Batch-Mix Negative Sampling for Learning Recommendation Retrievers

Yongfu Fan
University of Electronic Science and Technology of China
van.lqz72@gmail.com

Jin Chen
University of Electronic Science and Technology of China
chenjin@std.uestc.edu.cn

Yongquan Jiang
Southwest Jiaotong University
yqjiang@swjtu.edu.cn

Defu Lian
University of Science and Technology of China
liandefu@ustc.edu.cn

Fangda Guo
Institute of Computing Technology, Chinese Academy of Sciences
guofangda@ict.ac.cn

Kai Zheng*
University of Electronic Science and Technology of China
zhengkai@uestc.edu.cn

ABSTRACT

Recommendation retrievers commonly retrieve user potentially preferred items from numerous items, where the query and item representation are learned according to the dual encoders with the log-softmax loss. Under real scenarios, the number of items becomes considerably large, making it exceedingly difficult to calculate the partition function with the whole item corpus. Negative sampling, which samples a subset from the item corpus, is widely used to accelerate the model training. Among different samplers, the in-batch sampling is commonly adopted for online recommendation retrievers, which regards the other items within the mini-batch as the negative samples for the given query, owing to its time and memory efficiency. However, the sample selection bias occurs due to the skewed feedback, harming the retrieval quality. In this paper, we propose a negative sampling approach named **B**atch-**M**ix **N**egative **S**ampling (**BMNS**), which adopts batch mixing operation to generate additional negatives for model training. Concretely, BMNS first generates new negative items with the sampled mix coefficient from the Beta distribution, after which a tailored correct strategy guided by frequency is designed to match the sampled softmax loss. In this way, the effort of re-encoding items out of the mini-batch is reduced while also improving the representation space of the negative set. The empirical experiments on four real-world datasets demonstrate BMNS is superior to the competitive negative inbatch sampling method.

CCS CONCEPTS

• **Information systems → Recommender systems**.

KEYWORDS

Information Retrieval; Recommender Systems; Negative Sampling

ACM Reference Format:
Yongfu Fan, Jin Chen, Yongquan Jiang, Defu Lian, Fangda Guo, and Kai Zheng. 2023. Batch-Mix Negative Sampling for Learning Recommendation Retrievers. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management (CIKM '23), October 21–25, 2023, Birmingham, United Kingdom.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3583780.3614789

1 INTRODUCTION

With the rapid development of the Internet, the problem of information overload has become increasingly severe. The recommender system, which typically infers users' preferences based on their historical behavior, has become one of the crucial solutions to alleviate the problem of information overload. As a key stage of large-scale recommenders [8, 9, 11], the recommender retrievers attempt to mine highly relevant items from the whole corpus. Recently, the popular retriever framework often trains dual-encoders (also called two-towers) [9, 26, 27], with each encoder dedicated to learning the representation of users and items, respectively. During the training, for each given user-item pair, the loss function, i.e., log-softmax [9], encourages the model to assign a higher score to the positive samples against other items. However, in large-systems, the scale of the items often reaches millions or even more, resulting in unbearable computational expenses for the partition function calculation.

Negative sampling [7, 18, 25–27] is an extensively used and effective technique for reducing the computational load during model training. It involves sampling a subset of samples from the entire corpus to approximate the gradient over all items. Considering the real scenarios with a huge number of items and deep networks, sampling from the entire corpus requires additional memory footprint to encode the unseen items out of the mini-batch through the large-scale network, which may not meet the latency and memory requirements. The in-batch sampling [26, 27], which directly considers other samples within the mini-batch as negatives, has becomes popular and has been successfully deployed in many recommender systems owing to its memory-efficient computation.

Unfortunately, the items in the mini-batches are more frequently exposed and follow the long-tail distribution, which leads to in-batch sampling excessively punishing the popular items and resulting in poor gradient approximating, usually referred to the sample

*Corresponding author.
Kai Zheng is with Yangtze Delta Region Institute (Quzhou), School of Computer Science and Engineering, and Shenzhen Institute for Advanced Study, University of Electronic Science and Technology of China.

CIKM '23, October 21–25, 2023, Birmingham, United Kingdom

ACM ISBN 979-8-4007-0124-5/23/10…$15.00
https://doi.org/10.1145/3583780.3614789

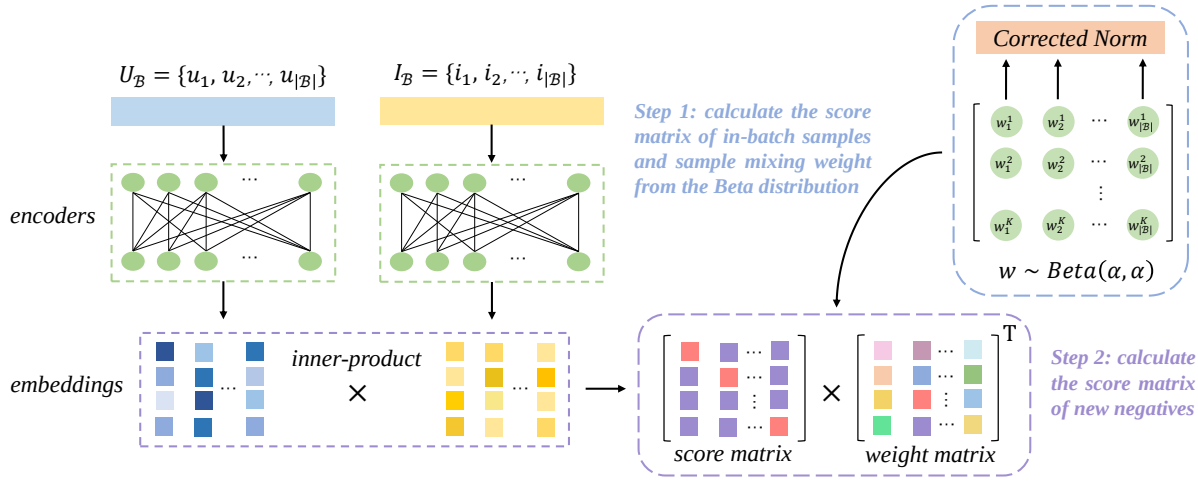


Figure 1: The illustration of Batch-Mix Negative Sampling

selection bias. Several works [26, 27] correct the sampling bias with the exposure frequency, which is actually consistent with the sampled softmax [2] with the popularity-based sampling distribution. To further alleviate the long-tail effect, MNS [26] mixes up the in-batch items with the additionally uniformly sampled items to enrich the representation space. XIR [7] caches the informative items in preceding iterations as the augmented negatives for current training. However, despite the better performance of MNS and XIR, they both require re-encoding of sampled items, which becomes incredibly inefficient with numerous features and large-scale encoders. On the one hand, loading items with their rich features among disks, CPU memory and GPU memory takes much time, which usually occupies the majority of running time. On the other hand, extensive computations for encoding additionally sampled items out of the mini-batch incurs unneglected computational expense.

To expand the negative set of in-batch sampling without introducing additional data loading and re-encoding by training two-tower models, we propose the **B**atch-**M**ix **N**egative **S**ampling, shortly as BMNS, which generates the new negative samples according to the mini-batch items. Inspired by Mixup [30], which augments the new data points by linearly mixing up two data points according to the Beta distribution, we generate the virtual negatives in a better representation space, thereby improving the performance and generalization of the model. Different from Mixup, which only exploits two data points to generate a new point, we mix up virtual negative items with multiple items within the mini-batch. Concretely, the weights according to the Beta distribution are assigned to each item, which are later normalized through the softmax function over items for each user query. The normalized weights are then utilized to mix up the virtual negatives, without the need of sampling additional items from the whole corpus. Moreover, in order to be consistent with the native sampled softmax loss with in-batch sampling, the frequency probabilities are fit into the loss function, where we conduct a theoretical analysis on the gradient. In this way, BMNS achieves the goal of alleviating the selection bias of in-batch sampling without additionally sampling items out of the mini-batch, which is both time and cost efficient.

