Cache-Augmented Inbatch Importance Resampling for Training Recommender Retriever

Anonymous Author(s)
Affiliation
Address
email

Abstract

Recommender retrievers aim to rapidly retrieve a fraction of items from the entire item corpus when a user query requests, with the representative two-tower model trained with the log softmax loss. For efficiently training recommender retrievers on modern hardwares, inbatch sampling, where the items in the mini-batch are shared as negatives to estimate the softmax function, has attained growing interest. However, existing inbatch sampling based strategies just correct the sampling bias of inbatch items with item frequency, being unable to distinguish the user queries within the mini-batch and still incurring significant bias from the softmax. In this paper, we propose a Cache-Augmented Inbatch Importance Resampling (χIR) for training recommender retrievers, which not only offers different negatives to user queries with inbatch items, but also adaptively achieves a more accurate estimation of the softmax distribution. Specifically, χIR resamples items for the given mini-batch training pairs based on certain probabilities, where a cache with more frequently sampled items is adopted to augment the candidate item set, with the purpose of reusing the historical informative samples. χIR enables to sample query-dependent negatives based on inbatch items and to capture dynamic changes of model training, which leads to a better approximation of the softmax and further contributes to better convergence. Finally, we conduct experiments to validate the superior performance of the proposed χIR compared with competitive approaches.

1 Introduction

Recommender systems (RS) have been prevalently developmented to address the issue of information overload, with the purpose of mining potential preferred items for users, bringing great commercial revenues to enterprises. Recommender retrievers, which retrieve a subset of highly relevant items from numerous items, are a key component of the commonly-used two-stage architecture [6, 10, 27]. The two-tower models, where the queries and items are represented with separate models, are representative of retrieval models. Generally, given the similarity scores of user and item embeddings, the models attempt to minimize the loss function [7, 14], i.e., the log-softmax loss, guiding the interacted items having higher similarity scores. However, it is extremely inefficient to calculate the partition function in the log-softmax loss over the item corpus. Negative Sampling, which samples a number of items to estimate the partition function or the gradient, has attained growing interests in recommender retrievers [13, 19, 24, 27, 28].

A straightforward method is to take the static uniform distribution as the proposal distribution to sample items from the entire item set. Nevertheless, restricted by the computational resource, a more practical strategy is *inbatch sampling* [12, 27], where the other training samples in the mini-batch are shared as negative samples. Considering the complicated features in online systems, it is time-consuming to get the embeddings of users and items via the tower model. Encoding the

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

sampled unseen items outside the batch occupies additional huge computational cost. Thus, the inbatch sampling is more efficient and profitable for online systems. There exists a sampling bias of training data in inbatch sampling, since the interactions are received from user logs and the item exposure follows the long-tailed distribution. Accordingly, several works correct the sampling bias using the popularity of items [28, 30] and have been successfully exploited into the online systems.

However, there still exists a huge bias of the approximated gradient from the full softmax with the inbatch sampling. The sampling strategies eventually fit in the sampled-softmax, which approximates the exact gradient of softmax loss depending on the importance sampling. The bias can be eliminated by forcing the proposal distribution, i.e., sampling distribution, to the softmax distribution [2]. Previous works [28, 27] indeed correct the sampling distribution for inbatch sampling as the frequency-based distribution (or popularity-based distribution), which remains a query-independent proposal. The ideal softmax distribution is dependent on different queries and changes dynamically during the training process. Existing studies [9, 18, 29, 26, 31] in negative sampling for recommender systems have proved that adaptive samplers lead to better convergence of the recommendation quality. Thus, there is an urgent need to develop an adaptive selection strategy for inbatch sampling to reduce the bias towards the ideal softmax distribution.

