MixGCF: An Improved Training Method for Graph Neural Network-based Recommender Systems

Tinglin Huang[†★], Yuxiao Dong[‡], Ming Ding[◇], Zhen Yang[◇], Wenzheng Feng[◇]
Xinyu Wang[†], Jie Tang[◇§]
[†]Zhejiang University, [‡]Facebook AI, [◇]Tsinghua University
tinglin.huang@zju.edu.cn,yuxiaod@fb.com,dm18@mails.tsinghua.edu.cn,zheny2751@gmail.com
fwz17@mails.tsinghua.edu.cn,wangxinyu@zju.edu.cn,jietang@tsinghua.edu.cn

ABSTRACT

Graph neural networks (GNNs) have recently emerged as state-of-the-art collaborative filtering (CF) solution. A fundamental challenge of CF is to distill negative signals from the implicit feedback, but negative sampling in GNN-based CF has been largely unexplored. In this work, we propose to study negative sampling by leveraging both the user-item graph structure and GNNs' aggregation process. We present the MixGCF method—a general negative sampling plugin that can be directly used to train GNN-based recommender systems. In MixGCF, rather than sampling raw negatives from data, we design the hop mixing technique to synthesize hard negatives. Specifically, the idea of hop mixing is to generate the synthetic negative by aggregating embeddings from different layers of raw negatives' neighborhoods. The layer and neighborhood selection process are optimized by a theoretically-backed hard selection strategy. Extensive experiments demonstrate that by using MixGCF, state-of-the-art GNN-based recommendation models can be consistently and significantly improved, e.g., 26% for NGCF and 22% for LightGCN in terms of NDCG@20.

CCS CONCEPTS

• **Information systems** → **Recommender systems**; • **Mathematics of computing** → **Graph algorithms**.

KEYWORDS

Collaborative Filtering; Recommender Systems; Graph Neural Networks; Negative Sampling

ACM Reference Format:
Tinglin Huang, Yuxiao Dong, Ming Ding, Zhen Yang, Wenzheng Feng, Xinyu Wang, Jie Tang. 2021. MixGCF: An Improved Training Method for Graph Neural Network-based Recommender Systems. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '21), August 14–18, 2021, Virtual Event, Singapore.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3447548.3467408

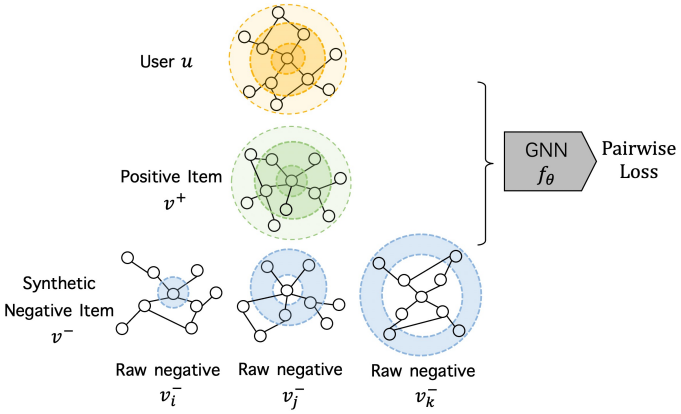


Figure 1: An illustration of hop mixing for synthesizing a negative item in MixGCF, where the user and item are represented by their ego-networks respectively, and the highlighted circles indicate the selected neighbors.

Codes are available at https://github.com/huangtinglin/MixGCF.
[§]Jie Tang is the corresponding author.
[★]The work has been done when the author was visiting Tsinghua University.

KDD '21, August 14–18, 2021, Virtual Event, Singapore.

ACM ISBN 978-1-4503-8332-5/21/08…$15.00
https://doi.org/10.1145/3447548.3467408

1 INTRODUCTION

Recommender systems have been widely used for avoiding information overload in applications as diverse as online shopping [53], social network [29], advertising [15], and Web search [17]. Its goal is to provide users with personalized information feeding, that is, for each user, the problem of recommendation is to predict the items that she or he will consume.

Among the most promising techniques for this problem has been the usage of collaborative filtering (CF) [21], which models users' historical interactions with items to profile users and items for predicting future interactions. To date, the most prevalent CF solutions are to project users and items into the latent embedding space, such as matrix factorization [23] and neural networks [14] based techniques. To further improve the embedding quality, one prominent direction is to model user-item interactions as a graph and leverage graph neural networks (GNNs) [8, 11, 20] to incorporate structural information into the embeddings. Notably, the GNN-based recommendation models, such as PinSage [49], NGCF [43], and LightGCN [13], have generated state-of-the-art performance with Web-scale applications.

The typical flow of (GNN-based) recommender systems is relatively straightforward. Given a user-item interaction graph, it begins with defining an aggregation function over the structure to propagate neighborhood information, usually followed by a pooling operation for outputting both user and item embeddings. Similar to conventional recommendation methods, its objective function is designed to prefer an observed user-item pair (as positive) to unobserved ones (as negative pairs). Take the widely-adopted BPR loss [31] for example, for each user and one of her positive items, we conduct negative sampling to pick one item as the negative from those she never interacts with.

Essentially, the negative samples play a decisive role in the performance of (GNN-based) recommendation models. Commonly, a uniform distribution is used for negative sampling [13, 43]. To improve the quality of negative samples, studies have attempted to design new sampling distributions for prioritizing informative negatives [17, 28, 40, 44, 48, 52]. In doing so, the model would be challenged and forced to distinguish their differences at a finer granularity. To improve negative sampling in GNNs, PinSage [49] samples the negatives based on their PageRank scores and MCNS [48] re-designs both positive and negative sampling distributions with their structural correlations in mind. However, these attempts in GNNs only focus on improving negative sampling in the discrete graph space, ignoring GNNs' unique neighborhood aggregation process in the embedding space.

Contributions. In this work, we propose to design negative sampling strategies for better training GNN-based recommender systems. We present a simple MixGCF framework for generating hard negative samples. Instead of directly sampling real negatives from the data, MixGCF takes inspirations from data augmentation and metric learning to synthesize negative samples by leveraging the underlying GNN-based recommender.

To make synthetic negatives hard for the recommendation models, MixGCF designs two strategies: positive mixing and hop mixing. In positive mixing, we introduce an interpolation mixing method to pollute the embeddings of raw negative samples by injecting information from positive ones into them. In hop mixing, we sample several raw negative samples, e.g., v_i^-, v_j^-, and v_k^-, in Figure 1 and generate the embedding of the synthetic negative v^- by using their polluted embeddings aggregated from selected hops of their neighbors. For example, as indicated by the blue circles, hop 0, 1, and 2 are selected from v_i^-, v_j^-, and v_k^-, respectively. It is worth mentioning that the selection of which hop from whose neighborhood is guided by a designed hard selection strategy.

We perform extensive experiments on benchmark recommendation datasets. Experimental results show that by replacing their default negative sampler to MixGCF, the state-of-the-art GNN-based recommendation models can be consistently improved, such as average relative increase of 22% for LightGCN and 26% for NGCF in terms of NDCG@20. Furthermore, we compare MixGCF with various negative sampling techniques, the results of which demonstrate the empirical advantage of MixGCF over them.

In summary, our work makes the following contributions:

- Introduce the idea of synthesizing negative samples rather than directly sampling negatives from the data for improving GNN-based recommender systems.
- Present a general MixGCF framework with the hop mixing and positive mixing strategies that can be naturally plugged into GNN-based recommendation models.
- Demonstrate the significant improvements that MixGCF brings to GNN recommenders, as well as its consistent outperformance over a diverse set of negative sampling techniques.

2 PRELIMINARIES AND PROBLEM

In this section, we first review the overall process of graph neural network (GNN) based collaborative filtering (CF) for recommender systems. We then introduce the studied problem that concerns the training of the above process.

2.1 Graph Neural Networks for Recommendation

Commonly, the input of recommendation systems includes a set of users $\mathcal{U} = \{u\}$, items $\mathcal{V} = \{v\}$, and users' implicit feedback $O^+ = \{(u, v^+)|u \in \mathcal{U}, v^+ \in \mathcal{V}\}$, where each pair indicates an interaction between user u and item v^+. The goal is to estimate the user preference towards items.

