IEICE TRANS. INF. & SYST., VOL.E108–D, NO.8 AUGUST 2025

PAPER *Special Section on Data Engineering and Information Management*

AdaCF: Adaptively Updating Adjacency Matrix in Collaborative Filtering

Chi ZHANG[†a)], **Luwei ZHANG**[†b)], ***Nonmembers***, ***and*** **Toshihiko YAMASAKI**[†c)], ***Senior Member***

SUMMARY Collaborative Filtering (CF) is a crucial task in recommender systems which tries to exploit past user-item interactions to recommend new items to users. Recently, Graph Convolutional Networks (GCN) based models have achieved great success in CF by exploring various methods to embed user and item nodes in the interaction graph. However, as node embeddings gradually learn how to interact with other nodes, previous models use the same adjacency matrix throughout the training process, which introduces errors during training under the Bayesian Personalized Ranking (BPR) loss function, and ignores the useful information already acquired by the embeddings. In this paper, we theoretically analyze the gradients generated from BPR loss function and prove it cannot train CF models well because of the sparsity of existing CF datasets. Based on such analysis, we propose a method called AdaCF (Adaptive Collaborative Filtering). AdaCF adaptively updates the adjacency matrix by dynamically incorporating the user-item interaction information into the original adjacency matrix during the training process. AdaCF can be applied to any GCN-based CF model, and it is simple to compute and does not require very much extra training time. Experimental results show that AdaCF improves the performance of the base models across all three widely used CF datasets.

key words: *collaborative filtering, graph convolutional networks, adjacency matrix*

1. Introduction

With the advancement of online services such as social networks and online shopping, recommender systems have become a hot topic in recent years. The central challenge of recommender systems is how to suggest new items to users. Collaborative filtering, which generates new recommendations based solely on past user-item interactions, remains a crucial task in this field.

As user-item interactions can be represented by graphs, graph convolutional networks (GCNs) have been widely used in collaborative filtering (CF) [1], [2], reaching high performance on multiple datasets [3]–[5]. In particular, LightGCN [2] has become a key baseline model in CF because of its simple linear structure and good performance. Many methods have been proposed to improve the performance of LightGCN, such as simplifying it into its infinite convergence point [6], designing new loss functions [7], and introducing graph contrastive learning (GCL) methods [8].

However, as these models try to improve LightGCN, they tend to ignore the network structure of LightGCN itself.

Manuscript received May 21, 2024.
Manuscript revised October 23, 2024.
Manuscript publicized December 10, 2024.
[†]The University of Tokyo, Tokyo, 113–8656 Japan.
a) E-mail: zhangchi@cvm.t.u-tokyo.ac.jp
b) E-mail: zhangluwei@cvm.t.u-tokyo.ac.jp
c) E-mail: yamasaki@cvm.t.u-tokyo.ac.jp
DOI: 10.1587/transinf.2024DAP0006

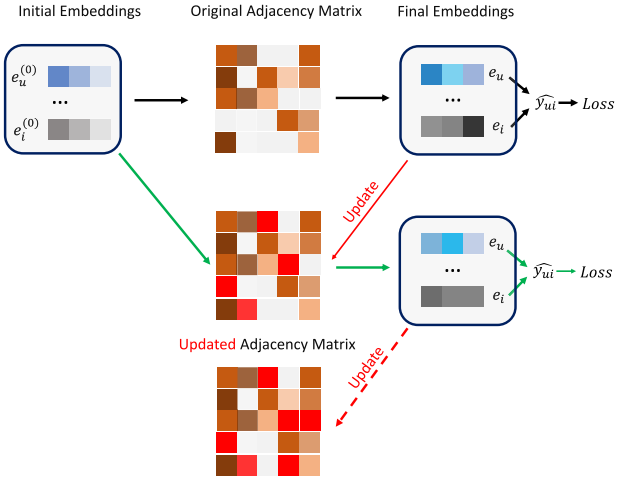


Fig. 1 The structure of our method AdaCF. During the training of the base GCN model, AdaCF updates the adjacency matrix using the final embeddings calculated by GCN at certain epochs and then uses the updated adjacency matrix to train the subsequent epochs.

Table 1 Statistics of the experimented datasets.

Datasets	#Users	#Items	#Interactions	Density
ML-1M [3]	6,040	3,952	1,000,209	4.2×10^{-2}
Yelp2018 [4]	31,668	38,048	1,561,406	1.3×10^{-3}
Amazon-Book [5]	52,643	91,599	2,984,108	6.2×10^{-4}

Given the initial user and item embeddings, represented by $\mathbf{E}^{(0)}$, as the only trainable parameters, the message passing process of LightGCN is defined as follows:

$$\mathbf{E}^{*} = \frac{1}{L+1}\left(\mathbf{E}^{(0)} + \tilde{\mathbf{A}}\mathbf{E}^{(0)} + \cdots + \tilde{\mathbf{A}}^{L}\mathbf{E}^{(0)}\right), \tag{1}$$

where L is the number of layers and $\tilde{\mathbf{A}}$ is the normalized adjacency matrix representing the user-item interactions. We can see that $\tilde{\mathbf{A}}$ is crucial in LightGCN as it represents the whole information used to guide the transformation of embeddings. However, previous studies did not recognize this importance and simply kept it unchanged throughout the entire training process.

Based on such observation, we argue that relying on the original adjacency matrix throughout training is unreasonable. Since adjacency matrices are usually very sparse as shown in Table 1, and contain incomplete data limiting their ability to fully capture the relationships between users and items that CF models can exploit. In fact, items that users do not interact with will be labeled the same as items that users

Copyright © 2025 The Institute of Electronics, Information and Communication Engineers

are not interested in, which can introduce significant errors to CF models during training using BPR loss functions. We refer to this phenomenon as the fake zero problem. Conversely, the user and item embeddings learned by CF models usually contain more information than the original adjacency matrix, as calculating their similarity can generate new recommended items to the users. Therefore, we propose our AdaCF, which updates the entries in the adjacency matrix using the similarity score of user and item embeddings during training. We apply our method to four widely used GCN-based CF models and train on three major datasets used in CF. Results show that our methods can improve the results of these models, showing the power and adaptivity of our model AdaCF. The performance improvement achieved by this method may appear small, but it is a universally applicable approach that ensures overall enhancement. Therefore, incorporating this method is advantageous.

In this paper, we make the following contributions:

- We theoretically analyze the training process of existing CF models and prove that the widely-used BPR loss function will introduce errors due to the sparsity of existing CF datasets and the presence of fake zeros.
- We introduce AdaCF, a method of updating the adjacency matrix based on user and item embeddings during training. AdaCF is a general method and can be applied to any GCN-based CF model.
- Experimental results show that our method can improve the performance of four state-of-the-art GCN-based CF models on three major datasets.

2. Related Works

2.1 Collaborative Filtering

Collaborative filtering is the task of recommending new items to users based only on user-item interaction history. Before the deep learning era, UserCF [9] and ItemCF [5] were proposed, using the similarity between users or items to generate recommendations. By formalizing CF as a matrix completion problem, Matrix Factorization [10] decomposes the interaction matrices as the product of low dimensional user and item matrix, pioneering the calculation of embeddings for users and items.

In the deep learning era, the graph structure of user-item interaction data makes GCN [11] a natural fit for collaborative filtering [1], [12]. GCN-based CF models typically use GCN to embed users and items based on the adjacency matrix and then calculate their similarity score between these embeddings to give new recommendations.

Among the GCN-based CF models, LightGCN [2] gains the most focus because of its simple linear propagation strategy. By eliminating the second-order information aggregation and nonlinear activation function, and simply performing linear transformations on the initial embeddings of users and items during training and inference, LightGCN reduces training parameters to these initial embeddings and achieves much better results than previous GCNs based models.

To train the GCN-based CF models, the BPR loss function [13] is the most widely used, which sums the log difference between every positive and negative pair for every user. Following this pioneering work, various kinds of loss functions have been proposed. MCL [14] is based on the distance between every user and item rather than user-item pairs. SML [15] calculates the similarity difference between positive pairs and a set of negative pairs for every user by negative sampling. DirectAU [7] proposes two new loss functions focusing on the alignment and uniformity of the user and item embeddings. Still, as discussed by [16], the BPR loss function is highly representative and continues to be widely used in training state-of-the-art CF models.