To this end, we propose a Cache-Augmented Inbatch Importance Resampling, shorted as χIR, for training recommender retrievers. Concretely, we resample the batch items according to the modified softmax weight over the mini-batch, which offers different negatives to user queries, instead of sharing a same set of batch items. This importance resampling based selection not only distinguish users but also provides a more accurate estimation of the full softmax. Furthermore, inspired by the recent work [27], which introduces additional uniformly sampled negatives to tackle the sampling bias, a cache mechanism is utilized to further eliminate the approximation bias caused by the sampling bias of the inbatch sampling. Rather than the uniformly sampled items, we cache the more frequently sampled items into a fixed-size set, usually having the same size as the mini-batch. The frequently sampled items tend to have higher similarity scores on average, which can be regarded as *hard* negative samples for training models. Therefore, the cache plays a role of item augmentation since more informative negatives are introduced to accelerate the model convergence. The proposed χIR is then evaluated on five public real-world datasets, demonstrating the superior performance over competitive baseline approaches in terms of effectiveness.

2 Preliminaries

2.1 Recommender Retrievers

Given the user query, recommender retrievers aim to rapidly response with highly relevant items from the full item corpus. Under the paradigm of the two-tower models, the user queries and items are represented by $\{\boldsymbol{q}_i \in \mathbb{R}^{d_u}\}_{i=1}^{M}$ and $\{\boldsymbol{e}_j \in \mathbb{R}^{d_i}\}_{j=1}^{N}$, where M and N denote the number of queries and items, respectively. Here, we assume that the number of queries is finite for simplification. The user tower and item tower then map the feature vectors to a k-dimensional embedding space with separate functions $\phi_Q : \mathbb{R}^{d_u} \to \mathbb{R}^k$ and $\phi_I : \mathbb{R}^{d_i} \to \mathbb{R}^k$, respectively. A similarity scorer s, such as the widely used inner product function $s(u, i) = \langle \phi_Q(\boldsymbol{q}_u), \phi_I(\boldsymbol{e}_i) \rangle$, outputs the relevance of the query u and candidate item i. In this paper, we investigate the log-softmax loss for training the two-tower models, where the preference probability of the item with respect to the query u and item i is calculated with $P(i|u) = \frac{\exp(s(u,i))}{\sum_{j \in \mathcal{I}} \exp(s(u,j))}$, where $\mathcal{I}$ denotes the candidate item corpus. By maximizing the likelihood, the objective function follows as:

$$\mathcal{L}_{\text{softmax}}(\mathcal{D}, \Theta) = -\frac{1}{|\mathcal{D}|} \sum_{(u,i) \in \mathcal{D}} \log P(i|u) = -\frac{1}{|\mathcal{D}|} \sum_{(u,i) \in \mathcal{D}} \log \frac{\exp\left(s\left(u, i\right)\right)}{\sum_{j \in \mathcal{I}} \exp\left(s\left(u, j\right)\right)} \tag{1}$$

where Θ denotes the model parameters in ϕ_Q and ϕ_I. $\mathcal{D}$ denotes the training data, each tuple of which records an interaction between the user query u and the item i. The two-tower model is such trained and the interacted items, i.e., positive pairs, are encouraged to get higher scores than uninteracted items. The maximum inner product search (MIPs) then can be fruitfully implemented, such as SPANN [5], ScaNN [11] and FAISS [17] to search for the topk items given a user query, meeting the requirements of rapid online response.

When the number of items N becomes extremely large, it would incur a substantial computational cost of the partition function, i.e., the denominator in Eq (1). Intuitively, the partition function can be estimated by sampling a subset from candidate items. The sampled softmax loss then behaves as the objective function,

$$\mathcal{L}_{\text{sampled_softmax}}(\mathcal{D}, \Theta) = -\frac{1}{|\mathcal{D}|} \sum_{(u,i)\in\mathcal{D}} \log \frac{\exp\left(s'\left(u,i\right)\right)}{\sum_{j\in\mathcal{S}} \exp\left(s'\left(u,j\right)\right)} \tag{2}$$

where $\mathcal{S}$ represents the subset of the sampled items. $s'\left(u,i\right) = s\left(u,i\right) - \log p(i|u)$ denotes the corrected logit with the sampling probability. Obviously, the computational cost is considerably reduced, which only depends on the size of the sampled subset rather than the total item corpus.