Recently, studies have shown that GNN-based CF models offer promising results for this task [13, 43, 49]. The main idea is to measure user u's preference on item v based on their multi-hop neighbors. Specifically, these techniques generate the latent representations of the target user and item, the inner product of which is used to quantify the preference, that is, $\hat{y}(u, v) = \mathbf{e}_u^{*\top}\mathbf{e}_v^*$.

Next, we briefly introduce the process of GNN-based CF, including aggregation, pooling, and its optimization with negative sampling.

Aggregation. Each user and item are associated with an initial embedding $\mathbf{e}_u$ and $\mathbf{e}_v$ as its representation vector, respectively. In order to exploit the CF signal from neighbor nodes, GNN-based recommendation models apply different aggregation functions to propagate information over neighbors [3, 13, 37, 43, 49]. Take LightGCN for example, its aggregation process is:

$$\mathbf{e}_u^{(l+1)} = \sum_{i \in N_u} \frac{1}{\sqrt{|N_u||N_i|}}\mathbf{e}_i^{(l)}, \ \mathbf{e}_v^{(l+1)} = \sum_{j \in N_v} \frac{1}{\sqrt{|N_v||N_j|}}\mathbf{e}_j^{(l)}. \quad (1)$$

where $\mathbf{e}_u^{(l)}, \mathbf{e}_v^{(l)}$ are the embeddings of user u and item v at l-th layer of GNN, $\mathcal{N}_u$ denotes the set of items that interact with user u, and $\mathcal{N}_v$ denotes the set of users that interact with item v. By stacking multiple aggregation layers, each user/item can gather the information from its higher-order neighbors. For simplicity, we use $\mathbf{e}^{(l)}$ as the l-th layer embedding in the following sections.

Pooling. Different from GNNs for node classification [11, 20], where representations in the final layer are used, the GNN-based CF models usually adopt the pooling operation to generate the final representations for users and items. According to [47], this can help avoid over-smoothing and determine the importance of a node's subgraph information at different ranges.

Specifically, at the final layer L, the pooling function is applied to generate the final user/item representations $\mathbf{e}_u^*/\mathbf{e}_v^*$. For example, LightGCN uses sum-based pooling:

$$\mathbf{e}_u^* = \sum_{l=0}^{L} \lambda_l \mathbf{e}_u^{(l)}, \mathbf{e}_v^* = \sum_{l=0}^{L} \lambda_l \mathbf{e}_v^{(l)}, \quad (2)$$

and NGCF uses concat-based pooling:

$$\mathbf{e}_u^* = \mathbf{e}_u^{(0)}||\cdots||\mathbf{e}_u^{(L)}, \mathbf{e}_v^* = \mathbf{e}_v^{(0)}||\cdots||\mathbf{e}_v^{(L)}, \tag{3}$$

Optimization with Negative Sampling. The task of learning to rank is to provide a user with a ranked list of items by assuming that the items preferred by the user should rank higher than others. However, the ranked item list for each user oftentimes can only be inferred from the implicit feedback that only consists of the positive observations. One straightforward solution is to assume that users prefer the observed items over all unobserved ones. Due to the large size of unobserved items (usually in $O(|\mathcal{V}|^2)$), the learning objective is usually simplified by negative sampling as the BPR loss [31]:

$$\max \prod_{v^+, v^- \sim f_S(u)} P_u(v^+ > v^- | \Theta) \tag{4}$$

where v^+ and v^- denote the positive and negative items, respectively, $P_u(a > b)$ represents user u prefers item a over b, $f_S(u)$ is the distribution of negative sampling, and Θ is the parameter of the model. Most recommendation methods consider negative sample from a uniform distribution ($f_S(u)$ as $f_{\text{uniform}}(u)$) [13, 14, 29, 31, 43].

2.2 The Negative Sampling Problem

According to the loss function in Eq. (4), the negative sampling strategy plays a critical role in the model training of recommendation. Intuitively, the negative samples close to positive ones, a.k.a. hard negative samples, can make the models better learn the boundary between positive and negative instances [49]. To this end, several attempts have been made to sample hard negatives to improve the optimization of general recommender systems [6, 17, 30, 52].

However, negative sampling for GNN-based recommenders has remained largely unexplored. Notably, early attempts—PinSage [49] and MCNS [48]—focus on improving the sampling distributions at the discrete structure level, seeking better hard negative (raw) nodes in the graph. In this work, we ask the question of whether we can synthesize harder negative samples in the continuous space, based on the GNN underlying the recommender.

3 THE MIXGCF METHOD

MixGCF is a general algorithm for negative sampling in GNN-based recommendation. It can be directly plugged into existing GNN-based recommendation algorithms, such as LightGCN and NGCF.

Instead of sampling real items from the data as negative ones [28, 31, 40, 48, 52], MixGCF proposes to synthesize informative (and fake) negative items based on the graph structure for training GNN-based CF recommendation models. Specifically, MixGCF introduces the positive mixing and hop mixing techniques to synthesize negative samples by mixing information from different local graphs.

The flow of MixGCF is illustrated in Figure 2. In positive mixing, we develop an interpolation mixing method to inject information from positive samples to negative ones, making hard negative candidates. In hop mixing, we first leverage a hard negative selection strategy to extract unique information from each of the hard negatives generated above, and then use the pooling operation to combine the diverse information extracted for creating the fake but informative negative items.

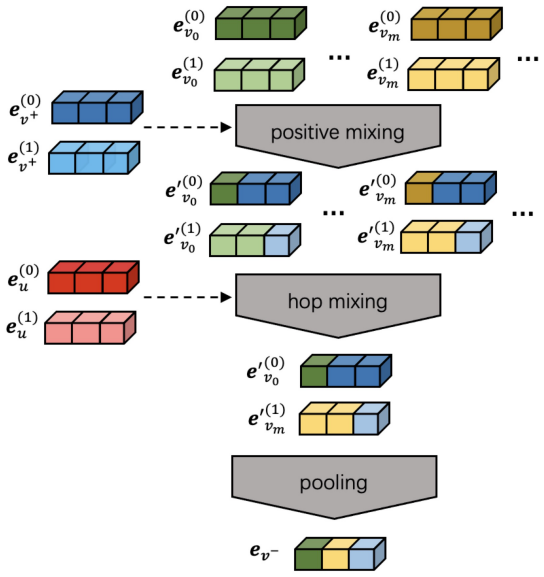


Figure 2: An overview of MixGCF, where $\mathbf{e}^{(l)}$ denotes the l-th layer embedding of node e, and $\mathbf{e}'^{(l)}$ denotes the l-th layer embedding generated by positive mixing.

3.1 Positive Mixing

Recall that in a L-layer GNN, for each item v, we can have $L+1$ embeddings for v, each of which $\mathbf{e}_v^{(l)}$ corresponds to the embedding that is aggregated with l layers ($0 \le l \le L$).

To fake the negative v_i^- with its embedding $\mathbf{e}_{v^-}$ in Figure 2, we first follows the convention [17, 49] to select M negative items to form the candidate set $\mathcal{M}$, with M usually being much smaller than the number of items in the data. These M negative items can form a candidate negative embedding set $\mathcal{E} = \{\mathbf{e}_{v_m}^{(l)}\}$ of size $M \times (L+1)$.

A very recent study [17] suggests that recommendation models usually operate on an input space that mainly comprises of easy negatives, therefore we propose to improve the quality of the embeddings $\mathcal{E}$ of candidate negatives. Inspired by *mixup* [18, 51], we introduce the idea of positive mixing to inject positive information $\mathbf{e}_{v^+}$ into negative embeddings in $\mathcal{E}$. *mixup* is an interpolation based data augmentation method, which enforces the model to output linearly between the training data. Specifically, for each candidate negative embedding $\mathbf{e}_{v_m}^{(l)} \in \mathcal{E}$, the positive mixing operation is formalized as:

$$\mathbf{e}'^{(l)}_{v_m} = \alpha^{(l)}\mathbf{e}_{v^+}^{(l)} + (1-\alpha^{(l)})\mathbf{e}_{v_m}^{(l)}, \alpha^{(l)} \in (0,1), \tag{5}$$

where $\alpha^{(l)}$ is the mixing coefficient that is uniformly sampled for each hop l. Note that the mixing coefficient of *mixup* is sampled from a beta distribution Beta(β, β), which has a heavy impact on the model's generalization ability [50]. To decouple the impact, the mixing coefficient $\alpha^{(l)}$ in our positive mixing is uniformly sampled from $(0, 1)$ (cf. Section 4.3 for the empirical discussions on $\alpha^{(l)}$).