2.2 LightGCN-Based Collaborative Filtering Models

The power and simplicity of LightGCN make it the backbone of GCN-based CF models thereafter. For example, GTN [17] introduces an optimization process and designs a simple object function to guide the embeddings before training. LT-OCF [18] redesigns LightGCN as a continuous diffusive process and solves heat equations to train the user and item embeddings. HOCF [19] changes the embedding space from Euclidean space to hyperbolic space because graph structures are more similar to the latter.

Since the datasets used by CF are usually very sparse, graph data augmentation and contrastive learning [20], [21] are powerful tools to help train the models. In CF, graph contrastive learning (GCL) is introduced by capturing the implicit information hidden in the user-item graph structure in an unsupervised way. SGL [8] is one of the first GCL-based models in CF. It defines three kinds of graph data augmentation, which are node dropout, edge dropout, and random walk. SimGCL [22] introduces another method of graph data augmentation: adding noise to the user and item embeddings. XSimGCL [23] further simplifies SimGCL by removing the extra networks used to generate embeddings for comparison, and directly comparing embeddings at different layers during the propagation processes. GCCE [24] proposes a new method of complementary embedding generation, achieved through normalization using the maximum and minimum values in the user and item embeddings. SimGCL and XSimGCL are now the state-of-the-art models in CF.

Although very useful in CF, all the above GCN-based models ignore the crucial role of the adjacency matrix in graph learning. Therefore, we propose our method AdaCF, which updates the adjacency matrix during training, to generate better user and item embeddings.

3. Theoretical Analysis

3.1 Preliminaries

Given a user-item interaction graph G where there are m

users $U = \{u_1, u_2, \ldots, u_m\}$ and n items $I = \{i_1, i_2, \ldots, i_n\}$, we represent this graph as an adjacency matrix $\mathbf{A} \in \{0,1\}^{(m+n)\times(m+n)}$, where $\mathbf{A}_{j,m+k} = \mathbf{A}_{m+k,j} = 1$ if u_j interacts with i_k and 0 otherwise.

The task of GCN-based CF models is to learn d-dimension embedding vectors $\mathbf{e}_u, \mathbf{e}_i$ for every user u and item i that represent them well for inference. Here we introduce the procedure of LightGCN, which is the backbone model of most GCN-based CF models. In the propagation process, starting from the initial embeddings $\mathbf{e}_u^{(0)}$ and $\mathbf{e}_i^{(0)}$, on the l-th layer, LightGCN updates these embeddings by linear transformations:

$$\mathbf{e}_u^{(l)} = \sum_{i\in\mathcal{N}_u} \frac{1}{\sqrt{|\mathcal{N}_u|}\sqrt{|\mathcal{N}_i|}} \mathbf{e}_i^{(l-1)}, \tag{2}$$

$$\mathbf{e}_i^{(l)} = \sum_{u\in\mathcal{N}_i} \frac{1}{\sqrt{|\mathcal{N}_i|}\sqrt{|\mathcal{N}_u|}} \mathbf{e}_u^{(l-1)}, \tag{3}$$

where $\mathcal{N}_u$ and $\mathcal{N}_i$ denote the neighborhood of user u and item i respectively, and $(\sqrt{|\mathcal{N}_i|}\sqrt{|\mathcal{N}_u|})^{-1}$ represents the entries in normalized adjacency matrix $\tilde{\mathbf{A}}$, also acting as the weight controlling the information passing from neighborhood nodes. Then, in the aggregation process, LightGCN will simply stack embeddings at each layer to generate the final embeddings:

$$\mathbf{e}_u = \frac{1}{L+1}\sum_{l=0}^{L} \mathbf{e}_u^{(l)}, \quad \mathbf{e}_i = \frac{1}{L+1}\sum_{l=0}^{L} \mathbf{e}_i^{(l)}, \tag{4}$$

where L is the number of layers in LightGCN.

In the training process, multiple types of loss functions can be used to train the embeddings. For example, LightGCN would first predicts the similarity score between any user u and item i based on $\mathbf{e}_u$ and $\mathbf{e}_i$:

$$\hat{y}_{ui} = \mathbf{e}_u^\mathsf{T} \mathbf{e}_i, \tag{5}$$

and then calculates the BPR loss function [13]:

$$L_{\text{BPR}} = -\sum_{u=1}^{m} \sum_{i\in\mathcal{N}_u} \sum_{j\notin\mathcal{N}_u} \log(\hat{y}_{ui} - \hat{y}_{uj}), \tag{6}$$

plus certain regularization terms to train the embeddings. GCL-based models also use graph augmentation methods to generate multiple views of user and item embeddings, and then compute contrastive loss functions based on these views, to help train these embeddings.

From Eq. (6), we can see that the sum is based on every pair of positive user-item pair (u,i) and negative user-item pair (u,j), which originate from the entries in the adjacency matrix $\mathbf{A}$. Thus we can see that $\mathbf{A}$ plays two important roles in LightGCN training guided by the BPR loss function: one in the propagation of LightGCN as shown in Eq. (1), and another in the calculation of BPR loss function.

3.2 Gradient Analysis

In this section, we give a theoretical gradient analysis for LightGCN training guided by the BPR loss function. We argue that the BPR loss function generates gradients that make users and potential items move further apart, a behaviour that is contrary to the purpose of the propagation process of GCN.

For LightGCN, the initial embeddings $\mathbf{e}_u^{(0)}$ and $\mathbf{e}_i^{(0)}$ are the only trainable parameters. For simplicity, we only discuss the gradients to the user embeddings here, as GCN-based CF models generally treat users and items equally, so user and item embeddings have a symmetric structure. By Eq. (3), (4) and (5), we can calculate the gradient as follows:

$$\frac{\partial L}{\partial \mathbf{e}_u^{(0)}} = -\sum_{v=1}^{m} \sum_{i\in\mathcal{N}_v} \sum_{j\notin\mathcal{N}_v} \frac{1}{\hat{y}_{vi} - \hat{y}_{vj}} \left(\frac{\partial \hat{y}_{vi}}{\partial \mathbf{e}_u^{(0)}} - \frac{\partial \hat{y}_{vj}}{\partial \mathbf{e}_u^{(0)}}\right). \tag{7}$$

Here we replace the u in Eq. (1) with v to distinguish between the user embedding $\mathbf{e}_u^{(0)}$ to be updated.

Theoretically, every user can be connected to every item, making it nearly impossible to calculate this gradient accurately. Since the adjacency matrix we use in CF is usually very sparse, and LightGCN is usually limited by 3 layers, we further make the following assumptions:

1. Every user and item has exactly δm and δn neighborhood nodes, where δ is a constant representing the density of the user-item interaction graph.
2. $L = 3$, which means that every node embedding is updated only by nodes in its 3-step neighborhoods.
3. There are no loops in the user-item interaction graph, which means that for every $\mathbf{e}_u^{(l)}$ or $\mathbf{e}_i^{(l)}$, other initial node embeddings $\mathbf{e}_u^{(0)}$ or $\mathbf{e}_i^{(0)}$ will appear at most once in their 3-step neighborhood.

Assumption (3) may seem too strong, but it is proposed based on assumption (1). Suppose there is a loop of users u_1 and u_2 are both related to items i_1 and i_2, then the probability of such loop existing is about $n(n-1)\delta^4$, which is usually very low for CF datasets shown in Table 1, so we can neglect them. Assumption (1) may also seem to be very strong, but the user nodes which have not too many neighborhood nodes are more important for CF, because users who are related to a limited number of items are more valuable than those who are related to a large number of items.

Under these assumptions, there are $\delta(1-\delta)mn^2$ terms in Eq. (6), and every user u will have δn neighborhood items, $\delta^2 mn$ 2-step neighborhood users, and $\delta^3 mn^2$ 3-step neighborhood items. Thus, to calculate the gradient in Eq. (7), we can split it into three parts based on the identity of the user v in the numerator $\hat{y}_{vi}$ and $\hat{y}_{vj}$ compared to the user u whose embedding is being updated in the denominator.