2.2 Negative Sampling in RS

To guarantee the unbiased gradient estimation of the sampled softmax, the softmax distribution is the ideal proposal distribution [2, 15], which is prohibitively expensive given the huge number of candidates and features. To meet the requirements of quick response, the query-independent sampling distribution, such as the uniform distribution, is applied to sample items. These distributions remain constant during the dynamic model training and they will select easy samples in later training epochs, which contribute a limited gradient to model convergence. The adaptive samplers are further investigated to sample more informative items depending on the model and can be grouped into two categories. One leverages techniques of adversarial frameworks [8, 20, 25], where negative samples are generated through the complicated networks. The other category attempts to assign a higher probability to items with larger similarity scores [16, 23, 29] or to reduce the bias of the sampling distribution from the softmax distribution [18, 4]. One promising approach is the two-pass sampler [1, 29], which samples a fixed-sized pool of items from the simple static distribution and then selects the items with the highest similarity scores in the pool to calculate the loss. Due to its intractable sampling distribution, it is seldom cooperated within the sampled softmax loss and would perform worse due to its huge approximation bias from the softmax distribution.

However, in the online service of recommender systems with a huge magnitude of items and features, the query and item embeddings are constantly calculated through the large-scale deep towers. The inbatch sampling becomes popular since it avoids feeding additional negative items to the item tower [27], saving much computational cost. Moreover, the inbatch sampling does not improve the computational costs from the complexity perspective [22] but lowers the time cost on modern hardwares. To reduce the bias caused by the item exposure in online systems, several works correct the bias with item frequency which fits the sampled softmax loss with the item frequency as the proposal [7, 27, 28, 30].

3 Training Recommender Retrieves with cache-augmented inbatch importance resampling

The inbatch sampling has attracted more attention due to its high efficiency for deployment on modern hardwares in online systems. Existing works attempt to correct the sampling bias within the mini-batch by the item frequency, which finally acts as the popularity-based proposal distribution in the sampled softmax loss. However, considering the ideal softmax distribution, whose sampling probability is associated with the given query, there still exists a bias of gradient estimation from the softmax by just correcting the selection bias of inbatch sampling. To achieve a better approximation of the softmax, we propose the inbatch importance resampling (BIR for short) and the cache-augmented importance resampling (χIR), which not only delivers different negative items to different queries based on the inbatch items but also yields an adaptive softmax estimation during model training.

3.1 Reducing Bias of Inbatch Sampling based on Importance Resampling

Items within the mini-batch actually follow the popularity-based distribution since the observed items are exposed to users under the long-tailed distribution in today's recommender systems. Thus, a query-independent sampler, i.e., frequency-based sampler, fits the sampled softmax with the inbatch sampling. However, this query-independent sampler corrects the selection bias of the inbatch sampling but does not improve the estimation performance.

The importance resampling [3] is adopted here to further correct the bias from the ideal softmax distribution. Given the mini-batch B, each item is resampled for the query u based on the weight

$$w(i|u) = \frac{\exp\left(s(u,i) - \log pop(i)\right)}{\sum_{j\in B}\exp\left(s(u,j) - \log pop(j)\right)},$$

where the probability $pop(i)$ here denotes the frequency of the items summarized from the training data. The resampled item set with respect to the query u is denoted as $\mathcal{R}_u$.

By applying the resampling strategy, items with larger similarity scores have a higher chance of being sampled and may appear multiple times. Moreover, the sampled items for different user queries are different while the queries share the same item set in inbatch sampling. The following theorem provides the evidence that the sampled items based on the inbatch importance resampling can accurately approximate the softmax distribution.