Let $\mathcal{E}'$ be the enhanced embedding set for the candidate negatives $\mathcal{M}$. Positive mixing enhances the negatives by (1) injecting positive information into negative samples, which can help enforce the optimization algorithm to exploit the decision boundary harder, and (2) introducing stochastic uncertainty into them with the random mixing coefficient.

3.2 Hop Mixing

With the embeddings $\mathcal{E}' = \{\mathbf{e}'^{(l)}_{v_m}\}$ of candidate negative items enhanced by positive mixing, we present the hop mixing technique to generate the synthetic negative item v^- and its embedding $\mathbf{e}_{v^-}$. The main idea of hop mixing is to leverage the hierarchical (layer-based) aggregation process in GNNs.

Specifically, for each layer l ($0 \le l \le L$), we sample one candidate negative embedding $\mathbf{e}'^{(l)}_{v_x}$ ($1 \le x \le M$) from $\mathcal{E}'^{(l)}$, which contains all the l-th layer embeddings of the candidate negative items in $\mathcal{M}$. Take L=2 for example, we can sample $\mathbf{e}'^{(0)}_{v_a}$, $\mathbf{e}'^{(1)}_{v_b}$, and $\mathbf{e}'^{(2)}_{v_c}$ from $\mathcal{E}'$. Note that a, b, and c are not necessary to be distinct.

The idea of hop mixing is then to combine all the L+1 embeddings selected by layer to generate the representation $\mathbf{e}_{v^-}$ of the (fake) negative v^-. Specifically, the representation is synthesized by fusing all candidate embeddings by the pooling operation:

$$\mathbf{e}_{v^-} = f_{\text{pool}}\left(\mathbf{e}'^{(0)}_{v_x}, \cdots, \mathbf{e}'^{(L)}_{v_y}\right), \tag{6}$$

where $\mathbf{e}'^{(l)}_{v_x}$ denotes the l-th layer embedding of v_x that is sampled at layer l, and $f_{\text{pool}}(\cdot)$ applies the same pooling operation used in the current GNN-based recommender.

The essential question for hop mixing is how to effectively sample candidate embedding $\mathbf{e}'^{(l)}_{v_x}$ ($1 \le x \le M$) from $\mathcal{E}'^{(l)}$ at each layer l. Notably, a recent study on negative sampling for graph representation learning (MCNS) [48] theoretically shows that the expected risk of the optimal parameter $\mathbf{e}_u^T\mathbf{e}_v$ between the expected loss $J(\theta^*)$ and empirical loss $J(\theta_T)$ satisfies:

$$\mathbb{E}\left[||(\theta_T - \theta^*)_u||^2\right] = \frac{1}{T}\left(\frac{1}{p_d(v|u)} - 1 + \frac{1}{Kp_n(v|u)} - \frac{1}{K}\right), \tag{7}$$

where $p_d(v|u), p_n(v|u)$ denote the estimated positive distribution and negative distribution, respectively, T is the number of node pairs, and K is the number of negatives for each user recruiting in the loss. This derivation suggests that if $p_n(v|u)$ is proportional to $p_d(v|u)$, the expected risk is only dependent on the $p_d(v|u)$, and the interaction probability between a user-item pair with a high inner product score can be estimated accurately.

Based on the above theory, the suggested way for negative sampling is to select the negative according to the estimated positive distribution. Here we apply the inner product score to approximate the positive distribution and pick the candidate sample with the highest score, which is also called the hard negative select strategy [30, 52]. Formally, the hard selection strategy at the l-th layer is implemented as:

$$\mathbf{e}'^{(l)}_{v_x} = \underset{\mathbf{e}'^{(l)}_{v_m} \in \mathcal{E}'^{(l)}}{\arg\max}\, f_Q(u, l) \cdot \mathbf{e}'^{(l)}_{v_m}, \tag{8}$$

where $\cdot$ is the inner product operation, and $f_Q(u, l)$ is a query mapping that returns an embedding related to the target user u for the l-th hop.

Algorithm 1: The training process with MixGCF

Input: Training set $\{(u, v^+)\}$, Recommender f_{GNN}, Number of negative candidate M, Number of aggregation layers L.

for $t = 1, 2, \cdots, T$ **do**
- Sample a mini-batch of positive pairs $\{(u, v^+)\}$.
- Initialize loss $\mathcal{L} = 0$.
- // Negative Sampling via MixGCF.
- **for** *each* (u, v^+) *pair* **do**
  - Get the aggregated embeddings of each node by f_{GNN}.
  - Get the set of candidate negative embeddings $\mathcal{E}$ by uniformly sampling M negatives.
  - Get the updated set of negative candidate $\mathcal{E}'$ by (5).
  - Synthesize a hard negative $\mathbf{e}_{v^-}$ based on $\mathcal{E}'$ by (6).
  - $\mathcal{L} = \mathcal{L} + \ln \sigma(\mathbf{e}_u \cdot \mathbf{e}_{v^-} - \mathbf{e}_u \cdot \mathbf{e}_{v^+})$.
- **end**
- Update θ by descending the gradients $\nabla_\theta \mathcal{L}$.

end

The query in Eq. (8) is dependent on the pooling module of the GNN used for recommendation. As discussed in Section 2, the mainstream pooling module in GNN-based recommendation [13, 43] can be categorized into sum-based and concat-based pooling operations. Therefore, there are two options for the inner product between the target user embedding $\mathbf{e}_u$ and the embedding of the synthesized negative $\mathbf{e}_{v^-}$:

- Sum-based pooling: $\mathbf{e}_u \cdot \mathbf{e}_{v^-} = \sum_{l=0}^{L} \lambda_l \mathbf{e}_u \cdot \mathbf{e}^{(l)}_{v^-}$
- Concat-based pooling: $\mathbf{e}_u \cdot \mathbf{e}_{v^-} = \sum_{l=0}^{L} \mathbf{e}^{(l)}_u \cdot \mathbf{e}^{(l)}_{v^-}$.

To make the selection process in Eq. (8) consistent with the pooling used in the GNN recommender, we let $f_Q(u, l) = \mathbf{e}_u$ for sum-based pooling and $f_Q(u, l) = \mathbf{e}^{(l)}_u$ for concat-based pooling, respectively.

3.3 Optimization with MixGCF

Now, we can use the proposed MixGCF method as the negative sampling method $f_S(\cdot)$ in the loss function (Eq. (4)) to optimize the parameters of the GNN-based recommendation model. Straightforwardly, the BPR loss function can be updated as

$$\mathcal{L}_{\text{BPR}} = \sum_{\substack{(u,v^+)\in O^+ \\ \mathbf{e}_{v^-} \sim f_{\text{MixGCF}}(u,v^+)}} \ln \sigma(\mathbf{e}_u \cdot \mathbf{e}_{v^-} - \mathbf{e}_u \cdot \mathbf{e}_{v^+}), \tag{9}$$

where $\sigma(\cdot)$ is the sigmoid function, O^+ is the set of the positive feedback, and $\mathbf{e}_{v^-} \sim f_{\text{MixGCF}}(u, v^+)$ represents that the instance (embedding) $\mathbf{e}_{v^-}$ is synthesized by the proposed MixGCF method.

3.4 Discussions on MixGCF

General Plugin. Observed from Eq. (9), the proposed negative sampling method—MixGCF—can be naturally plugged into the ranking loss. In addition, MixGCF is a simple and non-parametric method. Altogether, these make MixGCF a general technique for improving a set of GNN-based recommendation models.

Data Augmentation. Unlike the prior efforts which elaborate versatile strategies to sample an existing negative item, MixGCF

instead proposes the paradigm of synthesizing negative items based on the hop-wise sampling. Such a method could be also understood from the perspective of data augmentation, since the synthesized instance is conformed to but different from the existing instances [33]. This enables the recommender to be trained on the more sophisticated data, leading to an improved generalization.

Approximation of Multiple Negatives. As shown in metric learning [16, 32, 34], recruiting multiple negative instances in loss function for each update can speed up the convergence of the underlying model and offer better performances. Instead of directly sampling multiple negatives, MixGCF naturally provides a low-cost approximation of them by hop mixing, potentially making it benefit from the metric learning conclusion above.