In this paper, our contributions can be summarized as follows:

- We propose a new negative sampling method, BMNS, which can generate virtual negatives using in-batch negatives for a better representation space, without additionally sampling items out of the mini-batch.
- We integrate the popularity into the mixing coefficient for adapting to the sampled softmax loss, with the aim of reducing the sample selection bias, after which a loss function containing in-batch samples and generated samples is utilized for optimization.
- We evaluate the proposed approach on four real-world datasets. The results indicate that our method can significantly improve recommendation performance compared to the baseline while keeping high efficiency.

2 RELATED WORK

2.1 Negative Sampling in Recsys

Negative sampling technique is widely used to accelerate the training process for implicit feedback in recommender systems, especially under the pair-wise loss functions, e.g., BPR loss [20], Margin-based loss [13] and Sampled Softmax loss [5]. According to the types of sampling distributions, the samplers can be divided into two groups: the static samplers and the adaptive samplers. The static samplers usually select items according to the static distributions, e.g., the uniform distribution [20] and the popularity-based distribution [19]. Despite of their simplicity and efficiency, they are prone to overfitting to easy samples in the later training process [17]. The adaptive samplers, whose sampling distribution changes along with the model updates, select those items with higher similarity scores as negatives [10, 32] and thus achieve faster convergence and better performance. With further research, the community has reached the agreement that the softmax distribution is the optimal sampling distribution [22]. Existing efforts [5, 17, 22] have been devoted to alleviating the bias of the sampling from the softmax distribution with keeping low cost.

However, these approaches sample items from the entire item corpus. Considering the scenarios of online recommender retrievers, the user-item pairs with their numerous features are grouped into mini-batches inputted into the models. Thus encoding the

items outside of the mini-batch requires additional calculations, which may exceed the storage and latency limitations with limited hardware resources. In-batch sampling, which treats other items within the batch as negative samples, has become a general strategy for online recommender retrievers. To better approximate the softmax distribution with limited in-batch items, correct-sfx [27] corrects the sampling bias based on the streaming popularity estimation. MNS [26] further increases the sampler number by mixing the globally uniformly sampled items. XIR [7] caches the historical informative items for faster converge.

2.2 MixUp Strategy

Mixup [30] is a powerful data augmentation method for contrastive learning and has gained widespread adoption in various fields. The vanilla Mixup [30] focuses the vicinal risk minimization [3] and performs linear interpolation on two raw data points based on the Beta distribution to generate the new data point. Subsequently, several approaches work for data samples, such as images [16, 21, 29] and sentences [12, 31], are proposed to generate the augmented samples. Cutmix [29] proposes to combine Cutout and Mixup, mixing image information by exchanging some continuous regions in two images. SeqMix [31] introduces Mixup into NLP tasks by mixing sentence pairs based on score functions. The other type of Mixup strategy performs the mixup operation on hidden spaces, such as Manifold Mixup [24], being popular owing to the better representation learning. Furthermore, BatchMix [28] interpolates hidden states of the entire mini-batch to improve training. In recommender systems, Mixup technique is introduced in MixGCF [14] for graph-based learning through positive mixing and hop mixing to generate hard negatives for each layer. MixKG [4] also adopts the mixup for sampling harder negatives in knowledge graph tasks.

3 METHODOLOGY

This section first introduces the preliminaries of the problem, and then describes the two important components of BMNS, i.e., the mixup based negative sampling strategy and the tailored popularity guided corrected strategy, along with the modified loss function adapted for this method, where we conduct the gradient analysis. We also analyze the time complexity and space complexity of the proposed method. Figure 1 illustrates the workflow of BMNS.

3.1 Preliminaries

Given a query, the task of retriever models is retrieving the most relevant items over numerous item corpus. In a retriever system comprising M queries and N items, the sets of queries and items are represented by $\{\boldsymbol{q}_i \in \mathbb{R}^{d_Q}\}_{i=1}^{M}$ and $\{\boldsymbol{z}_i \in \mathbb{R}^{d_I}\}_{i=1}^{N}$, respectively. Here $\boldsymbol{q}_i$ and $\boldsymbol{z}_i$ are both vector of continuous or categorical features, which can be mapped into dense embedding space with two trainable embedding functions $\phi_Q : \mathbb{R}^{d_Q} \to \mathbb{R}^{d_D}$, $\phi_I : \mathbb{R}^{d_I} \to \mathbb{R}^{d_D}$. Under the two-tower architecture, the similarity of specific query u and item i can be calculated by the inner-product score function: $s(u,i) = \langle \phi_Q(\boldsymbol{q}_u;\theta), \phi_I(\boldsymbol{z}_i;\theta) \rangle$, where θ denotes the model parameters. The retrieval task is usually treated as a classification problem by optimizing the log-softmax loss function. Specifically, given a user query u, the likelihood of the user preferences with respect to the item i is represented as:

$$P(i|u) = \frac{\exp s(u,i)}{\sum_{j \in \mathcal{I}} \exp s(u,j)}$$

where $\mathcal{I}$ denotes the item set. The objective is to maximize the likelihood of each user-item pair (u,i) in the train data $\mathcal{D}$. Therefore, by the maximum likelihood estimation, the retrieval model is trained with log-softmax loss:

$$\begin{aligned}\mathcal{L}(\mathcal{D},\Theta) &= -\frac{1}{|\mathcal{D}|}\sum_{(u,i)\in\mathcal{D}} \log P(i|u) \\ &= -\frac{1}{|\mathcal{D}|}\sum_{(u,i)\in\mathcal{D}} \log \frac{\exp(s(u,i))}{\sum_{j\in\mathcal{I}}\exp(s(u,j))}\end{aligned}$$

where $|\cdot|$ denotes the element number of the set. In the large-scale reommender, the computation of partition function, i.e., the denominator of the softmax function, will become significantly complicated as the enlarging of $|\mathcal{I}|$. A popular strategy to alleviate this problem is in-batch sampling, i.e., treating other items within the batch as negatives. Compared with other global sampling methods, i.e., static sampling over the entire corpus, in-batch sampling is free from encoding out-batch items. Moreover, the sophisticated features and neural networks leveraged in online systems usually lead to significant time consumption for re-encoding items. Therefore, in-batch sampling obtains more extensive application in real recommendation scenarios.

However, due to the long-tail effect, popular items are more likely to appear within a batch. Therefore, the sampling distribution of in-batch sampling is biased from the softmax distribution since popular items are frequently selected as negatives. According to the sampled softmax [1, 2], the correction term $\log Q$ reduces sampling bias by lowering the penalty for popular items. Thus, the loss function with respect to the mini-batch $\mathcal{B}$ is following as:

$$\mathcal{L}(\mathcal{B},\Theta) = -\frac{1}{|\mathcal{B}|}\sum_{(u,i)\in\mathcal{B}} \log \frac{\exp(s^c(u,i))}{\sum_{j\in\mathcal{I}_\mathcal{B}}\exp(s^c(u,j))} \tag{1}$$

where $\mathcal{I}_\mathcal{B}$ denotes the item set of mini-batch $\mathcal{B}$. The corrected logit $s^c(u,i)$ is defined as:

$$s^c(u,j) = s(u,j) - \log pop(j),\ pop(j) = \frac{f_j}{\sum_{k\in\mathcal{I}} f_k} \tag{2}$$

where f_j denotes the occurrence numbers of item j in the training data. Here we do not consider the streaming data and the popularity is the accurate estimation of all training data.

3.2 Batch-Mix Negative Sampling

In-batch negative sampling directly samples negatives from the current batch and achieves better training efficiency. Increasing the batch size can provide more diverse negatives during the model training. However, the batch size is limited due to the constraint of computing resources, which engenders the bottleneck of in-batch sampling strategy. Although a series of methods [7, 26] based on in-batch sampling improve the sampling distribution by incorporating additional negative samples from the entire corpus or caching historical informative items, these methods inevitably introduce considerable computational overhead for re-encoding items. Due

to the complex features and deep neural networks, this expense is especially remarkable in large-scale recommendation systems.