Lemma 3.1. *Assume items follow a distribution $P = \{p_1, p_2, ..., p_N\}$, the item set $\mathcal{J}$ sampled from $P' = \{\frac{1}{p_1}, \frac{1}{p_2}, ..., \frac{1}{p_N}\}$ follows the uniform distribution when $|\mathcal{J}| \to \infty$.*

Theorem 3.1. *When the batch size $|B| \to \infty$, items in the resampled set $\mathcal{R}_u$ with respect to the query u based on sampling weight $w(i|u)$ follow the softmax distribution w.r.t. the query u.*

Proof. Assume $pop(i)$ denote the item frequency over whole item corpus, the probability of appearing in $\mathcal{R}_u$ can be obtained according to the multiplication rule in the following way:

$$\begin{aligned}
P(i|i \in \mathcal{R}_u) &= P(i|i \in B) \cdot w(i|u) \\
&= pop(i) \times \frac{|B|}{|\mathcal{D}|} \times \frac{\exp(s(u,i) - \log pop(i))}{\sum_{j\in B}\exp(s(u,j) - \log pop(j))} \\
&= \frac{|B|}{|\mathcal{D}|} \times \frac{\exp(s(u,i))}{\sum_{j\in B}\frac{1}{pop(i)} \cdot \exp(s(u,j))} \\
&\approx \frac{|B|}{|\mathcal{D}|} \times \frac{\exp(s(u,i))}{\frac{|B|}{|\mathcal{D}|} \times \sum_{j\in\mathcal{I}}\exp(s(u,j))} = \frac{\exp(s(u,i))}{\sum_{j\in\mathcal{I}}\exp(s(u,j))}
\end{aligned}$$

The approximately equal sign $\approx$ turns to the equal sign when $|B| \to \infty$. □

According to the resampling approach, the sampled items follows the softmax distribution given the user query. Let's first review the expectation of gradient of the full softmax loss as follows:

$$\nabla\mathcal{L}_{\text{softmax}} = -\nabla s(u,i) + \sum_{j}^{N}\nabla s(u,j) = -\nabla s(u,i) + \mathbb{E}_{j\sim P^*(i|u)}\nabla s(u,j)$$

where $P^*(i|u) = \frac{\exp(s(u,i))}{\sum_{j\in\mathcal{I}}\exp(s(u,j))}$. Since the sampled items follows the softmax distribution, the estimated gradient with the item set $\mathcal{R}_u$ achieves an asymptotic-unbiased approximation of the expectation of gradient, i.e., $\sum_{i\in\mathcal{R}_u}\nabla s(u,i) \approx \mathbb{E}_{j\sim P^*(i|u)}\nabla s(u,j)$. Therefore, the objective function in a mini-batch has the following form:

$$\mathcal{L}_{\text{BIR}}(\mathcal{D},\Theta) = \sum_{(u,i)\in B}\frac{\exp\left(s(u,i)\right)}{\sum_{j\in\mathcal{R}_u}\exp\left(s(u,j)\right)} \tag{3}$$

where $\mathcal{R}_u$ denotes the resampled item set with the same size as B. The Algorithm 1 sketches the pseudo-code for inbatch importance resampling. Compared with the naive inbatch sampling, BIR has the capability of distinguishing negative samples among different queries within inbatch items and capturing the dynamical changes of softmax, yielding an adaptive and flexible sampler to strengthen recommender retrievers. Note that the asymptotic-unbiased estimation is achieved when the batch size becomes extremely larger. A non-asymptotic bound can be attached with the following theorem.

Theorem 3.2. *Assum that the gradients of the logits $\nabla_\Theta s(u,i)$ have their coordinates bounded by G, the bias of $\nabla_\Theta\mathcal{L}_{BIR}$ satisfies:*

$$\begin{aligned}
\mathbb{E}[\nabla_\Theta\mathcal{L}_{BIR}(u)] \le \nabla_\Theta\mathcal{L}_{softmax}(u) + \mathbb{E}[U]\cdot\left(\frac{\sum_{i\in B}e^{s(u,i)}p(i)Z_{Bp} - Z_B^2}{|B|Z^3} + o\left(\frac{1}{|B|}\right)\right) + \\
\left(\frac{G}{|B|}\cdot\frac{\max_{j,l\in B}|p(j)-p(l)|Z_{Bp}\cdot Z_B}{Z^2 + \sum_{i\in B}e^{s(u,i)}p(i)Z_{Bp}} + o\left(\frac{1}{|B|}\right)\right)\cdot\mathbf{1}
\end{aligned}$$