3.5 Time Complexity

The time cost of MixGCF mainly comes from two parts. For the hop mixing module, the computational complexity of the hop-grained negative selection scheme is $O(MLd)$, where M is the size of the negative candidate set, L is the number of GNN layers, and d is the embedding dimension. For the positive mixing module, the complexity of the linear combination is $O(MLd)$, which is equivalent to the hop mixing module's. Therefore, the time complexity of MixGCF is $O(MLd)$.

4 EXPERIMENTS

We evaluate our proposed MixGCF method on three benchmark datasets with three representative GNN-based CF modes—LightGCN, NGCF, and PinSage—as the base recommenders. We compare MixGCF with other state-of-the-art negative sampling methods to demonstrate the superiority of our proposed method. We then conduct the hyper-parameter study and ablation study to analyze the behavior of MixGCF.

4.1 Experimental Settings

Dataset Description. In this paper, we evaluate our method on three benchmark datasets: Alibaba [48], Yelp2018 [43], and Amazon [48], which are publicly available. Each dataset consists of the user-item interactions solely, as summarized in Table 1. We follow the same settings described in [43, 48] to split the datasets into training, validation, and testing sets.

Evaluation Metrics. We choose the widely-used Recall@N and NDCG@N as the evaluation metrics ($N = 20$ by default). Unlike the previous studies [14, 39, 48] that conduct the sampled metrics, we compute Recall@N and NDCG@N by the all-ranking protocol [22]. We report the average metrics of all users in each testing set.

Recommender. To verify the effectiveness of MixGCF, we perform experiments on three GNN-based recommenders as follows. The detailed descriptions are in Section 2.1.

- **LightGCN** [13]: This is a state-of-the-art CF method. LightGCN claims that the design of NGCF is heavy and burdensome since each node in the user-item graph is only described by an ID. LightGCN omits the non-linear transformation and applies the sum-based pooling module to achieve better empirical performance.

Table 1: Statistics of the datasets.

Dataset	#Users	#Items	#Interactions	Density
Alibaba	106, 042	53, 591	907, 407	0.00016
Yelp2018	31, 668	38, 048	1, 561, 406	0.00130
Amazon	192, 403	63, 001	1, 689, 188	0.00014

Table 2: Overall Performance Comparison.

	Alibaba		Yelp2018		Amazon	
	Recall	NDCG	Recall	NDCG	Recall	NDCG
LightGCN+RNS	0.0584	0.0275	0.0628	0.0515	0.0398	0.0177
LightGCN+DNS	0.0737	0.0343	0.0695	0.0571	0.0449	0.0211
LightGCN+IRGAN	0.0605	0.0280	0.0641	0.0527	0.0412	0.0185
LightGCN+AdvIR	0.0583	0.0273	0.0624	0.0510	0.0401	0.0185
LightGCN+MCNS	0.0632	0.0284	0.0658	0.0529	0.0423	0.0192
LightGCN+MixGCF	**0.0763***	**0.0357***	**0.0713***	**0.0589***	**0.0460***	**0.0216***
NGCF+RNS	0.0426	0.0197	0.0577	0.0469	0.0294	0.0123
NGCF+DNS	0.0453	0.0207	0.0650	0.0529	0.0312	0.0130
NGCF+IRGAN	0.0435	0.0200	0.0615	0.0502	0.0283	0.0120
NGCF+AdvIR	0.0440	0.0203	0.0614	0.0500	0.0318	0.0134
NGCF+MCNS	0.0430	0.0200	0.0625	0.0501	0.0313	0.0136
NGCF+MixGCF	**0.0544***	**0.0262***	**0.0688***	**0.0566***	**0.0350***	**0.0154***
PinSage+RNS	0.0196	0.0085	0.0410	0.0328	0.0193	0.0080
PinSage+DNS	0.0405	0.0183	0.0590	0.0488	0.0217	0.0088
PinSage+IRGAN	0.0200	0.0090	0.0422	0.0343	0.0248	0.0088
PinSage+AdvIR	0.0196	0.0090	0.0387	0.0313	0.0243	0.0087
PinSage+MCNS	0.0212	0.0095	0.0432	0.0349	0.0202	0.0088
PinSage+MixGCF	**0.0489***	**0.0226***	**0.0632***	**0.0525***	**0.0273***	**0.0124***

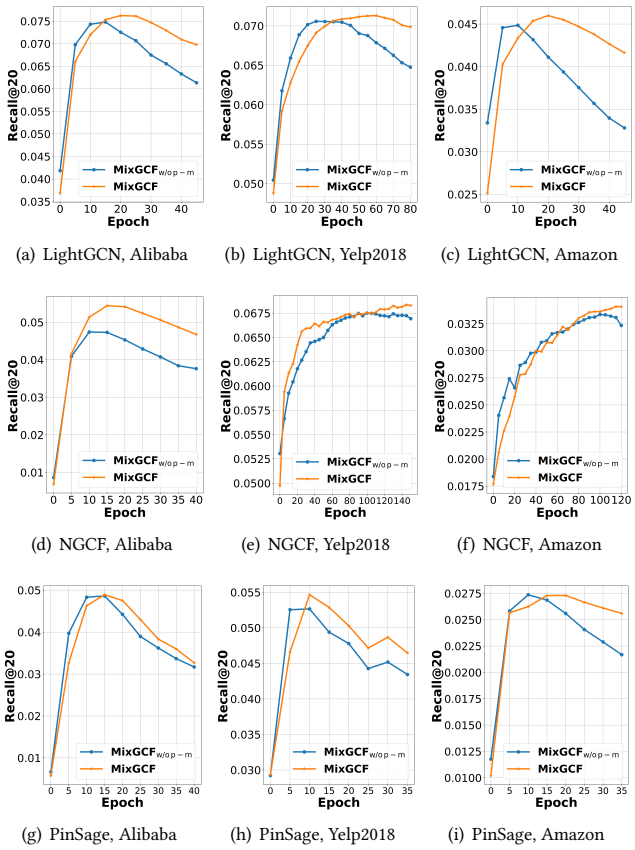
- **NGCF** [43]: Inspired by Graph Convolution Network [11, 20], NGCF applies the message-passing scheme on the bipartite user-item graph to exploit the high-order neighbors' information. To be specific, each node obtains the transformed representations of its multi-hop neighbors by recursively aggregating the message propagated from the adjacent neighbors.
- **PinSage** [49]: As an inductive variant of GNN, PinSage designs a random-walk based sampling method to sample neighbors for each node and proposes to sample hard negative based on PageRank score. We apply a sum-based pooling strategy to generate the embeddings of nodes.

Baselines. We compare the proposed method, MixGCF, with static (RNS), hard negative (DNS), GAN-based (IRGAN and AdvIR), and graph-based (MCNS) sampler, as follows:

- **RNS** [31]: Random negative sampling (RNS) strategy applies uniform distribution to sample negative items. It is independent of the recommender and is applied in various tasks.
- **DNS** [52]: Dynamic negative sampling (DNS) strategy is the state-of-the-art sampler [17, 30], which adaptively selects the negative item scored highest by the recommender. Such the negative is viewed as the hard negative and can provide a large gradient to the parameters.
- **IRGAN** [40]: IRGAN integrates the recommender into a generative adversarial net (GAN) where the generator performs as a sampler to pick the negative for confusing the recommender.
- **AdvIR** [28]: AdvIR is also an adversarial sampler that incorporates adversarial sampling with adversarial training by adding adversarial perturbation.
- **MCNS** [48]: Markov chain Monte Carlo negative sampling (MCNS) is the pioneer to theoretically analyze the impact of negative sampling in link prediction. Based on the deduced theory, MCNS

Table 3: Performance of MixGCF without positive mixing.

	Alibaba		Yelp2018		Amazon	
	Recall	NDCG	Recall	NDCG	Recall	NDCG
LightGCN+MixGCF$_{\text{w/o p-m}}$	0.0748	0.0347	0.0705	0.0581	0.0448	0.0212
NGCF+MixGCF$_{\text{w/o p-m}}$	0.0479	0.0226	0.0674	0.0555	0.0332	0.0147
PinSage+MixGCF$_{\text{w/o p-m}}$	0.0486	0.0231	0.0526	0.0432	0.0273	0.0120



(a) LightGCN, Alibaba (b) LightGCN, Yelp2018 (c) LightGCN, Amazon

(d) NGCF, Alibaba (e) NGCF, Yelp2018 (f) NGCF, Amazon

(g) PinSage, Alibaba (h) PinSage, Yelp2018 (i) PinSage, Amazon

Figure 3: Impact of positive mixing.

proposes to sample negative by approximating the positive distribution and accelerate the process by Metropolis-Hastings algorithm.