Thus, we are motivated to design new sampling strategy to expand the sampling set without introducing additional re-encoding computation. Inspired by Mixup [30], which interpolates two samples as the new sample to augment the model training, we recognize that generating virtual negative samples may be a more efficient and effective way to sample negatives. Given two samples (x_i, y_i) and (x_j, y_j) drawn from training data, the standard Mixup synthesizes the new sample as follows:

$$\tilde{x} = \lambda x_i + (1-\lambda)x_j, \tilde{y} = \lambda y_i + (1-\lambda)y_j$$

where x and y denote input data and label, respectively. λ is the mixing coefficient sampled from Beta(α, α). The subsequent works [24, 28] further interpolate the representations in the hidden space instead of the original input data point for better representation space. This is different from that of negative sampling, where the items to be sampled are universally treated as negatives and thus the label mixup is not required. Based on this fact, we develop a negative sampling strategy which only depends on negative mixup.

Our method involves two important stages: sampling and mixing. In the first stage, for each mini-batch $\mathcal{B}$, we treat in-batch items as part of training negatives and encode them into embeddings. Meanwhile, we also sample mixing coefficient from the Beta distribution for each item. Then, in the mixing stage, BMNS employs hidden space interpolation to generate new negatives. Specifically, the embedding of new virtual negative k is generated as follows:

$$\boldsymbol{e}'_k = \sum_{j \in \mathcal{I}_\mathcal{B}} \hat{w}_j^{(k)} \phi_I(\boldsymbol{z}_j; \theta), \ \hat{w}_j^{(k)} = \frac{\exp\left(w_j^{(k)}\right)}{\sum_{l \in \mathcal{I}_\mathcal{B}} \exp\left(w_l^{(k)}\right)} \tag{3}$$

where $w_j^{(k)}$ is the unnormalized mixing coefficient sampled from Beta(α, α). $k \in \{1, 2, ..., K\}$ denotes the k-th sampling to generate the total K virtual negatives. Ideally, each pair of (query, item) would be assigned with a mixing coefficient to generate a new negative for each query, which takes $O(|\mathcal{B}| \times |\mathcal{B}|)$ complexity and finally takes $O(K \times |\mathcal{B}| \times |\mathcal{B}|)$ time complexity. However, considering the huge computational cost of sampling and mixing, we reuse the mixing coefficient for each item, thus only requiring to sample $K \cdot |\mathcal{B}|$ times from Beta(α, α) for the generation of K batch-shared negatives.

Futhermore, due to the large batch size applied in retriever model training, linear interpolation based on latent representation space incurs $O(|\mathcal{B}| \times d_D)$ time complexity for synthesizing a new negative. Especially when the embedding dimension is larger, the computational overhead of performing inner-product on numerous items will become more unacceptable. Fortunately, note that the mixup, i.e., Eq (3), is a linear operation, the score of generated negative k can be rewritten as:

$$\begin{aligned} s^{mix}(u,k) &= < \phi_Q(\boldsymbol{q}_u; \theta), \sum_{j \in \mathcal{I}_\mathcal{B}} \hat{w}_j^{(k)} \phi_I(\boldsymbol{z}_j; \theta) > \\ &= \sum_{j \in \mathcal{I}_\mathcal{B}} \hat{w}_j^{(k)} < \phi_Q(\boldsymbol{q}_u; \theta), \phi_I(\boldsymbol{z}_j; \theta) > \\ &= \sum_{j \in \mathcal{I}_\mathcal{B}} \hat{w}_j^{(k)} s(u,j) \end{aligned} \tag{4}$$

Naturally, by directly mixing the scores of negative samples, we eliminate the need for repetitive calculation of inner-product, thereby reducing the computational complexity of the mixing step from $O(|\mathcal{B}| \times d_D)$ to $O(|\mathcal{B}|)$.

3.3 Corrected Strategy for Sampled Softmax

Similar to the corrected logit in sampled softmax with in-batch sampling, we design the corrected strategy for the generated negatives according to BMNS, with the aim of alleviating the selection bias of in-batch sampling. Specifically, we correct the mixing coefficient based on the popularity of the corresponding item as follows:

$$\hat{w}_j^{(k)} = \frac{\exp\left(w_j^{(k)} - \log pop(j)\right)}{\sum_{l \in \mathcal{I}_\mathcal{B}} \exp\left(w_l^{(k)} - \log pop(l)\right)} \tag{5}$$

where $w_j^{(k)}$ denotes the mixing coefficient sampled according to the Beta distribution and $pop(\cdot)$ denotes the popularity of the item over the training data.

After negative sampling, we use both in-batch negatives and generated negatives for model training. To endow the adjustable capableness for loss function, we calculate the loss separately for two pieces of data and integrate them through a hyperparameter:

$$\begin{aligned} \mathcal{L}_{\text{BMNS}}(\mathcal{B}, \Theta) = &-\gamma \sum_{(u,i) \in \mathcal{B}} \log \frac{\exp\left(s^c(u,i)\right)}{\sum_{j \in \mathcal{I}_\mathcal{B}} \exp\left(s^c(u,j)\right)} \\ &- (1-\gamma) \sum_{(u,i) \in \mathcal{B}} \log \frac{\exp\left(s^c(u,i)\right)}{\sum_{j \in \mathcal{K}_\mathcal{B}} \exp\left(s^{mix}(u,j)\right)} \end{aligned} \tag{6}$$

where $s^c(u,i) = s(u,i) - \log pop(i)$ denotes the corrected logit and $s^{mix}(u,j)$ denotes the similarity score of the generated negative j. $\mathcal{K}_\mathcal{B}$ denotes the set of generated negatives. The hyperparameter $\gamma \in [0,1]$ controls the proportion of the loss function that corresponds to in-batch negatives and generated negatives. Note that when γ=1, the loss function degenerates to in-batch sampled softmax loss, i.e., Eq (1). In other words, the BMNS strategy can be interpreted as the generalization of in-batch sampling. The algorithm 1 describes the pseudocode of Batch-Mix Negative Sampling.

Compared to the vanilla in-batch sampling, BMNS generates more negatives according to the in-batch items, without introducing additional items outside of the mini-batch. We further analyze the gradient with respect to the $\mathcal{L}_{\text{BMNS}}$ to investigate the influence of the generated negative samples. To simplify the analysis, we consider the loss with respect to each user-item pair (u,i) within the mini-batch:

$$\begin{aligned} \mathcal{L}(u,i) = &-s^c(u,i) + \gamma \log \sum_{j \in \mathcal{I}_\mathcal{B}} \exp\left(s^c(u,j)\right) \\ &+ (1-\gamma) \log \sum_{j \in \mathcal{K}_\mathcal{B}} \exp\left(s^{mix}(u,j)\right) \end{aligned} \tag{7}$$

The gradient of Equation (7) *w.r.t* the parameters θ follows as:

$$\begin{aligned} \nabla_\theta \mathcal{L}(u,i) = &-\nabla_\theta s^c(u,i) + \gamma \sum_{j \in \mathcal{I}_\mathcal{B}} P(j|u) \nabla_\theta s^c(u,j) \\ &+ (1-\gamma) \sum_{k \in \mathcal{K}_\mathcal{B}} P^*(k|u) \sum_{j \in \mathcal{I}_\mathcal{B}} \hat{w}_j^{(k)} \nabla_\theta s(u,j) \end{aligned}$$

Algorithm 1: Batch-Mix Negative Sampling

Input: Train data set $\mathcal{D} = \{(u, i)\}$, Sampling distribution $\text{Beta}(\alpha, \alpha)$, Generated negative sample size K, Hyperparameter γ, The number of epochs T