(1) v is the user u. In this case, we first calculate the gradients from $\hat{y}_{vi}$ and $\hat{y}_{vj}$ separately. For every positive item i, its δn neighborhood items and $\delta^3 mn^2$ 3-step neighborhood items will be involved, and we have the following:

$$\frac{\partial \hat{y}_{ui}}{\partial \mathbf{e}_u^{(0)}} = \frac{1}{(L+1)^2} \frac{\partial (\sum_{l=0}^{3} \mathbf{e}_u^{(l)})^\mathsf{T} (\sum_{l=0}^{3} \mathbf{e}_i^{(l)})}{\partial \mathbf{e}_u^{(0)}} \tag{8}$$

$$= \frac{1}{16}\left(\frac{\partial(\sum_{l=0}^{3}\mathbf{e}_u^{(l)})}{\partial \mathbf{e}_u^{(0)}}\sum_{l=0}^{3}\mathbf{e}_i^{(l)} + \frac{\partial(\sum_{l=0}^{3}\mathbf{e}_i^{(l)})}{\partial \mathbf{e}_u^{(0)}}\sum_{l=0}^{3}\mathbf{e}_u^{(l)}\right) \tag{9}$$

$$= \frac{1}{16}\left(\sum_{l=0}^{3}\mathbf{e}_i^{(l)} + \frac{1}{\delta\sqrt{nm}}\sum_{l=0}^{3}\mathbf{e}_u^{(l)}\right) \tag{10}$$

$$= \frac{1}{4}\left(\mathbf{e}_i + \frac{1}{\delta\sqrt{nm}}\mathbf{e}_u\right), \tag{11}$$

where it should be noted that under assumption (3) only $\mathbf{e}_i^{(1)}$ and $\mathbf{e}_u^{(0)}$ are related to $\mathbf{e}_u^{(0)}$ and we have

$$\mathbf{e}_i^{(1)} = \frac{1}{\sqrt{\delta n}\sqrt{\delta m}}\mathbf{e}_u^{(0)} + \sum \text{others}, \tag{12}$$

from Eq. (3).

For the negative item j, notice that it cannot lie in the neighborhood of u, so j is either a 3-step neighborhood item of u, or it is not connected to u in 3 steps. For the first case, we have the following:

$$\frac{\partial \hat{y}_{uj}}{\partial \mathbf{e}_u^{(0)}} = \frac{1}{16}\frac{\partial(\sum_{l=0}^{3}\mathbf{e}_u^{(l)})^{\mathsf{T}}(\sum_{l=0}^{3}\mathbf{e}_j^{(l)})}{\partial \mathbf{e}_u^{(0)}} \tag{13}$$

$$= \frac{1}{16}\left(\frac{\partial(\sum_{l=0}^{3}\mathbf{e}_u^{(l)})}{\partial \mathbf{e}_u^{(0)}}\sum_{l=0}^{3}\mathbf{e}_j^{(l)} + \frac{\partial(\sum_{l=0}^{3}\mathbf{e}_j^{(l)})}{\partial \mathbf{e}_u^{(0)}}\sum_{l=0}^{3}\mathbf{e}_u^{(l)}\right) \tag{14}$$

$$= \frac{1}{16}\left(\sum_{l=0}^{3}\mathbf{e}_j^{(l)} + \frac{1}{\delta^3\sqrt{n^3m^3}}\sum_{l=0}^{3}\mathbf{e}_u^{(l)}\right) \tag{15}$$

$$= \frac{1}{4}\left(\mathbf{e}_j + \frac{1}{\delta^3\sqrt{n^3m^3}}\mathbf{e}_u\right), \tag{16}$$

because if we suppose the 3-step connection between u and j is $u \leftrightarrow j' \leftrightarrow u' \leftrightarrow j$, then we have:

$$\mathbf{e}_i^{(3)} = \frac{1}{\sqrt{\delta n}\sqrt{\delta m}}\mathbf{e}_{u'}^{(2)} + \sum \text{others} \tag{17}$$

$$= \frac{1}{\delta n \delta m}\mathbf{e}_{j'}^{(1)} + \sum \text{others} \tag{18}$$

$$= \frac{1}{\sqrt{\delta^3 n^3}\sqrt{\delta^3 m^3}}\mathbf{e}_u^{(0)} + \sum \text{others}. \tag{19}$$

When j is not connected to u in three steps, we have:

$$\frac{\partial \hat{y}_{uj}}{\partial \mathbf{e}_u^{(0)}} = \frac{1}{4}\mathbf{e}_j, \tag{20}$$

Generally speaking, $\delta\sqrt{nm}$ is a large number for CF datasets, so approximately we have the following:

$$\frac{\partial \hat{y}_{ui}}{\partial \mathbf{e}_u^{(0)}} - \frac{\partial \hat{y}_{uj}}{\partial \mathbf{e}_u^{(0)}} \approx \frac{1}{4}\left(\mathbf{e}_i - \mathbf{e}_j + \frac{1}{\delta\sqrt{nm}}\mathbf{e}_u\right) \tag{21}$$

$$\approx \frac{1}{4}(\mathbf{e}_i - \mathbf{e}_j). \tag{22}$$

From here we can already see that the role of BPR loss function during training is to make user embeddings $\mathbf{e}_u^{(0)}$ closer to positive item embeddings $\mathbf{e}_i$, and far away from negative embeddings $\mathbf{e}_j$.

(2) When v is in the 2-step neighborhood of u. According to assumption (3), we can assume i' is the only item connecting both of them. Thus we have

$$\frac{\partial \hat{y}_{vi'}}{\partial \mathbf{e}_u^{(0)}} = \frac{1}{16}\frac{\partial(\sum_{l=0}^{3}\mathbf{e}_v^{(l)})^{\mathsf{T}}(\sum_{l=0}^{3}\mathbf{e}_{i'}^{(l)})}{\partial \mathbf{e}_u^{(0)}} \tag{23}$$

$$= \frac{1}{16}\left(\frac{\partial \mathbf{e}_v^{(2)}}{\partial \mathbf{e}_u^{(0)}}\sum_{l=0}^{3}\mathbf{e}_{i'}^{(l)} + \frac{\partial \mathbf{e}_{i'}^{(1)}}{\partial \mathbf{e}_u^{(0)}}\sum_{l=0}^{3}\mathbf{e}_v^{(l)}\right) \tag{24}$$

$$= \frac{1}{4}\left(\frac{1}{\delta^2 nm}\mathbf{e}_{i'} + \frac{1}{\delta\sqrt{nm}}\mathbf{e}_v\right), \tag{25}$$

and for other positive items i of u that is not positive for v, we have

$$\frac{\partial \hat{y}_{vi}}{\partial \mathbf{e}_u^{(0)}} = \frac{1}{16}\left(\frac{\partial \mathbf{e}_v^{(2)}}{\partial \mathbf{e}_u^{(0)}}\sum_{l=0}^{3}\mathbf{e}_i^{(l)} + \frac{\partial \mathbf{e}_i^{(3)}}{\partial \mathbf{e}_u^{(0)}}\sum_{l=0}^{3}\mathbf{e}_v^{(l)}\right) \tag{26}$$

$$= \frac{1}{4}\left(\frac{1}{\delta^2 nm}\mathbf{e}_i + \frac{1}{\delta^3\sqrt{n^3m^3}}\mathbf{e}_v\right). \tag{27}$$

Similarly, for negative item j of u not related to v, we have

$$\frac{\partial \hat{y}_{vj}}{\partial \mathbf{e}_u^{(0)}} = \begin{cases} \frac{1}{4}\left(\frac{1}{\delta^2 nm}\mathbf{e}_j + \frac{1}{\delta^3\sqrt{n^3m^3}}\mathbf{e}_v\right) & j \in \mathcal{N}_u^3, \\ \frac{1}{4}\frac{1}{\delta^2 nm}\mathbf{e}_j & \text{otherwise.} \end{cases} \tag{28}$$

where $j \in \mathcal{N}_u^3$ means j is in the 3-step neighborhood of u. Thus, we have approximately

$$\frac{\partial \hat{y}_{vi}}{\partial \mathbf{e}_u^{(0)}} - \frac{\partial \hat{y}_{vj}}{\partial \mathbf{e}_u^{(0)}} \approx \tag{29}$$

$$\begin{cases} \frac{1}{4}\left(\frac{1}{\delta^2 nm}(\mathbf{e}_i - \mathbf{e}_j) + \frac{1}{\delta\sqrt{nm}}\mathbf{e}_v\right), & i \in \mathcal{N}_u, \\ \frac{1}{4}\frac{1}{\delta^2 nm}(\mathbf{e}_i - \mathbf{e}_j), & \text{otherwise.} \end{cases} \tag{30}$$

It can be seen that for v, in every case, the gradient is much smaller than the gradients for u in Eq. (22). Since there are $\delta^2 nm$ 2-step neighborhoods v for u, when summed over all these v cases, the total gradient is about the same as the first case. It should also be noticed that the embeddings $\mathbf{e}_v$ of 2-step neighborhood v are also included in the gradients, which means that BPR loss functions will make users closer to their 2-step neighbor users.