Parameter Settings. We implement our MixGCF and recommendations models by PyTorch and will release our code upon acceptance. For each recommender, we fix the embedding size as 64, the optimizer as Adam [19], and use Xavier [9] to initialize the parameters. We set the batch size as 1024 for NGCF, 2048 for PinSage, and 2048 for LightGCN. We conduct a grid search to find the optimal settings for each recommender: the learning rate is searched in $\{0.0001, 0.0005, 0.001\}$, and L_2 regularization is tuned in $\{10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}\}$. The number of sampled neighbors in each layer of PinSage is fixed as 5. As for MCNS, the maximum depth of dfs is 100 by default. The candidate size M in MixGCF and DNS are searched in $\{8, 16, 32, 64\}$ and we report its effect on MixGCF in Section 4.3. The aggregation module number of the recommender L is set as 3 by default, and we evaluate the impact of L in Section 4.3. The detailed settings of MixGCF and recommenders are provided in Appendix A.

Table 4: Impact of the number of aggregation modules (L).

	Alibaba		Yelp2018		Amazon	
	Recall	NDCG	Recall	NDCG	Recall	NDCG
LightGCN+MixGCF-1	0.0651	0.0309	0.0684	0.0564	0.0403	0.0193
LightGCN+MixGCF-2	0.0726	0.0335	0.0707	0.0582	0.0438	0.0209
LightGCN+MixGCF-3	0.0763	0.0357	0.0713	0.0589	0.0460	0.0216
NGCF+MixGCF-1	0.0484	0.0234	0.0647	0.0526	0.0320	0.0151
NGCF+MixGCF-2	0.0545	0.0262	0.0664	0.0542	0.0345	0.0153
NGCF+MixGCF-3	0.0544	0.0262	0.0688	0.0566	0.0350	0.0154
PinSage+MixGCF-1	0.0487	0.0231	0.0639	0.0526	0.0289	0.0130
PinSage+MixGCF-2	0.0472	0.0223	0.0627	0.0519	0.0278	0.0121
PinSage+MixGCF-3	0.0489	0.0226	0.0632	0.0525	0.0273	0.0124

Table 5: Impact of the size of candidate set (M).

		Alibaba		Yelp2018		Amazon	
		Recall	NDCG	Recall	NDCG	Recall	NDCG
LightGCN+MixGCF	M=8	0.0697	0.0311	0.0664	0.0547	0.0443	0.0203
	M=16	0.0728	0.0339	0.0684	0.0562	**0.0460***	**0.0216***
	M=32	**0.0763***	**0.0357***	0.0703	0.0579	0.0455	0.0215
	M=64	0.0744	0.0355	**0.0713***	**0.0589***	0.0430	0.0206
NGCF+MixGCF	M=8	0.0468	0.0201	0.0627	0.0512	0.0350	0.0147
	M=16	0.0518	0.0237	0.0658	0.0539	0.0333	0.0144
	M=32	0.0532	0.0253	0.0682	0.0560	0.0347	0.0154
	M=64	**0.0544***	**0.0262***	**0.0688***	**0.0566***	**0.0350***	**0.0154***
PinSage+MixGCF	M=8	0.0178	0.0075	0.0495	0.0402	0.0204	0.0072
	M=16	0.0388	0.0173	0.0546	0.0448	0.0207	0.0084
	M=32	0.0435	0.0195	0.0608	0.0501	0.0238	0.0106
	M=64	**0.0489***	**0.0226***	**0.0632***	**0.0525***	**0.0273***	**0.0124***

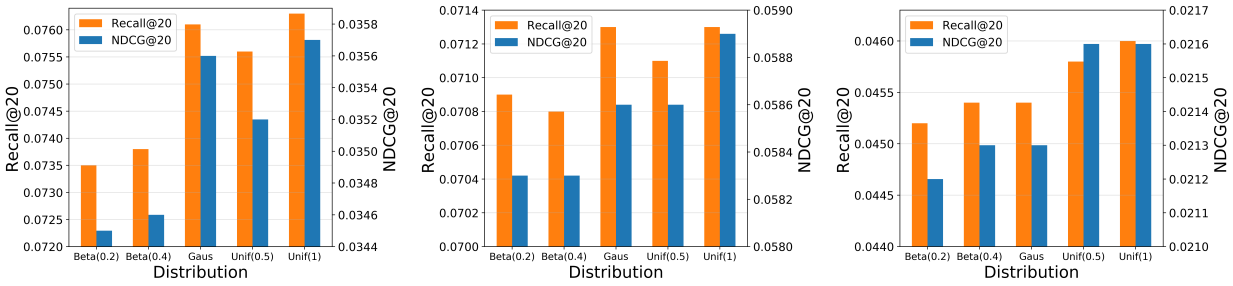
4.2 Performance Comparison

We report the detailed performance comparison in Table 2 where we highlight the results of the best baselines (underlined) and our MixGCF (starred). The observations are as followed:

- LightGCN consistently outperforms NGCF and PinSage by a large margin on three datasets, verifying that the nonlinearity and weight matrix are useless for collaborative filtering.
- MixGCF yields the best performance on all the datasets. These improvements are attributed to the following reasons: (1) Through hop mixing technique, MixGCF augments the negative samples, which improves the generalization of recommender; (2) The synthesized hard negative that incorporates the different semantics information from multiple instances offers a informative gradient to the recommender.
- The relative improvements on NGCF and PinSage are more significant than on LightGCN. Some possible reasons are: (1) The burdensome design offers a larger parameter space, meaning that NGCF and PinSage may benefit more from an informative negative; (2) As a state-of-the-art CF model, LightGCN is capable of discriminating between the positive and easy negative item. Thus, the impact of synthesized hard negative to LightGCN is not as significant as the other two recommendation models.
- DNS performs as the strongest baseline in most cases, which is consistent with previous studies [17, 30, 44]. It reveals that selecting the hardest negative provides a meaningful gradient to guide the model.

Running Time. Take LightGCN as the example, the time elapsed for training per epoch of RNS, DNS, MixGCF, IRGAN, AdvIR and MCNS are about 23s, 78s, 88s, 93s, 100s, 200s as for Alibaba; 45s, 100s, 128s, 156s, 170s and 326s as for Yelp2018; and 50s, 90s, 100s, 200s, 232s, and 400s approximately as for Amazon.

Overall, the experimental results demonstrate the superiority of our proposed MixGCF. Specifically, it outperforms all baselines across



(a) Alibaba (b) Yelp2018 (c) Amazon

Figure 4: Performance comparison over different distributions, i.e., Beta, Gaussian and Uniform distribution, of random coefficient ($\alpha^{(l)}$) on three datasets.

Table 6: Impact of the number of negative instances (K).

		Alibaba		Yelp2018		Amazon	
		Recall	NDCG	Recall	NDCG	Recall	NDCG
LightGCN+MixGCF	K=1	0.0763	0.0357	0.0713	0.0589	0.0460	0.0216
	K=2	0.0763	0.0359	0.0716	0.0591	0.0460	0.0216
	K=4	0.0775	0.0368	0.0722	0.0595	0.0462	0.0217
	K=8	0.0794	0.0378	0.0723	0.0593	0.0464	0.0220
NGCF+MixGCF	K=1	0.0544	0.0262	0.0688	0.0566	0.0350	0.0154
	K=2	0.0582	0.0279	0.0681	0.0560	0.0345	0.0157
	K=4	0.0596	0.0288	0.0688	0.0564	0.0346	0.0156
	K=8	0.0624	0.0305	0.0683	0.0562	0.0345	0.0157
PinSage+MixGCF	K=1	0.0489	0.0226	0.0632	0.0525	0.0273	0.0124
	K=2	0.0529	0.0249	0.0631	0.0523	0.0302	0.0138
	K=4	0.0550	0.0256	0.0640	0.0532	0.0300	0.0137
	K=8	0.0560	0.0261	0.0651	0.0537	0.0300	0.0140

three datasets on three GNN-based recommendation models. It also has comparable complexity to other samplers, especially the MCNS.