Algorithm 1: Inbatch importance resampling (BIR)

Input: Training data $\mathcal{D} = \{(u, i)\}$, Number of epochs T
Output: Model parameters Θ

Statistically get item popularity $P = \{pop(i) | \forall i \in \mathcal{I}\}$ from $\mathcal{D}$;
for $e = 1, 2, ..., T$ **do**
  for *mini-batch* $B \in \mathcal{D}$ **do**
    $U = \{u | (u, i) \in B\}$, $I = \{i | (u, i) \in B\}$;
    Map query feature vectors into embeddings $E_U \in \mathbb{R}^{B \times k}$ based on ϕ_Q;
    Map item feature vectors into embeddings $E_I \in \mathbb{R}^{B \times k}$ based on ϕ_I;
    Calculate the matrix of similarity $S = E_U E_I^\top$;
    for $u \in U$ **do**
      Calculate the sampling probability $w(i|u) = \frac{\exp(s(u,i) - \log pop(i))}{\sum_{j \in I} \exp(s(u,j) - \log pop(j)}, \forall i \in I$;
      Resample a set of items $\mathcal{R}_u$ based on $w(i|u)$;
    Update model parameters based on the objective function, i.e., Eq (3);

where $\mathbb{E}[U] = \sum_{i \in \mathcal{I}} \exp(s(u, i)) \nabla_\Theta s(u, i)$, $Z_{Bp} = \sum_{j \in B} \exp(s(u, j) - \log p(j))$, $Z_B = \sum_{j \in B} \exp(s(u, j))$, $Z = \sum_{i \in \mathcal{I}} \exp(s(u, i))$. *The probability* $p(i)$ *refers to the popularity* $pop(i)$.

This theorem can be proved according to Rawat's work [21] and more details can be attached in Appendix. We can observe that the bias is correlated to the batch size and the popularity distribution within the mini-batch.

3.2 Reusing Historical Informative Items with Cache-Augmented Resampling

Increasing the sample size is another crucial approach to controlling the bias and variance. Inspired by the recent work, MNS[27], which uses a mixture of inbatch negative samples and additional uniformly sampled negatives, we design a paradigm with cache-augmented samples, with the aim of reusing highly informative items in previous training epochs.

MNS feeds the model with additional uniformly sampled items, alleviating the problem of long-tailed distribution to some extent. However, both the uniform distribution and popularity-based distribution are static from beginning to end, which may oversample the easy samples after several epochs. In order to reduce the bias of the query-based softmax during model training, we intuitively cache the items with averagely higher scores, depending on which historical harder negatives have the chance of being more trained.

More concretely, we count the occurrence number of times the items have been sampled into a vector $\boldsymbol{o} = \{o_1, o_2, ..., o_N\}$. If the item has been sampled for more times according to the aforementioned BIR, it has an averagely higher similar score among user queries, indicating that it has a higher probability of being hard negatives. The cache $\mathcal{C}$ is generated and updated by sampling items based on o_i, which serves as an additional candidate set to be resampled and augments the items from the original batch. Since the cache-augmented items follow a different distribution from the inbatch items, they can not share a unified normalization function. Thus, the items are resampled independently from the cache and the batch, and fit in two parts of loss as follows:

$$\mathcal{L}_{\chi\text{IR}}(\mathcal{D}, \Theta) = \lambda \sum_{(u,i) \in B} \frac{\exp(s(u, i))}{\sum_{j \in \mathcal{K}_u} \exp(s(u, j))} + (1 - \lambda) \sum_{(u,i) \in B} \frac{\exp(s(u, i))}{\sum_{j \in \mathcal{R}_u} \exp(s(u, j))} \tag{4}$$