(3) For the users v who cannot be related to u in less than 3 steps, we cannot find a connection between u, i, and v, so the gradient is 0:

$$\frac{\partial \hat{y}_{vi}}{\partial \mathbf{e}_u^{(0)}} = 0 \tag{31}$$

and similarly, for negative items j, only when j is a 3-step neighbor of u, would a connection between u, j and v exist, so we have:

$$\frac{\partial \hat{y}_{vj}}{\partial \mathbf{e}_u^{(0)}} = \begin{cases} \frac{1}{4\delta^3\sqrt{n^3m^3}}\mathbf{e}_v & j \in \mathcal{N}_u^3, \\ 0 & \text{otherwise.} \end{cases} \tag{32}$$

Thus, we have approximately

$$\frac{\partial \hat{y}_{vi}}{\partial \mathbf{e}_u^{(0)}} - \frac{\partial \hat{y}_{vj}}{\partial \mathbf{e}_u^{(0)}} \approx \begin{cases} -\frac{1}{4\delta^3\sqrt{n^3m^3}}\mathbf{e}_v & j \in \mathcal{N}_u^3, \\ 0, & \text{otherwise.} \end{cases} \tag{33}$$

which means that the gradients are quite small in this case, and the BPR loss function would make users further from users not related to them in less than 3 steps.

In all, we have the following gradients

$$\frac{\partial \hat{y}_{vi}}{\partial \mathbf{e}_u^{(0)}} - \frac{\partial \hat{y}_{vj}}{\partial \mathbf{e}_u^{(0)}} \approx \tag{34}$$

$$\begin{cases} \frac{1}{4}(\mathbf{e}_i - \mathbf{e}_j), & v = u, i \in \mathcal{N}_u, \\ \frac{1}{4}(\frac{1}{\delta^2 nm}(\mathbf{e}_i - \mathbf{e}_j) + \frac{1}{\delta\sqrt{nm}}\mathbf{e}_v), & v \in \mathcal{N}_u^2, i \in \mathcal{N}_u, \\ \frac{1}{4}\frac{1}{\delta^2 nm}(\mathbf{e}_i - \mathbf{e}_j), & v \in \mathcal{N}_u^2, i \notin \mathcal{N}_u, \\ -\frac{1}{4\delta^3\sqrt{n^3m^3}}\mathbf{e}_v & v \notin \mathcal{N}_u^2, j \in \mathcal{N}_u^3, \\ 0, & \text{otherwise.} \end{cases} \tag{35}$$

From Eq. (7) and (34), we can see that, generally speaking, the role of BPR loss function when training user embeddings $\mathbf{e}_u^{(0)}$ is to ensure that they incorporate positive item embeddings $\mathbf{e}_i$ and 2-step neighbor user embeddings $\mathbf{e}_v$ and subtract negative item embeddings $\mathbf{e}_j$. The problem with this training process is that LightGCN should recommend originally negative items j to the users, but since the entries y_{uj} in the adjacency matrix $\mathbf{A}$ would always be zero, propagation strategy of LightGCN would then update them in the way that they become further away from each other. Thus, $\mathbf{e}_u^{(0)}$ oscillates around the recommended items, and this should be the theoretical reason that LightGCN takes a long time to converge [17], [22].

It should be noticed that recent GCN-based CF models generally inherit the structure and training process of LightGCN using BPR loss function, so such behaviour does not only apply to LightGCN but should also exist in all LightGCN-based CF models.

In conclusion, we have theoretically proven that LightGCN, under the BPR loss function, updates the user embeddings far away from potential positive embeddings, causing user embeddings to oscillate around the recommended items. Thus, we propose our method, AdaCF, that tries to solve this problem by adding entries based on trained embeddings during training.

4. Method

4.1 Motivation

As we theoretically analyze the problem of LightGCN using the BPR loss function under three assumptions, we find that these problems become worse when real datasets are taken into account. This arises from the fake zero problem in CF datasets.

Since most recommender system datasets originate from online services [3]–[5], they often contain a lot of fake zeros. Although a user may be interested in an item, such potential interactions are usually missing in the dataset, because it is impossible for online services to collect every user's reactions towards every item. Thus, the large number of zeros in the adjacency matrix $\mathbf{A}$ does not mean that a user is not interested in most items. But the propagation process of GCN, as shown in Eq. (2) and Eq. (3), considers all these potentially interested items as not interested items. For a simple example shown in Fig. 2, it can be seen that user u_1 and u_2 are highly relevant to each other because they interact with a very similar set of items. In such a case, since item i_1 is related to user u_1, it is highly probable that item i_1 is also related to user u_2. Although there exist three-step connections between user u_1 and u_2, such as the path $i_1 \leftrightarrow u_1 \leftrightarrow i_2 \leftrightarrow u_2$, from Eq. (33) we can see that because i_1 is a negative item for u_2, during gradient update both the gradients of u_1 and i_1 will be subtracted from u_2, which will make i_1 far away from u_2. Thus, although during propagation u_2 can gain a small proportion of information of i_1 by 3 steps because i_1 is labeled negative in the dataset, this information will be subtracted during training. Considering the sparsity of existing CF datasets, this problem is even worse.

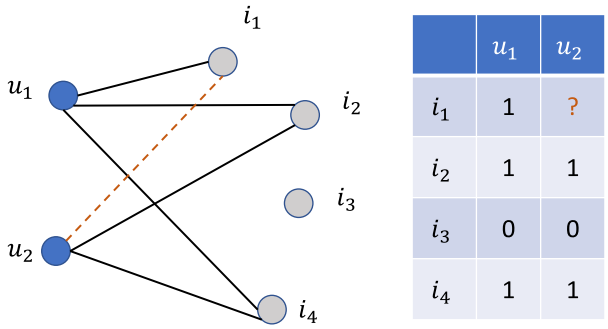


Fig. 2 One example showing the problem of fake negatives in a CF dataset. Here we give a user-item interaction graph of two users and four items, and it can be seen that although u_1 and u_2 are very similar and u_2 is highly likely to be interested in i_1, because u_2 does not have an interaction with i_1, this may result in embedding oscillation for u_2.

In other tasks of recommender system research, one may use extra information, such as user and item profiles, to alleviate this problem. However, collaborative filtering only uses user-item interactions, which means that the sparse $\mathbf{A}$ is all we can use. To solve this problem, we propose our method, AdaCF. The main idea of AdaCF is to use trained user and item embeddings to update the adjacency matrix so that the updated adjacency matrix can represent the user-item relationships better than the original adjacency matrix.

4.2 AdaCF: Updating Adjacency Matrix during Training

In this section, we introduce details of our method, AdaCF, which is shown in Algorithm 1. AdaCF introduces four extra hyperparameters, update start epoch t_0 denoting the training epoch when the first update occurs, update epoch τ denoting the frequency of updating during training, update threshold

Algorithm 1 AdaCF

Input: Adjacency Matrix $\mathbf{A}$, user embeddings $\mathbf{e}_u$, item embeddings $\mathbf{e}_i$, training epoch t, update start epoch t_0, update epoch τ, update threshold θ, update value $\alpha \geq 1$.
$\mathbf{A}_{\text{upd}} \leftarrow \mathbf{A}$
if $(t - t_0)\%\tau == 0$ **then**
  for all users u and items i **do**
    $\hat{y}_{ui} = \mathbf{e}_u^\top \mathbf{e}_i$
    if $\hat{y}_{ui} \geq \theta$ **then**
      if $\mathbf{A}(u, m+i)$=0 **then**
        $\mathbf{A}_{\text{upd}}(u, m+i), \mathbf{A}_{\text{upd}}(m+i, u) \leftarrow 1$
      else
        $\mathbf{A}_{\text{upd}}(u, m+i), \mathbf{A}_{\text{upd}}(m+i, u) \leftarrow \alpha$
      end if
    end if
  end for
end if
Output: Updated adjacency matrix $\mathbf{A}_{\text{upd}}$

θ controlling the update of every entry in the adjacency matrix, and the update value α distinguishing entries between original true values and updated true values.