4.3 Study of MixGCF

In this section, we first conduct an ablation study to investigate the effect of positive mixing. Towards the further analysis, we explore the influence of neighbor range, i.e., the number of aggregation modules. We then study how different size of candidate set and the distribution of the random coefficient in positive mixing affect the performance. In what follows, we extend the pairwise BPR loss to K-pair loss for analyzing the impact of the number of negative instances. For a fair comparison, we use the identical experimental settings as Section 4.2.

Impact of Presence of Positive Mixing. To verify the impact of positive mixing, we do an ablation study by disabling the positive mixing module (cf. Eq. (5)) of MixGCF, termed MixGCF$_{\text{w/o p-m}}$. The experimental results are shown in Table 3 and we further plot the training curves of LightGCN, NGCF and PinSage in Figure 3. The observations are as followed:

- Aligning with Table 2 in Section 4.2, removing the positive mixing module degrades the performance of recommenders in most cases, indicating the necessity of positive mixing.
- Without positive mixing, MixGCF still outperforms DNS in most cases, manifesting that the hop-wise sampling is more effective than instance-wise sampling in GNN-based recommendation.
- Comparing the training curves of MixGCF and MixGCF$_{\text{w/o p-m}}$, it can be founded that the performance of MixGCF$_{\text{w/o p-m}}$ soars in the early phase but quickly reaches the peak and plumbs while MixGCF benefits from positive mixing and becomes more robust against the over-fitting.

Impact of Neighbor Range. To analyze the impact of the neighbor range, we vary L in the range of $\{1, 2, 3\}$ and summarize the results in Table 4. We observe that:

- Increasing the neighbor range can improve the performance of recommenders in most cases. It is reasonable since considering the larger range of neighbors, more negative items can be integrated into the synthesized negative.
- It can be found that MixGCF-1 achieves comparable or even higher performance to the MixGCF-2 and MixGCF-3 on PinSage. The possible reason is that the heavy design makes PinSage suffer from the risk of over-smoothing.

Impact of Candidate Set. We also conduct an experiment to analyze the influence of the size of the candidate set M. We search M in range of $\{8, 16, 32, 64\}$ and sum up the results in Table 5 where the best results is highlighted (starred). The observations are as followed:

- Increasing the candidate set size enhances the accuracy of recommenders in most cases. For example, NGCF obtains the best performance on three datasets with $M = 64$.
- Interestingly, the improvements on Amazon is not significant as the other datasets when M is increased. We attribute it to the different scale and distribution of these datasets.

Impact of Distribution of Positive Mixing. Since the random coefficient $\alpha^{(l)}$ is the core of positive mixing, we conduct a experiment to investigate the impact of its distribution on LightGCN. We explore different choices according to the relevant researches [18, 51]:

- **Beta distribution**. *mixup* [51] samples a mixing coefficient from Beta distribution Beta(β, β) to interpolates two samples. Following its experimental settings, we vary β in range $\{0.2, 0.4\}$.
- **Gaussian distribution**. Gaussian distribution Gaus(μ, σ) is one of the most prevalent distributions in machine learning. Here we fix $\mu = 0.5, \sigma = 0.1$.
- **Uniform distribution**. MoCHi [18] serves for self-supervised learning approaches, and proposes creating a convex linear combination of negative feature and query feature [12] for alleviating the false negative issue. It randomly samples a mixing coefficient

from uniform distribution Uni$(0, a)$. Following its setting and our positive mixing, we fix $a = 0.5, 1$ to study.

The experimental results are illustrated in Figure 4. We have the following observations:

- Our original setting in positive mixing, i.e., Uni$(0, 1)$, yields the best performance in all the cases. Comparing with Uni$(0, 0.5)$, the possible reason why Uni$(0, 1)$ achieves a higher performance is that limiting the selecting range of random coefficient will reduce the search space of the model's parameters.
- The beta distribution does not perform well on three datasets. We attribute it to the requirement of a sophisticated selection of the hyper-parameter β, which is known as a limitation of *mixup*.

Based on the above discussions, we find that sampling $\alpha^{(l)}$ from uniform distribution leads to a good performance. Thus, to avoid complicating MixGCF, we fix the sampling distribution as Uni$(0, 1)$.

Impact of Number of Negatives. We further conduct an experiment to investigate the influence of the number of negative instances K. Most of the prior work on negative sampling [28, 40, 44, 52] apply pairwise BPR loss to optimize the model. Inspired by the efforts on metric learning [32, 34], we generalize the pairwise BPR loss to K-pair loss and study its effect on MixGCF. We vary K in range of $\{1, 2, 4, 8\}$. Note that when $K = 1$, the K-pair loss will degenerate into the BPR loss. The detailed description of K-pair loss is in Appendix A.2.

We summarize the results in Table 6, and observe that increasing the K enhances the performance of the recommender consistently in most cases, verifying the effectiveness of the combination of MixGCF and K-pair loss. Such a phenomenon is expected since that the model distinguishes multiple synthesized negatives at the same time can lead to a more discriminative representation of the data and faster convergence.

5 RELATED WORK

5.1 GNN-Based Recommendation

Recommendation system has been applied to many online services, such as E-commerce and social media [5, 14, 23, 24]. To provide personalized information accurately, the recommendation system should estimate the preference of users towards myriad items. The most common paradigm is reconstructing the historical interactions by parameterizing the users and items.

In the recent years, GNN-based recommendation has gained considerable attention since the most information can be organized into a graph structure. To be specific, many recent works [3, 13, 35, 37, 43, 49] consider the interactions between users and items as a bipartite graph, and utilize the graph neural network (GNN) to refine the representations of each node. For example, PinSage [49] designs a random-walk based sampling method to sample neighborhoods and the visit counts are taken as the importance coefficients during aggregation. GC-MC [37] models each node only with its adjacent neighbors by the aggregation module for the rating prediction task. NGCF [43] exploits the multi-hop community information by recursively propagating the message and combines the representations of different layers to obtain a better performance. LightGCN [13] argues the burdensome design of NGCF and simplifies the aggregation module by removing the nonlinear feature transformation.

To alleviate the cold-start issue and enhance the performance, some previous works take the side information, e.g., social network [7, 29, 46] and knowledge graph [2, 38, 39, 42, 45], into account. For instance, GraphRec [7] integrates the users into a social network and refines the representations of each node with a graph attention network. KGAT [42] integrates the items into a knowledge graph and considers the interaction between users and items as a relation. It adopts the graph attention mechanism to exploit the high-order connections.

5.2 Negative Sampling for Recommendation

Negative sampling is first proposed to accelerate the training of skip-gram [26]. In recent years, negative sampling has been studied in recommendation task for solving one-class problem [4, 27]. To be specific, most interactions between users and items are in the form of implicit feedback, which only consists of positive feedback. Existing negative sampling methods roughly fall into four groups.

Static Sampler [26, 31] samples the negative item from a fixed distribution. Bayesian Personalized Ranking (BPR) [31] applies uniform distribution to sample negative, which is one of the most prevalent methods. In addition, the negative sampling distribution of some previous work on network embedding [10, 36] is proportional to the 3/4 power of the positive distribution.

GAN-based Sampler [1, 28, 40, 41] is an adversarial sampler based on generative adversarial network (GAN). For example, IRGAN [40] trains a generative adversarial network to play a min-max game with the recommender. The sampler performs as a generator and samples the negative to confuse the recommender. KBGAN [1] applies the framework to knowledge graph embedding task by training two translation-based models. AdvIR [28] adds the perturbation in adversarial sampling to make the model more robust.

Hard Negative Sampler [6, 17, 30, 52] adaptively picks the hardest negative by the current recommender. As a representative hard negative sampler, DNS [52] selects the negative scored highest by the recommender. The main difference between MixGCF and DNS is the sampling level. To be specific, DNS is an instance-wise sampling method, which aims to sample a representation for one certain item. MixGCF is developed to serve the GNN-based recommender and is a hop-wise sampling method which samples the representations of each hop among the negative. Besides, MixGCF applies the positive mixing to improve the quality of negative candidates (Section 3.1).

Graph-based Sampler [44, 48, 49] samples the negative based on the graph information. For example, MCNS [48] derives a theory to quantify the impact of the negative sampling distribution, and based on the theory, it approximates the positive distribution with self-contrast approximation. KGPolicy [44] incorporates knowledge graph into negative sampling and develop a reinforcement learning agent to sample high-quality negative items. PinSage [49] samples the node according to their Personalized PageRank scores.