Output: Model parameters θ

1 Calculate the item popularity $\mathcal{P} = \{pop(i) \mid i \in \mathcal{I}\}$ based on the training data $\mathcal{D}$;
2 **for** $t = 1, 2, \ldots, T$ **do**
3 **for** *mini-batch* $\mathcal{B} \in \mathcal{D}$ **do**
4 $\mathcal{U}_\mathcal{B} = \{u \mid (u, i) \in \mathcal{B}\}, \mathcal{I}_\mathcal{B} = \{i \mid (u, i) \in \mathcal{B}\}$;
5 Encode queries into embeddings $\boldsymbol{E}_{U_B}$ based on ϕ_Q;
6 Encode items into embeddings $\boldsymbol{E}_{I_B}$ based on ϕ_I;
7 Calculate the score matrix $\boldsymbol{S} = \boldsymbol{E}_{U_B} \boldsymbol{E}_{I_B}^\top$;
8 Sample mixing weight $\boldsymbol{W} \in \mathbb{R}^{K \times |\mathcal{B}|}$ from $\text{Beta}(\alpha, \alpha)$;
9 **for** $k = 1, 2, \ldots, K$ **do**
10 $\hat{w}_i^{(k)} = \frac{\exp\left(w_i^{(k)} - \log pop(i)\right)}{\sum_{j \in \mathcal{I}_\mathcal{B}} \exp\left(w_j^{(k)} - \log pop(j)\right)}, \forall i \in \mathcal{I}_\mathcal{B}$;
11 **end**
12 Calculate the mix score matrix $\boldsymbol{S}^{mix} = \boldsymbol{S} \cdot \hat{\boldsymbol{W}}^\top$;
13 Update θ based on the loss function Eq (6);
14 **end**
15 **end**

where $P(j|u) = \frac{\exp s^c(u,j)}{\sum_{l \in \mathcal{I}_\mathcal{B}} \exp s^c(u,l)}$ and $P^*(k|u) = \frac{\exp s^{mix}(u,k)}{\sum_{l \in \mathcal{K}_\mathcal{B}} \exp s^{mix}(u,l)}$. Taking expectation of the gradient, then we have:

$$\mathbb{E}\left[\nabla_\theta \mathcal{L}(u, i)\right] = -\nabla_\theta s\left(u, i\right) + \sum_{j \in \mathcal{I}_\mathcal{B}} \left(\gamma P\left(j|u\right) + (1 - \gamma) o_j\right) \nabla_\theta s(u, j)$$

where $o_j = \frac{1/pop(j)}{\sum_{l \in \mathcal{I}_\mathcal{B}} 1/pop(l)}$. For the sake of conciseness, we have included the details of the derivation in the appendix. Compared with the original in-batch sampling with sampled softmax, BMNS actually introduces an additional gradient term, where items with lower popularity tend to have higher weights. Assuming the gradient of logits $\nabla_\theta s(u, j)$ meet with $|\nabla_\theta s(u, j)| \leq C$, where C is a positive constant. Then the absolute bias between the expected loss gradient of the BMNS and that of in-batch sampling satisfies:

$$\begin{aligned}
\delta &= \left|\mathbb{E}\left[\nabla_\theta \mathcal{L}_{\text{BMNS}}(u, i)\right] - \mathbb{E}\left[\nabla_\theta \mathcal{L}_{\text{In-batch}}(u, i)\right]\right| \\
&= \left| \sum_{j \in \mathcal{I}_\mathcal{B}} (1 - \gamma) \left(-P\left(j|u\right) + o_j\right) \nabla_\theta s(u, j) \right| \\
&\leq (1 - \gamma) \sum_{j \in \mathcal{I}_\mathcal{B}} \left|-P\left(j|u\right) + o_j\right| \cdot C
\end{aligned}$$

where the upper bound of bias δ is determined by the hyperparameter γ and the deviation between the sampled softmax distribution and the inverse-popularity-based distribution. We suggest using appropriate γ to control the level of bias δ, since BMNS will be significantly biased from the softmax distribution when $\gamma \to 0$. Finally, we highlight the two advantages of BMNS: 1) BMNS generates additional virtual negatives without introducing additional encoding computational costs. 2) BMNS enforces the model training by mixing the gradient of negative to achieve better representation learning.

Table 1: Details of Datasets

Dataset	User	Item	Interaction	Sparsity
Gowalla	29,858	40,988	1,027,464	99.9160%
Tmall	125,553	58,058	2,064,290	99.9717%
Yelp	77,277	45,638	2,105,480	99.9403%
Amazon	130,380	128,939	2,415,650	99.9856%

3.4 Complexity Analysis

Time complexity. As mentioned above, BMNS trains the model using both in-batch negative samples and generated negative samples. Therefore, compared to in-batch sampling, BMNS needs to pay an additional sampling and mixing burden. Sampling from Beta distribution and weight normalization have a complexity of $O\left(T_s \times K \times |\mathcal{B}| + K \times |\mathcal{B}|\right)$ (Line 8-11 in Alg 1), where T_s denotes the time for sample an coefficient from Beta distribution. Considering the expectation of $\text{Beta}(\alpha, \alpha)$ is equivalent to $U(0, 1)$, we choose a simpler uniform distribution as the surrogate for Beta distribution. In the worst-case scenario, the time complexity of calculating score matrix (Line 12 in Alg 1) is $O\left(K \times |\mathcal{B}| \times |\mathcal{B}|\right)$. However, with the help of excellent algorithm and GPU parallel computing, we only need to pay computational cost far less than this upper bound. The complexity of inner-product takes $O\left(|\mathcal{B}| \times |\mathcal{B}| \times d_D\right)$, which is acceptable. Moreover, in the computing of training loss, BMNS consists two parts, leading to $O\left(|\mathcal{B}| \times K + |\mathcal{B}| \times |\mathcal{B}|\right)$) complexity.

Space complexity. Regardless of the space occupation caused by dataset loading and model parameters, the main memory cost in BMNS is the storage of mixing coefficient. Therefore, the space complexity is $O\left(K \times |\mathcal{B}|\right)$, which is proportional to the number of samples drawn from the uniform distribution.

4 EXPERIMENT RESULTS

4.1 Experiment Settings

4.1.1 Dataset Setting. In this paper, we conduct experiments on four public datasets: Gowalla, Tmall, Yelp and Amazon. **Gowalla** [1] is a check-in dataset that collects users' check-in location information. **Tmall** [2] records users' behavior (e.g., click and purchase) on the Tmall platform and used in IJCAI16 contest [23]. **Yelp** [3] contains reviews from users on yelp's website. In this paper, we directly use the preprocessed dataset in [6]. **Amazon** [4] includes user rating data for Amazon books. Fowllowing previous setting in [6], we consider items rated above 4 in the explict dataset as positive samples.

To alleviate data sparsity, we adopt 5-core strategy to filter dataset, i.e., retain users and items with at least 5 interactions. The statistic details of these datasets are summaried in Tabel 1. We use a hold-out strategy to partition the dataset, where for each user, 80% of items are used for training and the rest are used for testing. The hyperparameters of all methods are tuned through cross-validation.

[1] **Gowalla:** http://snap.stanford.edu/data/loc-gowalla.html
[2] **Tmall:** https://tianchi.aliyun.com/dataset/53
[3] **Yelp**: https://www.yelp.com/dataset
[4] **Amazon:** http://jmcauley.ucsd.edu/data/amazon

Table 2: Overall Performance Comparison, δ = 1e-4

	Gowalla			Tmall		
	NDCG@10	NDCG@50	Recall@50	NDCG@10	NDCG@50	Recall@50
SSL	0.1440±5.0δ	0.2942±7.0δ	0.2653±4.8δ	0.0495±2.5δ	0.1052±1.9δ	0.1081±2.3δ
SSL-Pop	0.1597±9.4δ	0.3027±8.3δ	0.2640±7.1δ	0.0643±3.9δ	0.1405±5.8δ	0.1481±5.4 δ
correct-sfx	0.1560±7.0δ	0.3100±9.6δ	0.2767±8.7δ	0.0512±1.1δ	0.1073±1.8δ	0.1101±2.2δ
MNS	0.1602±9.5δ	0.3038±10.4δ	0.2651±8.6δ	0.0655±3.9δ	0.1421±4.9δ	0.1500±4.5δ
BMNS	**0.1720±7.4**δ	**0.3392±9.9**δ	**0.3022±10.8**δ	**0.0717±4.6**δ	**0.1512±4.6**δ	**0.1580±4.5**δ
Improvement	7.37%	9.42%	9.22%	9.47%	6.40%	5.33%
	Yelp			Amazon		
	NDCG@10	NDCG@50	Recall@50	NDCG@10	NDCG@50	Recall@50
SSL	0.0672±2.8δ	0.1627±2.2δ	0.1559±3.4δ	0.0815±2.0δ	0.1592±2.3δ	0.1767±3.0δ
SSL-Pop	0.0662±1.9δ	0.1680±6.0δ	0.1614±7.1δ	0.0941±3.8δ	0.1828±3.9δ	0.1997±4.6δ
correct-sfx	0.0673±4.1δ	0.1623±5.6δ	0.1548±6.6δ	0.0863±3.9δ	0.1673±3.5δ	0.1853±4.4δ
MNS	0.0661±3.8δ	0.1669±4.6δ	0.1603±4.3δ	0.0957±3.0δ	0.1845±2.9δ	0.2014±3.7δ
BMNS	**0.0779±2.6**δ	**0.1927±7.6**δ	**0.1853±8.2**δ	**0.0970±3.2**δ	**0.1888±2.9**δ	**0.2052±4.2**δ
Improvement	15.75%	14.70%	14.81%	1.36%	2.33%	1.89%

4.1.2 Evaluation Metrics . We choose *Normalized Discounted Cumulative Gain* (NDCG) [15] and Recall to evaluate the performance of recommenders. The NDCG metric introduces the positional information of the top-N list and further normalizes each user. The Recall metric is the proportion of top-N items found in the test data. In our experiments, the cutoff number N is chosen from $\{10, 50\}$.