In detail, after the update start epoch t_0, during every update epoch τ, AdaCF updates entries in the adjacency matrix $\mathbf{A}$. For every user u and item i with their trained embeddings $\mathbf{e}_u$ and $\mathbf{e}_i$, AdaCF calculates the predicted user-item interaction scores $\hat{y}_{ui}$ based on Eq. (5), and update each entry based on this score according to the strategy shown in Eq. (36).

$$\mathbf{A}_(u, m+i) \leftarrow \begin{cases} \alpha, & \text{if } \mathbf{A}(u, m+i) \geq 1 \text{ and } \hat{y}_{ui} \geq \theta, \\ 1, & \text{if } \mathbf{A}(u, m+i) = 0 \text{ and } \hat{y}_{ui} \geq \theta, \\ & \text{or } \mathbf{A}(u, m+i) \geq 1 \text{ and } \hat{y}_{ui} < \theta, \\ 0, & \text{if } \mathbf{A}(u, m+i) = 0 \text{ and } \hat{y}_{ui} < \theta. \end{cases} \tag{36}$$

And $\mathbf{A}(m+i, u)$ is updated in the same way. It can be seen that after each update, entries in $\mathbf{A}_{\text{upd}}$ will belong to one of the three values 0, 1, and α, where α is set to be larger than 1. This update strategy ensures that potential interactions can be constructed as many as possible, while not introducing wrong interactions. As AdaCF updates the entries based on user and item embeddings, these embeddings should already possess a certain level of accuracy before using for update. Therefore, the update start epoch is set to avoid introducing too much noise to the adjacency matrix. After each update, it takes some time for the embeddings to adjust and learn from the updated adjacency matrix, so the update epoch is set to ensure that each update can be used completely.

When a user and item have an interaction in the ground truth, and their embeddings are close to each other, we give them a higher interaction score α. This indicates that the trained embeddings correspond with the ground truth, making them more trustworthy. When the user and item have a high similarity score but the score is not recorded in the data, we update the corresponding value in the adjacency matrix to 1, treating it as important as the ground truth values where the corresponding embeddings are not sufficiently close.

Thus, we introduce a new adjacency matrix $\mathbf{A}_{\text{upd}}$ that includes many potential interests unseen in the original $\mathbf{A}$ and provides more information by varying in three values rather than two. Additionally, it's worth noting that the original nonzero entries will always remain nonzero, ensuring that no ground truth is omitted throughout the entire training process.

After every update, the message passing paradigm would also change as follows:

$$\mathbf{e}_u^{(l)} = \sum_{i \in \mathcal{N}_u} \frac{1}{\mathcal{Z}} \mathbf{e}_i^{(l-1)} + \sum_{\hat{y}_{ui} \geq \theta} \frac{1}{\mathcal{Z}} \mathbf{e}_i^{(l-1)} \tag{37}$$

$$\mathbf{e}_i^{(l)} = \sum_{u \in \mathcal{N}_i} \frac{1}{\mathcal{Z}} \mathbf{e}_u^{(l-1)} + \sum_{\hat{y}_{ui} \geq \theta} \frac{1}{\mathcal{Z}} \mathbf{e}_u^{(l-1)} \tag{38}$$

where $\mathcal{Z}$ is the new normalization constant. Here in each equation, the first term represents the message passing based on ground truth, and the second term represents the message passing from updated nodes.

It should be noticed that since $\tilde{\mathbf{A}}$ is a sparse matrix, under Eq. (1), the final embeddings $\mathbf{e}_u$ and $\mathbf{e}_i$ will be sparse even if the initial embeddings $\mathbf{e}_u^{(0)}$ and $\mathbf{e}_i^{(0)}$ are dense. This ensures the predicted similarity scores $\hat{y}_{ui}$ will generally be small. Thus our update strategy maintains the sparsity of the adjacency matrix, resulting in a relatively small additional computational cost.

5. Experiments

5.1 Experimental Settings

Base Models. We apply our method AdaCF to the following base models that are highly representative in CF:

- LightGCN [2]: A simple linear graph convolutional networks model considered as one important baseline in collaborative filtering.
- NCL [25]: One GCL-based model that sets structural and semantic neighbors of nodes as their positive contrastive pairs.
- SimGCL [22]: One GCL-based model that does contrastive learning by adding noise to user and item embeddings.
- XSimGCL [23]: Simplifies SimGCL by doing layer-wise contrastives, which is to compare embeddings generated by different layers of the network.

Dataset. We train our model on three widely used datasets in CF, MovieLens1M [3], Yelp2018 [4], and Amazon-book [5]. Their details are shown in Table 1. It can be seen that MovieLens1M is more dense while Yelp2018 and Amazon-book are quite large and sparse. We adapt the same train/test split as previous works [22], [23].

Metrics. We adopt the three widely used evaluation metrics in collaborative filtering, Precision@K, Recall@K and Normalized Discounted Cumulative Gain (NDCG@K). We set K as 20 by default.

Table 2 The performance comparison of top-20 recommendations by Precision, recall and NDCG values on the MovieLens1M(ML-1M), Yelp2018 and Amazon-book datasets.

Method	ML-1M			Yelp2018			Amazon-Book		
	Precision	Recall	NDCG	Precision	Recall	NDCG	Precision	Recall	NDCG
LightGCN [2]	0.198	0.269	0.302	0.0269	0.0597	0.0489	0.0164	0.0394	0.0309
LightGCN+AdaCF	**0.199**	**0.270**	0.302	**0.0270**	**0.0601**	**0.0493**	0.0164	**0.0395**	0.0308
NCL [25]	0.205	0.278	0.315	0.0302	0.0669	0.0551	0.0179	0.0432	0.0334
NCL+AdaCF	**0.206**	**0.280**	**0.319**	**0.0304**	**0.0676**	**0.0555**	**0.0180**	**0.0435**	**0.0337**
SimGCL [22]	0.175	0.253	0.278	0.0326	0.0722	0.0597	0.0200	0.0480	0.0377
SimGCL+AdaCF	**0.176**	0.253	**0.280**	**0.0330**	**0.0733**	**0.0602**	**0.0204**	**0.0492**	**0.0386**
XSimGCL [23]	0.199	0.277	0.310	0.0326	0.0726	0.0598	0.0201	0.0482	0.0379
XSimGCL+AdaCF	**0.200**	**0.279**	**0.311**	**0.0327**	**0.0727**	**0.0599**	**0.0203**	**0.0487**	**0.0381**

Implementation Details. To ensure a fair comparison, we implement our model based on the official Github repository provided by [26] using PyTorch, and for the baseline models, we directly use the codes from this repository. We report our experimental results rather than copying from the original papers. We use a single core A100-40G GPU for training. LightGCN is trained for 500 epochs, NCL for 120 epochs, SimGCL and XSimGCL for 50 epochs. The update start epoch was set to 100, 20, 10, and 10, and the update epoch was set to 20, 10, 10, and 10 for LightGCN, NCL, SimGCL, and XSimGCL, respectively. The update threshold was set to 0.9, and the update value α was set to 2. For other hyperparameters, we followed the settings from [22]. The update threshold is set as 0.9, and the update value α is set as 2. For other hyperparameters, we follow the settings from [22]. During training, we set the batch size as 2,048 and the embedding size as 64. We use Adam optimizer with learning rate as 10^{-3} and L_2 regularization rate as 10^{-4} to train the models, and the number of GCN layers as 3. For NCL, SimGCL, and XSimGCL, we follow the settings reported in their papers.

5.2 Performance Comparison with Base Models

5.2.1 Performance Comparison

In Table 2 we present the results of applying AdaCF to four base models across three datasets. For each (model, model+AdaCF) pair we denote the model+AdaCF generated values in bold if they are higher than those generated by the model alone. It can be seen from the performance table that:

1) Overall, AdaCF enhances the performance of three baseline models. Out of the 36 evaluation metrics across three datasets for four models, AdaCF improves 32 of them. This demonstrates AdaCF's capability to enhance the results of various GCN-based CF models.