6 CONCLUSION

In this work, we study the GNN-based recommendations with the goal of improving the quality of negative samples. We devise a simple and non-parametric method: MixGCF, which is a general

technique for the GNN-based recommendation models. Instead of sampling existing negative items, MixGCF integrates multiple negatives to synthesize a hard negative by positive mixing and hop mixing. We conduct experiments on public datasets and the results suggest that MixGCF can empower state-of-the-art GNN-based recommenders to achieve significant performance improvements over their original versions.

Capturing (negative) signals from data implicitly plays a crucial role in various fundamental learning tasks, such as contrastive learning and self-supervised learning. One promising direction is to bridge the developments from different topics for deriving general insights and inspiring perspectives. In the future, we would like to explore and further the techniques proposed in MixGCF for graph and relational data pre-training, so as to learn strongly generalizable representations of data.

Acknowledgements. We thank Fanjin Zhang and Da Yin for their helpful comments. This work is funded by National Natural Science Foundation of China (Key Program, No. 61836013), National Science Foundation for Distinguished Young Scholars (No. 61825602).

REFERENCES

[1] Liwei Cai and William Yang Wangc. 2018. KBGAN: Adversarial Learning for Knowledge Graph Embeddings. In *NAACL-HLT'18*. 1470–1480.
[2] Yixin Cao, Xiang Wang, Xiangnan He, Zikun Hu, and Tat-Seng Chua. 2019. Unifying Knowledge Graph Learning and Recommendation: Towards a Better Understanding of User Preferences. In *WWW*.
[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 *AAAI*.
[4] Ting Chen, Yizhou Sun, Yue Shi, and Liangjie Hong. 2017. On Sampling Strategies for Neural Network-based Collaborative Filtering. In *KDD*. 767–776.
[5] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep Neural Networks for YouTube Recommendations. In *RecSys*. 191–198.
[6] Jingtao Ding, Yuhan Quan, Quanming Yao, Yong Li, and Depeng Jin. 2020. Simplify and Robustify Negative Sampling for Implicit Collaborative Filtering. In *NeurIPS*.
[7] Wenqi Fan, Yao Ma, Qing Li, Yuan He, Eric Zhao, Jiliang Tang, and Dawei Yin. 2019. Graph neural networks for social recommendation. In *WWW*. 417–426.
[8] Wenzheng Feng, Jie Zhang, Yuxiao Dong, Yu Han, Huanbo Luan, Qian Xu, Qiang Yang, Evgeny Kharlamov, and Jie Tang. 2020. Graph Random Neural Network for Semi-Supervised Learning on Graphs. In *NeurIPS*.
[9] Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*. 249–256.
[10] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *KDD*. 855–864.
[11] William L. Hamilton, Rex Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *NeurIPS*. 1025–1035.
[12] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. 2020. Momentum contrast for unsupervised visual representation learning. In *CVPR*.
[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 *SIGIR*.
[14] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural Collaborative Filtering. In *WWW*. 173–182.
[15] Xinran He, Junfeng Pan, Ou Jin, Tianbing Xu, Bo Liu, Tao Xu, Yanxin Shi, Antoine Atallah, Ralf Herbrich, Stuart Bowers, and Joaquin Quiñonero Candela. 2014. Practical Lessons from Predicting Clicks on Ads at Facebook. In *ADKDD*.
[16] Cheng-Kang Hsieh, Longqi Yang, Yin Cui, Tsung-Yi Lin, Serge Belongie, and Deborah Estrin. 2017. Collaborative Metric Learning. In *WWW*.
[17] Jui-Ting Huang, Ashish Sharma, Shuying Sun, Li Xia, David Zhang, Philip Pronin, Janani Padmanabhan, Giuseppe Ottaviano, and Linjun Yang. 2020. Embedding-based Retrieval in Facebook Search. In *KDD*.
[18] Yannis Kalantidis, Mert Bulent Sariyildiz, Noe Pion, Philippe Weinzaepfel, and Diane Larlus. 2020. Hard Negative Mixing for Contrastive Learning. In *NeurIPS*.
[19] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *ICLR*.
[20] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR*.
[21] Yehuda Koren, Robert M. Bell, and Chris Volinsky. 2009. Matrix Factorization Techniques for Recommender Systems. In *IEEE Computer*. 42, 8 (2009), 30–37.
[22] Walid Krichene and Steffen Rendle. 2020. On Sampled Metrics for Item Recommendation. In *KDD*. 1748–1757.
[23] Hao Ma, Haixuan Yang, Michael R Lyu, and Irwin King. 2008. Sorec: social recommendation using probabilistic matrix factorization. In *CIKM*. 931–940.
[24] Hao Ma, Dengyong Zhou, Chao Liu, Michael R Lyu, and Irwin King. 2011. Recommender systems with social regularization. In *WSDM*. 287–296.
[25] Julian McAuley, Christopher Targett, Qinfeng Shi, and Anton Van DenHengel. 2015. Image-based recommendations on styles and substitutes. In *SIGIR*.
[26] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *NeurIPS*. 3111–3119.
[27] Rong Pan, Yunhong Zhou, Bin Cao, Nathan N Liu, Rajan Lukose, Martin Scholz, and Qiang Yang. 2008. One-class collaborative filtering. In *ICDM*. 502–511.
[28] Dae Hoon Park and Yi Chang. 2019. Adversarial Sampling and Training for Semi-Supervised Information Retrieval. In *WWW*. 1443–1453.
[29] Jiezhong Qiu, Jian Tang, Hao Ma, Yuxiao Dong, Kuansan Wang, and Jie Tang. 2018. DeepInf: Social Influence Prediction with Deep Learning. In *KDD*. 2110–2119.
[30] Steffen Rendle and Christoph Freudenthaler. 2014. Improving pairwise learning for item recommendation from implicit feedback. In *WSDM*. 273–282.
[31] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian Personalized Ranking from Implicit Feedback. In *UAI*. 452–461.
[32] Florian Schroff, Dmitry Kalenichenko, and James Philbin. 2015. FaceNet: A Unified Embedding for Face Recognition and Clustering. In *CVPR*.
[33] Patrice Y. Simard, Yann A. Le Cun, John S. Denker, and Bernard Victorri. 1998. Transformation invariance in pattern recognitiontangent distance and tangent propagation. In *Neural networks: tricks of the trade*.
[34] Kihyuk Sohn. 2016. Improved Deep Metric Learning with Multi-class N-pair Loss Objective. In *NeurIPS*.
[35] Qiaoyu Tan, Ninghao Liu, Xing Zhao, Hongxia Yang, Jingren Zhou, and Xia Hu. 2020. Learning to Hash with Graph Neural Networks for Recommender Systems. In *WWW*. 1988–1998.
[36] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, JunYan, and Qiaozhu Mei. 2015. Line: Large-scale information network embedding. In *WWW*. 1067–1077.
[37] Rianne van den Berg, Thomas N Kipf, and Max Welling. 2018. Graph Convolutional Matrix Completion. In *KDD*.
[38] Hongwei Wang, Fuzheng Zhang, Jialin Wang, Miao Zhao, Wenjie Li, Xing Xie, and Minyi Guo. 2018. RippleNet: Propagating User Preferences on the Knowledge Graph for Recommender Systems. In *CIKM*. 417–426.
[39] Hongwei Wang, Fuzheng Zhang, Mengdi Zhang, Jure Leskovec, Miao Zhao, Wenjie Li, and Zhongyuan Wang. 2019. Knowledge-aware Graph Neural Networks with Label Smoothness Regularization for Recommender Systems. In *KDD*.
[40] Jun Wang, Lantao Yu, Weinan Zhang, Yu Gong, Yinghui Xu, Benyou Wang, Peng Zhang, and Dell Zhang. 2017. IRGAN: A Minimax Game for Unifying Generative and Discriminative Information Retrieval Models. In *SIGIR*. 515–524.
[41] Qinyong Wang, Hongzhi Yin, Zhiting Hu, Defu Lian, Hao Wang, and Zi Huang. 2018. Neural Memory Streaming Recommender Networks with Adversarial Training. In *KDD*. 2467–2475.
[42] Xiang Wang, Xiangnan He, Yixin Cao, Meng Liu, and Tat-Seng Chua. 2019. KGAT:Knowledge Graph Attention Network for Recommendation. In *KDD*. 950–958.
[43] Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. 2019. Neural Graph Collaborative Filtering. In *SIGIR*.
[44] Xiang Wang, Yaokun Xu, Xiangnan He, Yixin Cao, Meng Wang, and Tat-Seng Chua. 2020. Reinforced Negative Sampling over Knowledge Graph for Recommendation. In *WWW*.
[45] Ze Wang, Guangyan Lin, Huobin Tan, Qinghong Chen, and Xiyang Liu. 2020. CKAN: Collaborative Knowledge-aware Attentive Network for Recommender Systems. In *SIGIR*. 219–228.
[46] Le Wu, Peijie Sun, Yanjie Fu, Richang Hong, Xiting Wang, and Meng Wang. 2019. A Neural Influence Diffusion Model for Social Recommendation. In *SIGIR*.
[47] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken ichi Kawarabayashi, and Stefanie Jegelka. 2018. Representation Learning on Graphs with Jumping Knowledge Networks. In *ICML*.
[48] Zhen Yang, Ming Ding, Chang Zhou, Hongxia Yang, Jingren Zhou, and Jie Tang. 2020. Understanding Negative Sampling in Graph Representation Learning. In *KDD*.
[49] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. 2018. Graph convolutional neural networks for web-scale recommender systems. In *KDD*. 974–983.
[50] Hao Yu, Huanyu Wang, and Jianxin Wu. 2021. Mixup Without Hesitation. *arXiv preprint arXiv:2101.04342* (2021).
[51] Hongyi Zhang, Moustapha Cisse, Yann N. Dauphin, and David Lopez-Paz. 2018. mixup: Beyond empirical risk minimization. In *ICLR*.
[52] Weinan Zhang, Tianqi Chen, Jun Wang, and Yong Yu. 2013. Optimizing top-n collaborative filtering via dynamic negative item sampling. In *SIGIR*. 785–788.
[53] Guorui Zhou, Xiaoqiang Zhu, Chenru Song, Ying Fan, Han Zhu, Xiao Ma, Yanghui Yan, Junqi Jin, Han Li, and Kun Gai. 2018. Deep interest network for click-through rate prediction. In *KDD*. 1059–1068.