where λ denotes a hyperparameter to control the influence of two separate parts. $\mathcal{K}_u$ denotes the item set resampled from the cache $\mathcal{C}$ based on the weight of $w_c = \frac{\exp(s(u,i) - \log q(i))}{\sum_{c \in \mathcal{C}} \exp(s(u,c) - \log q(c))}$, where $q(i) = pop(i)$. Note that the newly seen items are affected by a long-tailed distribution, we, therefore, correct the sampling bias for computing the normalized resampling weight. $\mathcal{R}_u$ refers to the resampled item set according to the inbatch importance resampling. In order to maintain a final resampled set with size $|B|$, we simply resample $\frac{1}{2}$ items from the cache and batch respectively, i.e., $|\mathcal{K}_u| = |\mathcal{R}_u| = \frac{|B|}{2}$. When newly resampled items are generated, the occurrence of items in the

resampled set is updated. The cache-augmented item set offers more historical informative samples for training retrievers, being beneficial to getting rid of the sampling bias caused by the exposure bias. The Algorithm 2 details the procedure of the cache-augmented inbatch importance resampling.

Algorithm 2: Cache-Augmented Inbatch Importance Resampling (χIR)

Input: Training data $\mathcal{D} = \{(u, i)\}$, Number of epochs T, Cache size C, Hyperparameter λ
Output: Model parameters Θ

1 Initialize the occurrence vector $\boldsymbol{o} = \{0\}_{i=1}^{N}$ and the cache $\mathcal{C}$ with uniformly sampled items;
2 Statistically get item popularity $\mathcal{P} = \{pop(i) | \forall i \in \mathcal{I}\}$ from $\mathcal{D}$;
3 **for** $e = 1, 2, ..., T$ **do**
4 **for** *mini-batch* $B \in \mathcal{D}$ **do**
5 $U = \{u | (u, i) \in B\}$, $I = \{i | (u, i) \in B\}$;
6 Map user feature vectors into embeddings $E_U \in \mathbb{R}^{B \times k}$ based on ϕ_Q;
7 Map item feature vectors into embeddings $E_I \in \mathbb{R}^{B \times k}$ based on ϕ_I;
8 Map feature vectors for items in the cache $\mathcal{C}$ into embeddings $E_c \in \mathbb{R}^{C \times k}$ based on ϕ_I;
9 Calculate the similarity matrix $S = E_U E_I^\top$ and $S' = E_u E_c^\top$;
10 **for** $u \in U$ **do**
11 Resample a set of items $\mathcal{R}_u$ based on $w(i|u) = \frac{\exp(s(u,i) - \log pop(i))}{\sum_{j \in I} \exp(s(u,j) - \log pop(j))}, \forall i \in I$;
12 Resample a set of items $\mathcal{K}_u$ based on $w_c(i|u) = \frac{\exp(s'(u,i) - \log q(i))}{\sum_{k \in \mathcal{C}} \exp(s'(u,k) - \log q(k))}, \forall i \in \mathcal{C}$;
 `// Update the cache and occurrence vector`
13 $\mathcal{O} = \{i | i \in \mathcal{K}_u \cup \mathcal{R}_u, \forall u \in U\}$, $o_i = o_i + \sum_{j \in \mathcal{O}} \mathbb{I}[j = i]$;
14 Sample C items based on $w = o_i$ to update the cache $\mathcal{C}$;
15 Update model parameters based on the objective function, i.e., Eq (4);

3.3 Complexity Analysis

Compared with the inbatch sampling, BIR introduces additional computational cost for each training mini-batch of calculating the resampling weight (Line 9 in Alg 1) and resampling items according to the multinomial distribution (Line 10 in Alg 1), which has a time complexity of $\mathcal{O}(|B| \times |B|)$. Such time complexity is acceptable compared with the computational cost of embedding functions ($\mathcal{O}(\phi_Q)$ and $\mathcal{O}(\phi_I)$) and similarity matrix computation $\mathcal{O}(|B| \times |B| \times k)$. Regarding χIR, it feeds the items of the cache into the item tower (Line 8 in Alg 2), taking the additional cost of embedding them. The time complexity of resampling is the same as that of BIR. Updating the cache and occurrences (Line 13-14 in Alg 2) takes $\mathcal{O}(N)$ time complexity, where N is the number of items.