2) On Yelp2018 and Amazon-Book dataset, the performance of SimGCL and XSimGCL surpasses that of LightGCN and NCL. This demonstrates the effectiveness of adding noise to embeddings as graph data augmentation methods. The improvement introduced by AdaCF is also much higher on SimGCL and XSimGCL than on LightGCN, which shows that AdaCF is more suitable for contrastive learning-based models than pure supervised learning-based models.

Table 3 The training time comparison on the MovieLens1M, Yelp2018 and Amazon-book datasets. All in seconds.

Model	ML-1M	Yelp2018	Amazon-Book
LightGCN	1,940	12,168	31,717
LightGCN+AdaCF	1,951	12,255	31,801
NCL	872	3,497	14,022
NCL+AdaCF	924	4,348	14,093
SimGCL	342	2,709	9,863
SimGCL+AdaCF	492	2,808	10,453
XSimGCL	258	1,312	3,853
XSimGCL+AdaCF	260	1,400	4,277

3) On the MovieLens1M dataset, the performance of LightGCN and NCL is quite promising compared with SimGCL and XSimGCL. This observation may stem from MovieLens1M's high density compared with other datasets as shown in Table 1. Consequently, we can infer that dataset sparsity has a considerable impact on the performance of state-of-the-art models such as SimGCL and XSimGCL. This suggests that the contrastive learning methods used by SimGCL and XSimGCL only work when the dataset is very sparse.

4) The performance of AdaCF on NCL is not as good as other models, which could be attributed to NCL's approach to constructing structural and semantic neighbors of nodes during data augmentation, as constructing a new neighbor may help the users and items to recognize their potential interests, so updating entries by AdaCF would not result in another high improvement in performance. Still, it can be seen that AdaCF does not decrease the performance of NCL.

5) Between SimGCL and XSimGCL, it is evident that the improvement introduced by AdaCF is lower on XSimGCL. This observation may indicate that XSimGCL's utilization of layerwise contrastives restricts its ability to fully leverage the additional information introduced by AdaCF, as compared to SimGCL.

6) Among the results on the three datasets, it is evident that the improvements introduced by AdaCF are higher on Amazon-Book. This suggests that sparser datasets like Amazon-Book contain more false negatives, allowing AdaCF to introduce more valuable information into the adjacency matrix.

Table 4 The influence of the update threshold θ on AdaCF. These results are conducted on the Yelp2018 dataset. Here the base case means the performance of the corresponding original method, which can be interpreted as the case of $\theta = 1.0$.

Threshold θ	LightGCN+AdaCF			NCL+AdaCF			SimGCL+AdaCF			XSimGCL+AdaCF		
	Precision	Recall	NDCG	Precision	Recall	NDCG	Precision	Recall	NDCG	Precision	Recall	NDCG
0.5	0.0253	0.0561	0.0455	0.0273	0.0595	0.0496	0.0291	0.0645	0.0536	0.0302	0.0715	0.0578
0.6	0.0266	0.0590	0.0484	0.0298	0.0658	0.0544	0.0312	0.0695	0.0569	0.0325	0.0722	0.0595
0.7	**0.0276**	**0.0613**	**0.0502**	**0.0304**	**0.0676**	0.0554	0.0322	0.0715	0.0585	0.0326	0.0722	0.0597
0.8	0.0270	0.0598	0.0491	0.0303	0.0672	0.0554	0.0326	0.0727	0.0595	0.0325	0.0726	0.0597
0.9	0.0270	0.0601	0.0493	**0.0304**	**0.0676**	**0.0555**	**0.0329**	**0.0730**	**0.0600**	**0.0327**	**0.0727**	**0.0599**
Base	0.0269	0.0597	0.0489	0.0302	0.0669	0.0551	0.0326	0.0722	0.0597	0.0326	0.0726	0.0598

Table 5 The influence of the update value α on AdaCF. These results are conducted on the Yelp2018 dataset. Here the base case means the performance of the corresponding original method.

Value α	LightGCN+AdaCF			NCL+AdaCF			SimGCL+AdaCF			XSimGCL+AdaCF		
	Precision	Recall	NDCG	Precision	Recall	NDCG	Precision	Recall	NDCG	Precision	Recall	NDCG
1.0	0.0268	0.0594	0.0489	0.0303	0.0671	0.0550	0.0327	0.0728	**0.0600**	0.0326	0.0726	0.0597
2.0	**0.0270**	**0.0601**	**0.0493**	**0.0304**	**0.0676**	**0.0555**	**0.0329**	**0.0730**	**0.0600**	**0.0327**	**0.0727**	**0.0599**
3.0	0.0269	0.0594	0.0488	0.0302	0.0669	0.0552	0.0327	0.0725	0.0599	0.0325	0.0722	0.0595
4.0	**0.0270**	0.0597	0.0490	0.0302	0.0673	0.0552	0.0326	0.07274	0.0596	**0.0327**	0.0726	0.0598
Base	0.0269	0.0597	0.0489	0.0302	0.0669	0.0551	0.0326	0.0722	0.0597	0.0326	0.0726	0.0598

5.2.2 Running Time Comparison

In Table 3, we present the total training time of each model on the MovieLens1M, Yelp2018, and Amazon-Book datasets. It can be seen that under all circumstances, AdaCF does not significantly increase the training time. Thus, we demonstrate that AdaCF is both effective in performance and efficient in computational cost.

Although Algorithm 1 introduces a computation of $O(nm)$ scale, this can be mitigated through matrix operations. Additionally, since the sparsity of the adjacency matrix is preserved, new entries in the adjacency matrix do not introduce a significant increase in extra computations for obtaining the final embeddings.

5.2.3 Behavior of Entries in the Adjacency Matrix

We conduct the experiment of LightGCN+AdaCF with the hyperparameter of $\theta = 0.9$ and $\alpha = 2.0$, and the results are shown in Table 8. The case of $0 \to 0$ is omitted because in all three experiments, the datasets used are very sparse, the update ratio is still small, so most 0's remain 0's.

It can be seen that for all three datasets, the ratio of changed elements in the matrix is rather small, and the class of $0 \to 1$ contributes more to the improvements introduced by AdaCF than the class of $0 \to \alpha$. We think this result explains why θ has a larger effect on the experimental results than α, as can be seen in the next section. How to increase the update ratio, and how to introduce valuable updates rather than noises, needs more research in the future.

5.3 Hyper-Parameter Studies

5.3.1 Impact of θ

In Table 4 we present the influence of update threshold θ on AdaCF by showing the results of all methods on Yelp2018 dataset. It can be seen that most methods would perform better as the threshold increases from 0.5 to 0.9. This behavior indicates that when the threshold is set too low, the user and item embeddings used by AdaCF introduce too much noise by adding excessive new entries to the adjacency matrix. Therefore, the threshold should not be set too low during experiments to avoid degrading performance.

However, on LightGCN it is shown that $\theta = 0.7$ achieves the best result. Such fact shows that the best choice of θ is different, while our choice of 0.9 is based on the performance of SimGCL+AdaCF on Yelp2018 dataset, which means the impact of θ to introduce noise or useful information requires further study in the future. The additional results on the Moveilens-1M and Amazon-book datasets can be seen in the Appendix.

5.3.2 Impact of α

In Table 5 we present the influence of update value α on AdaCF by showing the results of all methods on Yelp2018 dataset. It can be seen that in all cases $\alpha = 2.0$ is the best choice. Compared with the cases of $\alpha = 1.0$, most results get better with a higher α, which proves that highlighting those user-item interactions whose embeddings are close and are in the ground truth is very important.

The optimal value of α is sometimes different, as can be seen from Table 5, where $\alpha = 4.0$ can also get good results with LightGCN and XSimGCL. Still, it should be

Table 6 The impact of embedding dimension d on AdaCF. The results are conducted on the Yelp2018 dataset based on the top-20 recommendation list of each model.