4.1.3 Implementation Settings. We use the two-tower model as the basic recommender for retrieval task and all methods are implemented by PyTorch 1.13.0 on a Linux operating system. For clearer comparison, we only use ID feature as model input and the dimension of the embedding layer is fixed by 32. Our experiments use Adam as the parameter optimizer, with a batch size of 2048 for each dataset, where the number of training epochs is fixed as 100. The learning rate and l_2-regularization are searched in $\{1e-2, 5e-3, 1e-3\}$ and $\{1e-4, 1e-5, 1e-6\}$, respectively. In our proposed method, the generated negative number K is consistent with the batch size by default and the weight of loss function γ is tuned in range $[0, 1]$ with a 0.1 step size.

4.1.4 Baselines.

- SSL is the vanilla sampling method for sampled softmax loss which directly samples in-batch items without bias correction. SSL-Pop considers the proposal sampling distribution as the popularity of items across the entire corpus, which can effectively mitigate the significant sampling bias, as shown in Eq (1).
- correct-sfx [27] improves correction term using streaming frequency estimation algorithm. It estimates the frequency probability by calculating the average interval between two consecutive hits of the item with hash techniques.
- MNS [26] adopts a mixture distribution based on the popularity and the uniform distribution in a certain proportion. The additional uniform negatives sampled from the entire corpus alleviate the problem of selection bias. Following the setting of BMNS, the number of uniform negatives is set as same as the batch size.

4.2 Performance Comparison

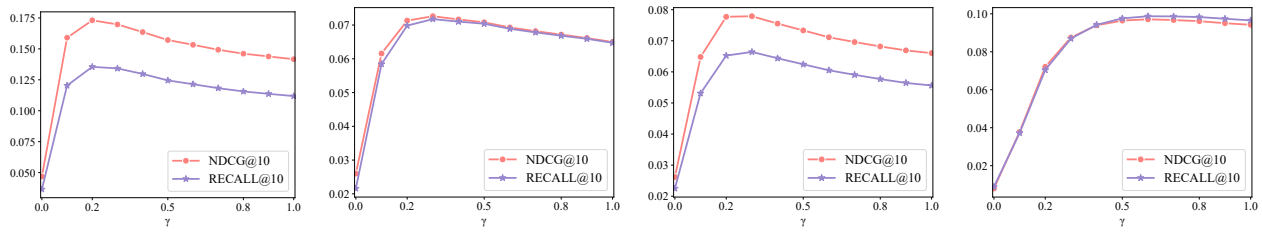
We run 5 times experiments with different random seeds (i.e., seed = $10, 20, \ldots, 50$) for each method and report the average performance and standard deviation. The results are summarized in Table 2. The best method is bold and the second-best method is underlined. We have the following observations: SSL, which does not incorporate with the item frequency, shows less favorable results over three datasets. Among Gowalla, Tmall and Amazon, SSL performs the worst, where in-batch sampling incurs sample selection bias but no corrected bias is provided. This indicates the necessity of correcting the sample selection caused by the in-batch sampling.

Both MNS and BMNS achieve better performance on relatively large datasets, i.e., Tmall and Amazon. This finding suggests that introducing additional negatives outside the batch contribute to reduce selection bias, and capturing global information in the expanded sample space is beneficial to model training, which has a less biased approximation for the partition function.

The proposed method BMNS consistently outperforms all baselines. Specifically, BMNS achieves relative 7.37%, 9.47%, 15.75%, 1.36% improvements of NDCG@10 compared to the strongest baseline for each dataset. With regard to Recall@50 metric, the relative improvements are 9.22%, 5.33%, 14.81%, 1.89%. The results fully demonstrate the significant improvement brought by the BMNS.

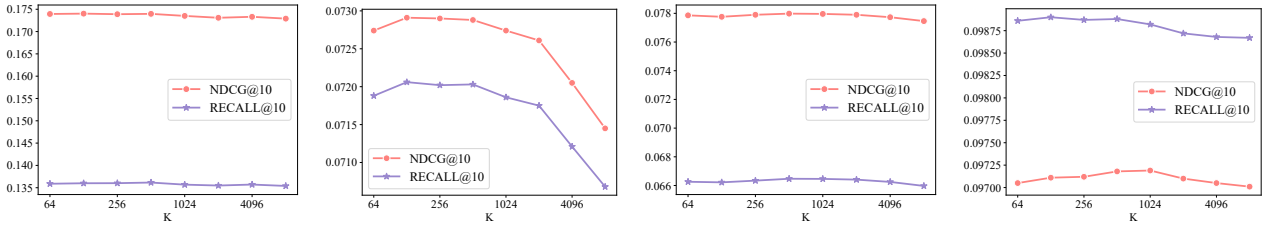
4.3 Varying the loss coefficient γ

The hyperparameter γ controls the proportion of in-batch loss and batch-mix loss. To investigate the impact of loss coefficient on model performance, we conduct experiments under various settings. We traverse the loss coefficient γ from 0 to 1 with a step size of 0.1. The results on different datasets are presented in Figure 2, and we observe the following phenomena: 1) On all datasets, the peak point corresponding to the ideal γ value is determined by the dataset distribution. As γ increases, the performance curve shows an upward trend, followed by a downward trend. 2) The model will



(a) Gowalla (b) Tmall (c) Yelp (d) Amazon

Figure 2: The effect of γ



(a) Gowalla (b) Tmall (c) Yelp (d) Amazon

Figure 3: The effect of generated negative number

experience a sharp degradation as γ asymptotically approaches 0, which is radically different from the pattern as γ approaches 1.

For the first phenomenon, we attribute the reason to the adaptability of BMNS to different dataset distributions. An appropriate choice of γ can provide beneficial information for model training and make the training process more robust. However, using only generated negative samples would lead to a distorted sampling distribution, resulting in the second phenomenon. The reason may lie in that the huge gradient bias from the softmax loss.

4.4 Varying the number of generated samples

In the aforementioned experiments, we assume that the number of generated negative samples K is the same as the batch size $|\mathcal{B}|$. Usually, the batch size is large during model training, e.g., 2048. As a result, significant time costs are incurred for generating new negatives. In this section, we will release this restriction and generate different number of batch-mix negatives to study the impact of K on the recommendation performance. In specific, we select K within a wide range $\{64, 128, 256, 512, 1024, 2048, 4096\}$ to conduct comparison experiments.

From Figure 3, we can see the number of generated negative number K slightly affects the performance of the retriever, especially on the Gowalla and Yelp datasets. On Tmall and Amazon datasets, increasing the number of negative samples may not necessarily lead to better performance. Taking into account the trade-off between performance and efficiency, we believe that generating 128 negative samples is a relatively better choice.

4.5 Effect of Batch Size

We conduct further investigation into the influence of batch size on BMNS. In this section, the number of generated negatives K is consistent with the batch size. As shown in Figure 4, we compare the NDCG@10 improvement of BMNS with SSL-Pop on four datasets. An important observation is that as the batch size increases, the relative improvement on Yelp and Amazon datasets also increases, whereas the conclusion is oppositive on Gowalla and Tmall datasets. Therefore, the batch size of 2048 is more advantageous and yields relatively balanced improvement in the previous experiments.