A APPENDIX

In the appendix, we present the detailed implementation note of MixGCF, including the running environment and the parameter settings. We then formally introduce K-pair loss. Finally, we describe the datasets, i.e., Alibaba, Yelp2018 and Amazon, and evaluation metrics. All codes used in this paper are publicly available.

A.1 Implementation Note

Running Environment. The experiments are conducted on a single Linux server with Intel(R) Xeon(R) CPU Gold 6420 @ 2.60GHz, 376G RAM and 10 NVIDIA GeForce RTX 3090TI-24GB. Our method is implemented on PyTorch 1.7.0 and Python 3.7.6.

Hyperparameter settings. The following Table lists the parameter settings of each recommender and MixGCF on three datasets. The hyperparameters include the learning rate lr, the embedding size d, the number of aggregation modules L, the batch size B, the coefficient of $L2$ regularization l_2, and the size of the candidate set M.

		lr	l_2	B	d	L	M
LightGCN	Alibaba	10^{-3}	10^{-4}	2048	64	3	32
	Yelp2018	10^{-3}	10^{-4}	2048	64	3	64
	Amazon	10^{-3}	10^{-4}	2048	64	3	16
NGCF	Alibaba	10^{-3}	10^{-4}	1024	64	3	64
	Yelp2018	10^{-4}	10^{-4}	1024	64	3	64
	Amazon	10^{-4}	10^{-4}	1024	64	3	64
PinSage	Alibaba	10^{-3}	10^{-4}	2048	64	3	64
	Yelp2018	10^{-3}	10^{-4}	2048	64	3	16
	Amazon	10^{-3}	10^{-4}	2048	64	3	64

Implementation Details. Our implementation can be split into four parts: data iteration, embedding generation, negative sampling, and model learning. For each epoch, we randomly shuffle the training data and select the positive pairs with a size of B orderly until the end of the epoch. Then for each data batch, the current recommender generates the aggregated embeddings of users and items based on PyTorch 1.7.0 APIs. In this paper, we focus on the negative sampling part. The negative sampler picks/synthesizes a negative instance for each positive pair. After obtaining the negative items, we feed the representations of target users, positive items, and negative items into the loss function and update the parameters of the model based on the gradient.

A.2 K-Pair Loss

Inspired by the prior researches on metric learning [16, 34], we extend the pairwise BPR loss [31] to K-pair loss aiming to enhance the performance of the model. Formally, the pairwise BPR loss is as follow:

$$\mathcal{L}_{\text{BPR}} = \sum_{(u,v^+,v^-)\in O} -\ln \sigma(y_{u,v^+} - y_{u,v^-})$$

where $y_{u,v}$ denotes the matching score between user u and item v, and $O = \{(u,v^+,v^-)|(u,v^+) \in O^+, (u,v^-) \notin O^+\}$ is the training dataset comprising of the interactions O^+ and the negative samples. Equivalently, we have

$$\mathcal{L}_{\text{BPR}} = \sum_{(u,v^+,v^-)\in O} -\ln \frac{\exp(y_{u,v^+})}{\exp(y_{u,v^+}) + \exp(y_{u,v^-})}$$

The K-pair loss proposed in [34] is formulated as:

$$\mathcal{L}_{K-\text{pair}} = \sum_{(u,v^+,v_0^-,\cdots,v_K^-)\in O} -\ln \frac{\exp(y_{u,v^+})}{\exp(y_{u,v^+}) + \sum_{i=0}^{K}\exp(y_{u,v_i^-})}$$

where $\{v_0^-,\cdots,v_K^-\}$ denotes the set of K sampled negatives for each interaction pair.

A.3 Datasets

We use three publicly available datasets which vary in terms of domain, size, and sparsity to evaluate our proposed MixGCF.

- Alibaba [48] is collected from the Alibaba online shopping platform. The authors of MCNS organize the purchase record of selected users to construct the bipartite user-item graph.
- Yelp2018 [43] is from the 2018 edition of the Yelp challenge. Each interaction of the graph is represented a restaurant or bar consumption record of a user. Besides, the authors applied the 10-core setting to filter out the nodes with less than 10 interactions.
- Amazon [48] is first released in [25] which contains several datasets of different products. The authors selected the "electronics" category to form a bipartite user-item graph whose time stamp spans from May 1996 to July 2014.

A.4 Evaluation Metrics

In this paper, we conduct the all-ranking protocol and adopt two evaluation metrics: Recall@N and NDCG@N. The detailed introduction is shown as follow:

- **Recall@N** is the proportion of relevant items found in the top-k recommendation. To be specific, we compute Recall@N as follow:
  (1) For each user u in testing set, we compute the matching score with the rest of items $\{v_0,\cdots,v_S\}$ which are never shown in the training interactions of u.
  (2) Sort the list of $L = \{\mathbf{e}_u^T\mathbf{e}_{v_0},\cdots,\mathbf{e}_u^T\mathbf{e}_{v_S}\}$ in descending order.
  (3) Count the number of relevant items $|\mathcal{I}_{u,N}|$ occurring in the top N terms of L.
  (4) Calculate Recall@$N = \frac{1}{|\mathcal{U}|}\sum_{u\in\mathcal{U}}\frac{|\mathcal{I}_{u,N}|}{|\mathcal{I}_u|}$, where $\mathcal{U}$ is the testing set of users and $\mathcal{I}_u$ is the set of testing items for u.
- **NDCG@N** measures the quality of ranking list at a finer granularity than Recall@N. It assigns different weight to different ranks of items. We compute NDCG@N as follow:
  (1) For each user u in testing set, we compute the matching score with the rest of items $\{v_0,\cdots,v_S\}$ which are never shown in the training interactions of u.
  (2) Sort the list of $L = \{\mathbf{e}_u^T\mathbf{e}_{v_0},\cdots,\mathbf{e}_u^T\mathbf{e}_{v_S}\}$ in descending order.
  (3) Calculate the discounted cumulative gain:
  $$\text{DCG@}N = \frac{1}{|\mathcal{U}|}\sum_{u\in\mathcal{U}}\sum_{n=1}^{N}\frac{\delta(L_n\in\mathcal{I}_u)}{\log_2(n+1)}$$
  where L_n denotes the n-th term of L, and $\delta(\cdot)$ is the indicator function.
  (4) Calculate NDCG@N = DCG@N/IDCG@N, where IDCG@N denotes the ideal discounted cumulative gain.