Dimension	$d = 16$		$d = 32$		$d = 64$		$d = 128$	
	Recall@20	NDCG@20	Recall@20	NDCG@20	Recall@20	NDCG@20	Recall@20	NDCG@20
LightGCN [2]	0.0553	0.0453	0.0587	0.0488	0.0597	0.0489	0.0597	0.0489
LightGCN+AdaCF	0.0564	0.0458	0.0584	0.0487	0.0601	0.0493	0.0610	0.0502
NCL [25]	0.0499	0.0407	0.617	0.0506	0.0669	0.0551	0.0699	0.0576
NCL+AdaCF	0.0500	0.0409	0.618	0.0507	0.0676	0.0555	0.0695	0.0572
SimGCL [22]	0.0630	0.0520	0.0696	0.0573	0.0722	0.0597	0.0738	0.0609
SimGCL+AdaCF	0.0631	0.0521	0.0693	0.0572	0.0733	0.0602	0.0742	0.0612
XSimGCL [23]	0.0646	0.0531	0.0697	0.0573	0.0722	0.0597	0.0740	0.0610
XSimGCL+AdaCF	0.0646	0.0531	0.0702	0.0577	0.0727	0.0599	0.0745	0.0613

Table 7 The impact of number of layers l on AdaCF. The results are conducted on the Yelp2018 dataset based on the top-20 recommendation list of each model. NCL asks for at least two layers so its $l = 1$ case is omitted.

Method	$l = 1$			$l = 2$			$l = 3$		
	Precision	Recall	NDCG	Precision	Recall	NDCG	Precision	Recall	NDCG
LightGCN [2]	0.0256	0.0570	0.0466	0.0265	0.0587	0.0481	0.0269	0.0597	0.0489
LightGCN+AdaCF	0.0256	0.0566	0.0462	0.0267	0.0590	0.0487	0.0270	0.0601	0.0493
NCL [25]	-	-	-	0.0297	0.0665	0.0546	0.0302	0.0669	0.0551
NCL+AdaCF	-	-	-	0.0302	0.0671	0.0551	0.0304	0.0676	0.0555
SimGCL [22]	0.0313	0.0692	0.0568	0.0325	0.0721	0.0596	0.0326	0.0722	0.0597
SimGCL+AdaCF	0.0314	0.0698	0.0575	0.0322	0.0722	0.0596	0.0330	0.0733	0.0602
XSimGCL [23]	0.0317	0.703	0.0576	0.0328	0.0727	0.0599	0.0326	0.0726	0.0598
XSimGCL+AdaCF	0.0319	0.704	0.0577	0.0329	0.0730	0.0600	0.0327	0.0727	0.0599

Table 8 Results of the elements updating from the experiment of LightGCN+AdaCF.

	ML-1M	Yelp2018	Amazon-Book
$0 \to 1$	3,852	572,064	364,373
$1 \to 1$	458,440	1,207,003	2,326,066
$1 \to \alpha$	1,919	30,256	54,664

noticed that our choice of $\alpha = 2.0$ in Table 2 is based on experiments of SimGCL+AdaCF on Yelp2018 dataset, so the optimal value of α can be different with other methods and datasets. This topic of the power of update value α still requires further study in the future. The additional results on the Moveilens-1M and Amazon-book datasets can be seen in the Appendix.

5.3.3 Impact of d

In Table 6 we show the influence of embedding dimension d on AdaCF by experiments on Yelp2018. It can be seen that for the first three cases $d = 16, 32$ and 64, AdaCF has quite a good positive result on all four models. But for the final case $d = 128$, AdaCF does not perform very well. This may be due to high-dimensional embeddings underfitting during training, leading to the introduction of errors when updating entries. Nonetheless, since most GCN-based CF models typically use embedding dimensions less than 64, the behavior of AdaCF demonstrates its effectiveness in improving existing GCN-based CF models.

5.3.4 Impact of l

In Table 7 we show the influence of network layers d on AdaCF by experiments on Yelp2018. It can be seen that for most cases AdaCF improves the result of the models. This supports our theoretical analysis in Sect. 3.2, where we demonstrate that even for single-layer LightGCN-based models, the fake zero problem significantly impacts embedding updates. Therefore, applying AdaCF is beneficial for training.

It is also quite interesting to note that for all these models, the performance remains consistent between 2-layer and 3-layer models, even with the addition of AdaCF. This observation suggests that considering a 2-step neighborhood is already highly beneficial in LightGCN-based CF models. It also implies that encoding long-range relationships in graphs remains an unsolved challenge.

6. Conclusion

In this paper, we theoretically analyze the problem of LightGCN training process using BPR loss function by gradient analysis, and we introduce AdaCF, a method that updates the adjacency matrix during the training process of GCN-based CF models. By introducing extra entries in the adjacency matrix, models can benefit from the embeddings it has trained. Experimental results show that AdaCF improves the performance of four widely used baseline CF models on three datasets, proving the effectiveness and efficiency of

Table 9 The influence of the update threshold θ on AdaCF. These results are conducted on the Movielens-1M dataset.

Threshold θ	LightGCN+AdaCF			NCL+AdaCF			SimGCL+AdaCF			XSimGCL+AdaCF		
	Precision	Recall	NDCG	Precision	Recall	NDCG	Precision	Recall	NDCG	Precision	Recall	NDCG
0.5	0.200	0.270	0.303	0.202	0.266	0.304	0.162	0.230	0.256	0.200	0.272	0.307
0.6	**0.201**	**0.271**	**0.304**	0.204	0.277	0.312	0.166	0.239	0.264	0.203	0.281	0.315
0.7	0.200	0.268	0.303	0.205	0.278	0.315	0.168	0.244	0.267	0.204	0.283	0.318
0.8	0.199	0.270	0.303	**0.206**	**0.281**	0.318	0.168	0.243	0.267	**0.205**	**0.285**	**0.319**
0.9	0.199	0.270	0.302	**0.206**	0.280	**0.319**	**0.176**	**0.253**	**0.280**	0.200	0.279	0.311
Base	0.198	0.269	0.302	0.205	0.278	0.315	0.175	0.253	0.278	0.199	0.277	0.310

Table 10 The influence of the update threshold θ on AdaCF. These results are conducted on the Amazon-Book dataset.

Threshold θ	LightGCN+AdaCF			NCL+AdaCF			SimGCL+AdaCF			XSimGCL+AdaCF		
	Precision	Recall	NDCG	Precision	Recall	NDCG	Precision	Recall	NDCG	Precision	Recall	NDCG
0.5	0.0160	0.0383	0.0299	0.0167	0.0402	0.0313	0.0203	0.0486	0.0381	**0.0211**	**0.0517**	**0.0404**
0.6	0.0164	0.0393	0.0305	0.0176	0.0430	0.0334	0.0203	0.0486	0.0382	0.0208	0.0505	0.0398
0.7	0.0165	0.0394	0.0306	0.0179	**0.0435**	**0.0338**	0.0204	0.0498	0.0391	0.0206	0.0495	0.0388
0.8	**0.0165**	**0.0398**	**0.0310**	**0.0180**	**0.0435**	0.0337	**0.0207**	**0.0503**	**0.0397**	0.0203	0.0490	0.0382
0.9	0.0164	0.0395	0.0308	**0.0180**	**0.0435**	0.0337	0.0204	0.0492	0.0386	0.0203	0.0487	0.0381
Base	0.0164	0.0394	0.0309	0.0179	0.0432	0.0334	0.0200	0.0480	0.0377	0.0201	0.0487	0.0381

Table 11 The influence of the update value α on AdaCF. These results are conducted on the Movielens-1M dataset.

Value α	LightGCN+AdaCF			NCL+AdaCF			SimGCL+AdaCF			XSimGCL+AdaCF		
	Precision	Recall	NDCG	Precision	Recall	NDCG	Precision	Recall	NDCG	Precision	Recall	NDCG
1.0	**0.200**	**0.272**	**0.304**	0.205	0.279	0.315	0.168	0.243	0.267	0.205	0.286	**0.321**
2.0	0.199	0.270	0.302	**0.206**	**0.280**	**0.319**	**0.176**	**0.253**	**0.280**	0.200	0.279	0.311
3.0	0.198	0.269	0.302	0.205	**0.280**	0.316	0.167	0.240	0.265	**0.206**	**0.287**	**0.321**
4.0	0.199	0.271	0.304	0.205	**0.280**	0.315	0.168	0.241	0.266	0.205	0.285	0.319
Base	0.198	0.269	0.302	0.205	0.278	0.315	0.175	0.253	0.278	0.199	0.277	0.310

Table 12 The influence of the update value α on AdaCF. These results are conducted on the Amazon-Book dataset.