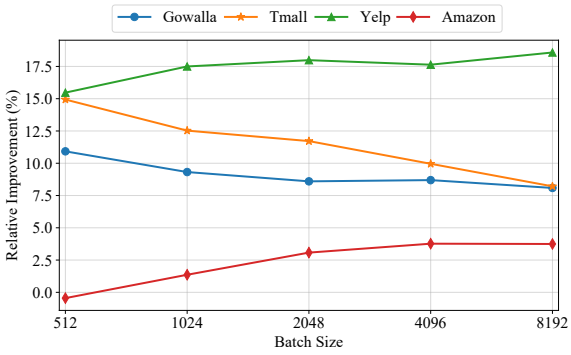
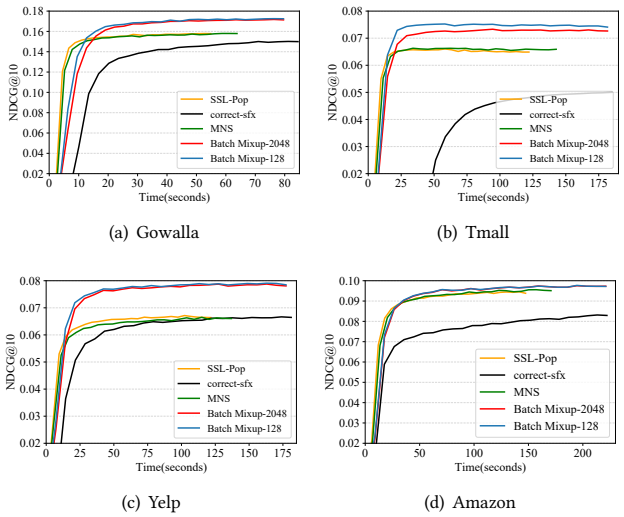


Figure 4: Batch Size vs. Relative Improvement

4.6 Running Time Experiments

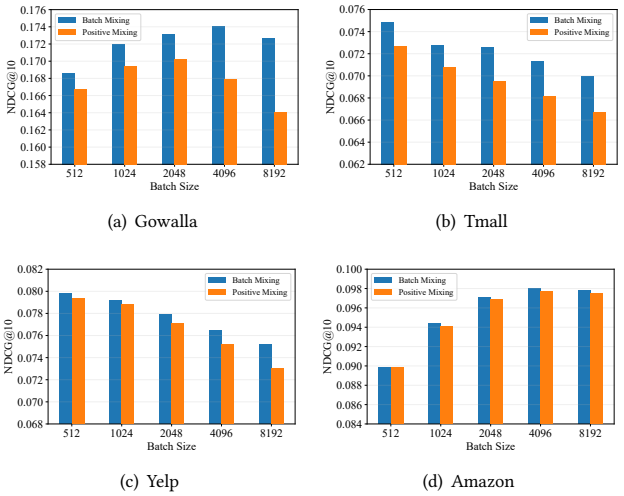
To demonstrate the efficiency of the proposed method, we conduct running time experiments to compare the convergence efficiency of different sampling method on four datasets . As for BMNS, the number of generated negatives K affects its running speed. Therefore, we use multiple configurations, e.g., $K = 128$ and $K = |\mathcal{B}|$, to plot their performance curves respectively. We also choose SSL-Pop, correct-sfx and MNS as baselines for comparison. In the experiments, we run each sampling method for 50 epochs and evaluate the performance every two epochs. We repeat each experiment 5



(a) Gowalla (b) Tmall

(c) Yelp (d) Amazon

Figure 5: Running time vs. Performance convergence



(a) Gowalla (b) Tmall

(c) Yelp (d) Amazon

Figure 6: Performance comparison of mixing methods over different batch size on four datasets

times and report the average results in the Figure 5. The results demonstrate the advantages of BMNS from two aspects: 1) High training efficiency. BMNS achieves better recommendation performance within the same training time, which reflects the excellent training efficiency of BMNS. 2) Better performance. When all the algorithms reach convergence, BMNS leads to a clear margin of performance improvement compared to other baselines.

In Section 3.4, we discuss the time complexity of BMNS. The additional computational cost incurred by the sampling and mixing stage mainly stems from the mixing coefficient sampling and the generation of virtual negative samples. During the training stage, we also need to spend extra time on the loss computing of generated negatives. Note that the complexity of these operations is closely related to the size of K. In other words, if we decrease the value of K, i.e., the number of generated negatives, the additional running time required for BMNS will also decrease. As depicted in Figure 5, convergence time would notably reduce when generating only 128 negative samples. Moreover, since the number of generated negatives has little impact on model performance, it still works well compared with the default setting. Generally speaking, BMNS improves the recommendation performance of the two-tower model without consuming a lot of training time, thus possessing the potential for application in practical scenarios.

Table 3: The impact of correction

dataset	Gowalla			Tmall		
metric	N@10	N@50	R@50	N@10	N@50	R@50
uniform	0.1722	0.3391	0.3021	0.0712	0.1505	0.1573
uniform+correct	0.1720	0.3392	0.3022	0.0717	0.1512	0.1580
dataset	Yelp			Amazon		
metric	N@10	N@50	R@50	N@10	N@50	R@50
uniform	0.0778	0.1926	0.1852	0.0970	0.1887	0.2051
uniform+correct	0.0779	0.1927	0.1853	0.0970	0.1888	0.2052

4.7 Effect of Corrected Strategy

During the sampling stage, we sample mixing coefficient from uniform distribution and adopt log Q correction to enable unpopular items are assigned higher mixing weights. In order to investigate the role of the correction term, we conduct experiments on four datasets to compare the effect of correction term. As shown in Table 3, in the vast majority of cases, correcting the mixing coefficient can bring a certain performance improvement, which is relatively evident on the Tmall dataset.

4.8 Comparison of different mix method

BMNS utilize all the in-batch negatives to generate the new negative. A trivial solution is the positive mixing proposed in MixGCF [14], which improves the quality of negatives by injecting the information of positive samples into negatives. In particular, for given interacted item i and in-batch item j, the score of new negative k is calculated by:

$$s^{mix}(u,k) = \alpha^{(k)} s(u,i) + (1-\alpha^{(k)}) s(u,j) \tag{8}$$

where $\alpha^{(k)} \in U(0,1)$. The training pair (u,i) denotes a positive sample while (u,j) represents a negative sample. For a fair comparison, all the mixing strategies is based on the same loss function, i.e., Eq (6). We report the performance of mixing methods under different batch size. The results are shown in Figure 6 and we have the following observations:

On all datasets, BMNS outperforms positive mixing. Especially on the Gowalla and Tmall datasets, batch mixing results in an obvious margin compared with positive mixing. This observation demonstrates the contribution of BMNS. The scale of batch size has an important effect on performance. Generally, batch mixing achieves a more significant improvement on larger batch size.

MixGCF is optimized under the BPR loss function, which requires fewer negative samples, e.g., near 5 samples would perform good performance. The sampled softmax usually requires much more negative samples, and pos-mixing strategy would get poor

performance with over-emphasized positive samples. Besides, the MF encoder may not be suitable for the MixGCF proposed for GNN.

5 CONCLUSION

In this paper, we propose a Batch-Mix Negative Sampling (BMNS) strategy for two-tower model training via mixing in-batch negatives. Our method extracts additional virtual negatives from the representation space of in-batch negatives to supplement the deficiency of in-batch negative sampling. Furthermore, we develop a mixture loss function to facilitate the coordinated training of in-batch negatives and virtual negatives. The introduction of additional negatives leads the model to benefit from different information and obtain more robust training. The empirical experiments on four real-world datasets fully demonstrate the effectiveness and efficiency of BMNS.

ACKNOWLEDGMENTS

This work is partially supported by NSFC (No. 61972069, 61836007, 61832017, 62272086), Shenzhen Municipal Science and Technology R&D Funding Basic Research Program (JCYJ20210324133607021), Municipal Government of Quzhou under Grant No. 2022D037, and Key Laboratory of Data Intelligence and Cognitive Computing, Longhua District, Shenzhen.

