie, and Deborah Estrin. 2017. Collaborative metric learning. In *Proceedings of the 26th international conference on world wide web*. 193–201.

[15] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q Weinberger. 2016. Deep networks with stochastic depth. In *European conference on computer vision*. Springer, 646–661.

[16] Kalervo Järvelin and Jaana Kekäläinen. 2002. Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems (TOIS)* 20, 4 (2002), 422–446.

[17] George Karypis and Vipin Kumar. 1998. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing* 20, 1 (1998), 359–392.

[18] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[19] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv* (2016).

[20] Wenqing Lin, Feng He, Faqiang Zhang, Xu Cheng, and Hongyun Cai. 2020. Initialization for network embedding: A graph partition approach. In *Proceedings of the 13th International Conference on Web Search and Data Mining*. 367–374.

[21] Qiang Liu, Haoli Zhang, and Zhaocheng Liu. 2020. Simplification of Graph Convolutional Networks: A Matrix Factorization-based Perspective. *arXiv preprint arXiv:2007.09036* (2020).

[22] Babak Loni, Roberto Pagano, Martha Larson, and Alan Hanjalic. 2016. Bayesian personalized ranking with multi-channel user feedback. In *Proceedings of the 10th ACM Conference on Recommender Systems*. 361–364.

[23] Xiaoke Ma and Di Dong. 2017. Evolutionary nonnegative matrix factorization algorithms for community detection in dynamic networks. *IEEE transactions on knowledge and data engineering* 29, 5 (2017), 1045–1058.

[24] Aditya Pal, Chantat Eksombatchai, Yitong Zhou, Bo Zhao, Charles Rosenberg, and Jure Leskovec. 2020. Pinnersage: Multi-modal user embedding framework for recommendations at pinterest. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2311–2320.

[25] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. 701–710.

[26] Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Kuansan Wang, and Jie Tang. 2018. Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec. In *Proceedings of the eleventh ACM international conference on web search and data mining*. 459–467.

[27] Steffen Rendle and Christoph Freudenthaler. 2014. Improving pairwise learning for item recommendation from implicit feedback. In *Proceedings of the 7th ACM international conference on Web search and data mining*. 273–282.

[28] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2012. BPR: Bayesian personalized ranking from implicit feedback. *arXiv preprint arXiv:1205.2618* (2012).

[29] Jinbo Song, Chao Chang, Fei Sun, Xinbo Song, and Peng Jiang. 2020. NGAT4Rec: Neighbor-Aware Graph Attention Network For Recommendation. *arXiv preprint arXiv:2010.12256* (2020).

[30] Jianing Sun, Yingxue Zhang, Wei Guo, Huifeng Guo, Ruiming Tang, Xiuqiang He, Chen Ma, and Mark Coates. 2020. Neighbor interaction aware graph convolution networks for recommendation. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 1289–1298.

[31] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. Line: Large-scale information network embedding. In *Proceedings of the 24th international conference on world wide web*. 1067–1077.

[32] 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*. 165–174.

[33] Xiang Wang, Hongye Jin, An Zhang, Xiangnan He, Tong Xu, and Tat-Seng Chua. 2020. Disentangled graph collaborative filtering. In *Proceedings of the 43rd international ACM SIGIR conference on research and development in information retrieval*. 1001–1010.

[34] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. 2019. Simplifying graph convolutional networks. In *International conference on machine learning*. PMLR, 6861–6871.

[35] Jiancan Wu, Xiang Wang, Fuli Feng, Xiangnan He, Liang Chen, Jianxun Lian, and Xing Xie. 2021. Self-supervised graph learning for recommendation. In *Proceedings of the 44th international ACM SIGIR conference on research and development in information retrieval*. 726–735.

[36] Zekun Yin, Xiaoming Xu, Kaichao Fan, Ruilin Li, Weizhong Li, Weiguo Liu, and Beifang Niu. 2019. DGCF: A Distributed Greedy Clustering Framework for Large-scale Genomic Sequences. In *2019 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*. IEEE, 2272–2279.