Value α	LightGCN+AdaCF			NCL+AdaCF			SimGCL+AdaCF			XSimGCL+AdaCF		
	Precision	Recall	NDCG	Precision	Recall	NDCG	Precision	Recall	NDCG	Precision	Recall	NDCG
1.0	0.0163	0.0391	0.0304	0.0177	0.0430	0.0335	0.0203	0.0487	0.0383	0.0200	0.0481	0.0378
2.0	0.0164	**0.0395**	0.0308	**0.0180**	**0.0435**	**0.0337**	**0.0204**	**0.0492**	**0.0386**	**0.0203**	**0.0487**	**0.0381**
3.0	0.0164	0.0390	0.0307	0.0179	**0.0437**	**0.0337**	0.0203	0.0489	0.0384	0.0200	0.0481	0.0376
4.0	**0.0165**	0.0394	0.0307	0.0309	0.0177	0.0334	0.0202	0.0486	0.0381	0.0200	0.0482	0.0376
Base	0.0164	0.0394	**0.0309**	0.0179	0.0432	0.0334	0.0200	0.0480	0.0377	0.0201	0.0487	0.0381

AdaCF. For future work, we would like to explore the update strategy to improve the learned embeddings and try to propose a deeper theory about the influence of datasets on CF model training.

References

[1] X. Wang, X. He, M. Wang, F. Feng, and T.-S. Chua, "Neural graph collaborative filtering," Proc. 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval, pp.165–174, 2019.

[2] X. He, K. Deng, X. Wang, Y. Li, Y. Zhang, and M. Wang, "Lightgcn: Simplifying and powering graph convolution network for recommendation," Proc. 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval, p.639–648, 2020.

[3] F.M. Harper and J.A. Konstan, "The movielens datasets: History and context," Acm Trans. interactive intelligent systems (tiis), vol.5, no.4, pp.1–19, 2015.

[4] N. Asghar, "Yelp dataset challenge: Review rating prediction," arXiv preprint arXiv:1605.05362, 2016.

[5] G. Linden, B. Smith, and J. York, "Amazon.com recommendations: Item-to-item collaborative filtering," IEEE Internet Computing, vol.7, no.1, pp.76–80, 2003.

[6] Y. Shen, Y. Wu, Y. Zhang, C. Shan, J. Zhang, B.K. Letaief, and D. Li, "How powerful is graph convolution for recommendation?," Proc. 30th ACM Int. Conf. information & knowledge management, pp.1619–1629, 2021.

[7] C. Wang, Y. Yu, W. Ma, M. Zhang, C. Chen, Y. Liu, and S. Ma, "Towards representation alignment and uniformity in collaborative filtering," Proc. 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, pp.1816–1825, 2022.

[8] J. Wu, X. Wang, F. Feng, X. He, L. Chen, J. Lian, and X. Xie,

"Self-supervised graph learning for recommendation," Proc. 44th International ACM SIGIR Conference on Research and Development in Information Retrieval, pp.726–735, 2021.
[9] D. Goldberg, D. Nichols, B.M. Oki, and D. Terry, "Using collaborative filtering to weave an information tapestry," Communications of the ACM, vol.35, no.12, pp.61–70, 1992.
[10] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," Computer, vol.42, no.8, pp.30–37, 2009.
[11] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," Advances in Neural Information Processing Systems, vol.29, 2016.
[12] R.v.d. Berg, T.N. Kipf, and M. Welling, "Graph convolutional matrix completion," arXiv preprint arXiv:1706.02263, 2017.
[13] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme, "Bpr: Bayesian personalized ranking from implicit feedback," Proc. Twenty-Fifth Conference on Uncertainty in Artificial Intelligence, p.452–461, 2009.
[14] Z. Gao, Z. Cheng, F. Pérez, J. Sun, and M. Volkovs, "Mcl: Mixed-centric loss for collaborative filtering," Proc. ACM Web Conference 2022, pp.2339–2347, 2022.
[15] K. Mao, J. Zhu, J. Wang, Q. Dai, Z. Dong, X. Xiao, and X. He, "Simplex: A simple and strong baseline for collaborative filtering," Proc. 30th ACM Int. Conf. Information & Knowledge Management, pp.1243–1252, 2021.
[16] S. Park, M. Yoon, J.-W. Lee, H. Park, and J. Lee, "Toward a better understanding of loss functions for collaborative filtering," Proc. 32nd ACM Int. Conf. Information and Knowledge Management, pp.2034–2043, 2023.
[17] W. Fan, X. Liu, W. Jin, X. Zhao, J. Tang, and Q. Li, "Graph trend networks for recommendations," arXiv preprint arXiv:2108.05552, 2021.
[18] J. Choi, J. Jeon, and N. Park, "Lt-ocf: learnable-time ode-based collaborative filtering," Proc. 30th ACM Int. Conf. Information & Knowledge Management, pp.251–260, 2021.
[19] I. Chami, Z. Ying, C. Ré, and J. Leskovec, "Hyperbolic graph convolutional neural networks," Advances in Neural Information Processing Systems, vol.32, 2019.
[20] P. Khosla, P. Teterwak, C. Wang, A. Sarna, Y. Tian, P. Isola, A. Maschinot, C. Liu, and D. Krishnan, "Supervised contrastive learning," Advances in neural information processing systems, vol.33, pp.18661–18673, 2020.
[21] X. Liu, F. Zhang, Z. Hou, L. Mian, Z. Wang, J. Zhang, and J. Tang, "Self-supervised learning: Generative or contrastive," IEEE Trans. knowledge and data engineering, vol.35, no.1, pp.857–876, 2021.
[22] J. Yu, H. Yin, X. Xia, T. Chen, L. Cui, and Q.V.H. Nguyen, "Are graph augmentations necessary? simple graph contrastive learning for recommendation," Proc. 45th international ACM SIGIR conference on research and development in information retrieval, pp.1294–1303, 2022.
[23] J. Yu, X. Xia, T. Chen, L. Cui, N.Q.V. Hung, and H. Yin, "Xsimgcl: Towards extremely simple graph contrastive learning for recommendation," IEEE Trans. Knowledge and Data Engineering, 2023.
[24] M. Liu, M. Jian, G. Shi, Y. Xiang, and L. Wu, "Graph contrastive learning on complementary embedding for recommendation," Proc. 2023 ACM Int. Conf. Multimedia Retrieval, pp.576–580, 2023.
[25] Z. Lin, C. Tian, Y. Hou, and W.X. Zhao, "Improving graph collaborative filtering with neighborhood-enriched contrastive learning," Proc. ACM web conference 2022, pp.2320–2329, 2022.
[26] J. Yu, H. Yin, X. Xia, T. Chen, J. Li, and Z. Huang, "Self-supervised learning for recommender systems: A survey," IEEE Trans. Knowledge and Data Engineering, vol.36, no.1, pp.335–355, 2024.

Appendix:

In the appendix, we show the additional results of the influence of θ and α of all methods on the Movielens-1M and Amazon-book dataset.

In Tables 9 and 10, we show the results of the influence of θ of all methods on both datasets. It can be seen from these tables that, all methods on all datasets, generally speaking, perform better as θ increases from 0.5 up to one optimal value, and perform worse as θ decreases from this optimal value to 1.0, which is represented by the base method. The optimal value varies, which means that the optimal θ for the noise-information balance is different with different methods and datasets.

In Tables 11 and 12, we show the results of the influence of α of all methods on both datasets. It can be seen from these tables that, at most times $\alpha = 2.0$ performs the best. For Movielens-1M the performance is a bit more unstable, with LightGCN performing best with $\alpha = 1.0$ and XSimGCL performing best with $\alpha = 3.0$. For Amazon-Book the situation is more stable.



Chi Zhang received the B.S. degree in physics from Peking University, and the M.S. degree in information and communication engineering from The University of Tokyo. He is currently a doctor student at the Department of Information Science and Technology at The University of Tokyo.



Luwei Zhang received the Ph.D degree from The University of Tokyo. He is currently a project researcher at the Department of Information Science and Technology at the University of Tokyo. He is a member of IEEE. His research interests include recommender systems, graph convolution networks, and large language models.



Toshihiko Yamasaki received the Ph.D. degree from The University of Tokyo. He is currently a Professor at Department of Information and Communication Engineering, Graduate School of Information Science and Technology, The University of Tokyo. He was a JSPS Fellow for Research Abroad and a visiting scientist at Cornell University from Feb. 2011 to Feb. 2013. His current research interests include attractiveness computing based on multimedia data, pattern recognition, machine learning, and so on.