A GRADIENT ANALYSIS

Given the fixed mini-batch $\mathcal{B}$ and the user-item pair (u, i), the expectation of the loss gradient $\nabla_\theta \mathcal{L}(u,i)$ can be written as:

$$\begin{aligned}\mathbb{E}\left[\nabla_\theta \mathcal{L}(u,i)\right] =& -\nabla_\theta s^c(u,i) + \gamma \sum_{j\in\mathcal{I}_\mathcal{B}} P(j|u)\nabla_\theta s^c(u,j) \\ &+ (1-\gamma)\mathbb{E}\left[\sum_{k\in\mathcal{K}_\mathcal{B}} P^*(k|u) \sum_{j\in\mathcal{I}_\mathcal{B}} \hat{w}_j^{(k)} \nabla_\theta s(u,j)\right]\end{aligned} \tag{9}$$

Here, we observe that $\nabla s^c = \nabla s$. The last term can be expanded as:

$$\begin{aligned}&\mathbb{E}\left[\sum_{k\in\mathcal{K}_\mathcal{B}} P^*(k|u) \sum_{j\in\mathcal{I}_\mathcal{B}} \hat{w}_j^{(k)} \nabla_\theta s(u,j)\right] \\ =&\mathbb{E}\left[\sum_{k\in\mathcal{K}_\mathcal{B}} \frac{\exp s^{mix}(u,k)}{\sum_{l\in\mathcal{K}_\mathcal{B}} \exp s^{mix}(u,l)} \sum_{j\in\mathcal{I}_\mathcal{B}} \hat{w}_j^{(k)} \nabla_\theta s(u,j)\right] \\ =&\mathbb{E}\left[\sum_{k\in\mathcal{K}_\mathcal{B}} \frac{\exp\left(\sum_{m\in\mathcal{I}_\mathcal{B}} \hat{w}_m^{(k)} s(u,m)\right)}{\sum_{l\in\mathcal{K}_\mathcal{B}} \exp\left(\sum_{n\in\mathcal{I}_\mathcal{B}} \hat{w}_n^{(l)} s(u,n)\right)} \sum_{j\in\mathcal{I}_\mathcal{B}} \hat{w}_j^{(k)} \nabla_\theta s(u,j)\right]\end{aligned}$$

For those generated negative l, i.e., $l \in \mathcal{K}_\mathcal{B}$ and $l \neq k$, we obtain the following expectations:

$$\begin{aligned}&\mathbb{E}_{l\in\mathcal{K}_\mathcal{B}, l\neq k}\left[\sum_{n\in\mathcal{I}_\mathcal{B}} \hat{w}_n^{(l)} s(u,n)\right] \\ =&\mathbb{E}_{l\in\mathcal{K}_\mathcal{B}, l\neq k}\left[\sum_{n\in\mathcal{I}_\mathcal{B}} \frac{\exp\left(w_n^{(l)} - \log pop(n)\right)}{\sum_{m\in\mathcal{I}_\mathcal{B}} \exp\left(w_m^{(l)} - \log pop(m)\right)} s(u,n)\right] \\ =&\sum_{n\in\mathcal{I}_\mathcal{B}} \frac{1/pop(n)}{\sum_{m\in\mathcal{I}_\mathcal{B}} 1/pop(m)} s(u,n) = o\end{aligned}$$

Thus, we have the following term:

$$\begin{aligned}&\mathbb{E}\left[\sum_{k\in\mathcal{K}_\mathcal{B}} P^*(k|u) \sum_{j\in\mathcal{I}_\mathcal{B}} \hat{w}_j^{(k)} \nabla_\theta s(u,j)\right] \\ =&\mathbb{E}\left[\sum_{j\in\mathcal{I}_\mathcal{B}} \sum_{k\in\mathcal{K}_\mathcal{B}} \frac{\hat{w}_j^{(k)} \exp\left(\sum_{m\in\mathcal{I}_\mathcal{B}} \hat{w}_m^{(k)} s(u,m)\right)}{\exp\left(\sum_{m\in\mathcal{I}_\mathcal{B}} \hat{w}_m^{(k)} s(u,m)\right) + (|\mathcal{K}_\mathcal{B}|-1)\exp o} \nabla_\theta s(u,j)\right]\end{aligned}$$

Similarly, we get the following expectation:

$$\begin{aligned}&\mathbb{E}_{k\in\mathcal{K}_\mathcal{B}}\left[\frac{\hat{w}_j^{(k)} \exp\left(\sum_{m\in\mathcal{I}_\mathcal{B}} \hat{w}_m^{(k)} s(u,m)\right)}{\exp\left(\sum_{m\in\mathcal{I}_\mathcal{B}} \hat{w}_m^{(k)} s(u,m)\right) + (|\mathcal{K}_\mathcal{B}|-1)\exp o}\right] \\ =&\mathbb{E}_{k\in\mathcal{K}_\mathcal{B}}\left[\frac{\exp\left(w_j^{(k)}\right)/pop(j)}{\sum_{l\in\mathcal{I}_\mathcal{B}} \exp\left(w_l^{(k)}\right)/pop(l)} \cdot \frac{1}{1 + \frac{(|\mathcal{K}_\mathcal{B}|-1)\exp o}{\exp\left(\sum_{m\in\mathcal{I}_\mathcal{B}} \hat{w}_m^{(k)} s(u,m)\right)}}\right] \\ =&\mathbb{E}_{k\in\mathcal{K}_\mathcal{B}}\left[\frac{\exp\left(w_j^{(k)}\right)/pop(j)}{\sum_{l\in\mathcal{I}_\mathcal{B}} \exp\left(w_l^{(k)}\right)/pop(l)} \cdot \frac{1}{1 + \frac{(|\mathcal{K}_\mathcal{B}|-1)\exp o}{\exp\left(\frac{\sum_{m\in\mathcal{I}_\mathcal{B}} s(u,m)\exp\left(w_m^{(k)}\right)/pop(m)}{\sum_{n\in\mathcal{I}_\mathcal{B}} \exp\left(w_n^{(k)}\right)/pop(n)}\right)}}\right] \\ =&\mathbb{E}_{k\in\mathcal{K}_\mathcal{B}}\left[\frac{\exp(1/2)/pop(j)}{\sum_{l\in\mathcal{I}_\mathcal{B}} \exp(1/2)/pop(l)} \cdot \frac{1}{1 + \frac{(|\mathcal{K}_\mathcal{B}|-1)\exp o}{\exp\left(\frac{\sum_{m\in\mathcal{I}_\mathcal{B}} s(u,m)\exp(1/2)/pop(m)}{\sum_{n\in\mathcal{I}_\mathcal{B}} \exp(1/2)/pop(n)}\right)}}\right] \\ =&\mathbb{E}_{k\in\mathcal{K}_\mathcal{B}}\left[\frac{1/pop(j)}{\sum_{l\in\mathcal{I}_\mathcal{B}} 1/pop(l)} \cdot \frac{1}{1 + \frac{(|\mathcal{K}_\mathcal{B}|-1)\exp o}{\exp\left(\frac{\sum_{m\in\mathcal{I}_\mathcal{B}} s(u,m)/pop(m)}{\sum_{n\in\mathcal{I}_\mathcal{B}} 1/pop(n)}\right)}}\right] \\ =&\frac{1/pop(j)}{\sum_{l\in\mathcal{I}_\mathcal{B}} 1/pop(l)} \cdot \frac{1}{1 + \frac{(|\mathcal{K}_\mathcal{B}|-1)\exp o}{\exp o}} = \frac{1/pop(j)}{|\mathcal{K}_\mathcal{B}| \sum_{l\in\mathcal{I}_\mathcal{B}} 1/pop(l)}\end{aligned}$$

Thus, the expectation of the last term in Eq (9) can be simplified as:

$$\begin{aligned}&\mathbb{E}\left[\sum_{k\in\mathcal{K}_\mathcal{B}} P^*(k|u) \sum_{j\in\mathcal{I}_\mathcal{B}} \hat{w}_j^{(k)} \nabla_\theta s(u,j)\right] \\ =&\mathbb{E}\left[\sum_{j\in\mathcal{I}_\mathcal{B}} \frac{1/pop(j)}{\sum_{l\in\mathcal{I}_\mathcal{B}} 1/pop(l)} \nabla_\theta s(u,j)\right] = \sum_{j\in\mathcal{I}_\mathcal{B}} o_j \nabla_\theta s(u,j)\end{aligned}$$

where $o_j = \frac{1/pop(j)}{\sum_{l\in\mathcal{I}_\mathcal{B}} 1/pop(l)}$. Finally, the gradient expectation of the integrated loss given the training pair (u, i) and the mini-batch follows as:

$$\begin{aligned}&\mathbb{E}\left[\nabla_\theta \mathcal{L}(u,i)\right] \\ =& -\nabla_\theta s(u,i) + \gamma \sum_{j\in\mathcal{I}_\mathcal{B}} P(j|u)\nabla_\theta s(u,j) + (1-\gamma) \sum_{j\in\mathcal{I}_\mathcal{B}} o_j \nabla_\theta s(u,j) \\ =& -\nabla_\theta s(u,i) + \sum_{j\in\mathcal{I}_\mathcal{B}} \left(\gamma P(j|u) + (1-\gamma) o_j\right) \nabla_\theta s(u,j)\end{aligned}$$

REFERENCES

[1] Yoshua Bengio and Jean-Sébastien Senecal. 2003. Quick Training of Probabilistic Neural Nets by Importance Sampling. In *Proceedings of the Ninth International Workshop on Artificial Intelligence and Statistics (Proceedings of Machine Learning Research, Vol. R4)*. 17–24.

[2] Yoshua Bengio and Jean-Sébastien Senécal. 2008. Adaptive importance sampling to accelerate training of a neural probabilistic language model. *IEEE Transactions on Neural Networks* 19, 4 (2008), 713–722.

[3] Olivier Chapelle, Jason Weston, Léon Bottou, and Vladimir Vapnik. 2000. Vicinal Risk Minimization. In *Proceedings of the 13th International Conference on Neural Information Processing Systems (NIPS'00)*. 395–401.

[4] Feihu Che, Guohua Yang, Pengpeng Shao, Dawei Zhang, and Jianhua Tao. 2022. MixKG: Mixing for harder negative samples in knowledge graph. *arXiv preprint arXiv:2202.09606* (2022).

[5] Jin Chen, Defu Lian, Binbin Jin, Xu Huang, Kai Zheng, and Enhong Chen. 2022. Fast variational autoencoder with inverted multi-index for collaborative filtering. In *Proceedings of the ACM Web Conference 2022*. 1944–1954.

[6] Jin Chen, Defu Lian, Binbin Jin, Kai Zheng, and Enhong Chen. 2022. Learning Recommenders for Implicit Feedback with Importance Resampling. In *Proceedings of the ACM Web Conference 2022*. 1997–2005.

[7] Jin Chen, Defu Lian, Yucheng Li, Baoyun Wang, Kai Zheng, and Enhong Chen. 2022. Cache-Augmented Inbatch Importance Resampling for Training Recommender Retriever. In *Advances in Neural Information Processing Systems*, Vol. 35. 34817–34830.

[8] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. 2016. Wide & deep learning for recommender systems. In *Proceedings of the 1st workshop on deep learning for recommender systems*. 7–10.

[9] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM conference on recommender systems*. 191–198.

[10] Jingtao Ding, Yuhan Quan, Quanming Yao, Yong Li, and Depeng Jin. 2020. Simplify and robustify negative sampling for implicit collaborative filtering. *Advances in Neural Information Processing Systems* 33 (2020), 1094–1105.

[11] Carlos A. Gomez-Uribe and Neil Hunt. 2016. The Netflix Recommender System: Algorithms, Business Value, and Innovation. *ACM Trans. Manage. Inf. Syst.* 6, 4, Article 13 (dec 2016), 19 pages.

[12] Hongyu Guo, Yongyi Mao, and Richong Zhang. 2019. Augmenting data with mixup for sentence classification: An empirical study. *arXiv preprint arXiv:1905.08941* (2019).

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

[14] Tinglin Huang, Yuxiao Dong, Ming Ding, Zhen Yang, Wenzheng Feng, Xinyu Wang, and 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 & Data Mining (KDD '21)*. 665–674.

[15] Kalervo Järvelin and Jaana Kekäläinen. 2002. Cumulated Gain-Based Evaluation of IR Techniques. *ACM Trans. Inf. Syst.* 20, 4 (oct 2002), 422–446.

[16] Jang-Hyun Kim, Wonho Choo, Hosan Jeong, and Hyun Oh Song. 2021. Co-mixup: Saliency guided joint mixup with supermodular diversity. *arXiv preprint arXiv:2102.03065* (2021).

[17] Defu Lian, Qi Liu, and Enhong Chen. 2020. Personalized ranking with importance sampling. In *Proceedings of The Web Conference 2020*. 1093–1103.

[18] Erik Lindgren, Sashank Reddi, Ruiqi Guo, and Sanjiv Kumar. 2021. In *Advances in Neural Information Processing Systems*, Vol. 34. 4134–4146.

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

[20] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian Personalized Ranking from Implicit Feedback. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence (UAI '09)*. 452–461.

[21] Zhiqiang Shen, Zechun Liu, Zhuang Liu, Marios Savvides, and Trevor Darrell. 2020. Rethinking image mixture for unsupervised visual representation learning. *arXiv preprint arXiv:2003.05438* 3, 7 (2020), 8.

[22] Wentao Shi, Jiawei Chen, Fuli Feng, Jizhi Zhang, Junkang Wu, Chongming Gao, and Xiangnan He. 2023. On the Theories Behind Hard Negative Sampling for Recommendation. In *Proceedings of the ACM Web Conference 2023*. 812–822.

[23] Tianchi. 2018. IJCAI-16 Brick-and-Mortar Store Recommendation Dataset. https://tianchi.aliyun.com/dataset/dataDetail?dataId=53

[24] Vikas Verma, Alex Lamb, Christopher Beckham, Amir Najafi, Ioannis Mitliagkas, David Lopez-Paz, and Yoshua Bengio. 2019. Manifold Mixup: Better Representations by Interpolating Hidden States. In *Proceedings of the 36th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 97)*. 6438–6447.

[25] Jinpeng Wang, Jieming Zhu, and Xiuqiang He. 2021. Cross-batch negative sampling for training two-tower recommenders. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 1632–1636.

[26] Ji Yang, Xinyang Yi, Derek Zhiyuan Cheng, Lichan Hong, Yang Li, Simon Xiaoming Wang, Taibai Xu, and Ed H Chi. 2020. Mixed negative sampling for learning two-tower neural networks in recommendations. In *Companion Proceedings of the Web Conference 2020*. 441–447.

[27] Xinyang Yi, Ji Yang, Lichan Hong, Derek Zhiyuan Cheng, Lukasz Heldt, Aditee Kumthekar, Zhe Zhao, Li Wei, and Ed Chi. 2019. Sampling-bias-corrected neural modeling for large corpus item recommendations. In *Proceedings of the 13th ACM Conference on Recommender Systems*. 269–277.

[28] Wenpeng Yin, Huan Wang, Jin Qu, and Caiming Xiong. 2021. BatchMixup: Improving training by interpolating hidden states of the entire mini-batch. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*. 4908–4912.

[29] Sangdoo Yun, Dongyoon Han, Sanghyuk Chun, Seong Joon Oh, Youngjoon Yoo, and Junsuk Choe. 2019. CutMix: Regularization Strategy to Train Strong Classifiers With Localizable Features. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*. 6022–6031.

[30] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. 2017. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412* (2017).

[31] Rongzhi Zhang, Yue Yu, and Chao Zhang. 2020. SeqMix: Augmenting Active Sequence Labeling via Sequence Mixup. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 8566–8579.

[32] Weinan Zhang, Tianqi Chen, Jun Wang, and Yong Yu. 2013. Optimizing Top-n Collaborative Filtering via Dynamic Negative Item Sampling. In *Proceedings of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '13)*. 785–788.