4 Experiments

The proposed BIR and χIR are then evaluated on five public real-world datasets in recommender retrievers to answer the following questions: (1) Can the inbatch importance resampling outperforms competitive strategies in inbatch sampling? (2) How do the cache-augmented items improve the performance of recommender retrievers?

4.1 Experimental Settings

4.1.1 Dataset

The experiments are conducted on the **Gowalla**, **Amazon**, **Ta-feng**, **Echonest** and **Tmall** datasets [1], as summarized in Table 1. The rated books in Amazon dataset with higher ratings than average ratings are considered as positive ones, while the interactions in other datasets are regarded as positive ones. 80% of items for a user are sampled for training and the other 20% of items constitute the test set. The performance of recommender retrievers is evaluated on the NDCG (Normalization Discounted Cumulative Gain) and RECALL at a cutoff of 10 by default.

[1] Datasets: https://recbole.io/cn/dataset_list.html

Table 1: Statistics of Datasets

Dataset	#User	#Item	#Interaction	Density
Gowalla	29,858	40,988	1,027,464	8.40e-4
Amazon	130,380	128,939	2,415,650	1.44e-4
Ta-feng	19,451	10,480	630,767	3.09e-4
Echonest	217,966	60,654	4,422,471	3.35e-4
Tmall	125,553	58,058	2,064,290	2.83e-4

4.1.2 Baselines

In this work, we investigate the inbatch sampling for retrievers and we compare the proposed BIR and χIR[2] with competitive inbatch-sampling-based methods. All the methods attempt to minimize the log sampled softmax loss, varying in the selected negatives and the proposal distributions. (1) **SSL** utilizes the inbatch samples with the uniform proposal. (2) **SSL-Pop** uses the inbatch items and adopts the item popularity as the proposal distribution. (3) **MNS** [27] refers to the mixed negative sampling, which uses a mixture of batch and uniformly sampled negatives. We set the number of uniform samples B' to be the same as the batch size. (4) **G-Tower** [28] corrects the selection bias in batch sampling by streaming frequency estimation. It records the last number of steps for each item in fixed-length hash arrays and reduces hash collision errors by using multiple hash arrays. In this way, the frequency is estimated as the reciprocal of the average number of steps. We use 5 different hash arrays to estimate. More implementation details can be attached in Appendix.

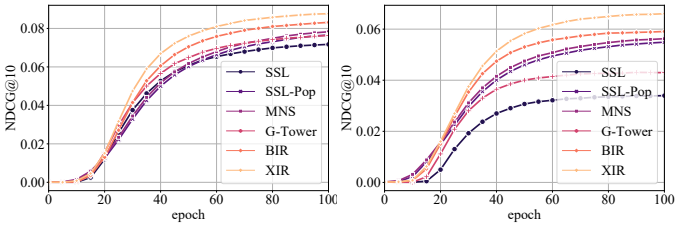
4.2 Performance Comparisons with Baseline Methods

Each method is run for 5 times with different random seeds and we report the average scores with their standard deviations in Table 2. The proposed BIR and χIR considerably outperform the competitive baselines. BIR achieves a relative 2.47%, 6.57%, 6.58%, 2.04% and 5.17% improvements of NDCG@10 on the five datasets respectively while the relative improvements for χIR are 3.81%, 12.23%, 15.24%, 11.73% and 17.12%. Both BIR and χIR show superior performances, indicating the efficient and preferable utilization of the inbatch samples. In addition to correcting the selection bias by the item popularity, BIR and χIR can dynamically select more informative negative samples for different queries within the mini-batch, contributing to better model convergence. According to the importance resampling paradigm, the items with higher similarity scores with the corrected sampling bias have a higher probability of being resampled, where a more accurate estimation of the query-dependent softmax is achieved. In addition, we have the following findings.

Table 2: Comparison with Baselines. $\delta = 1e-4$

	Gowalla	Amazon	Ta-feng	Echonest	Tmall
Method	NDCG@10	NDCG@10	NDCG@10	NDCG@10	NDCG@10
SSL	0.1381±5.5δ	0.0718±3.7δ	0.0437±9.4δ	0.1436±3.1δ	0.0340±3.1δ
SSL-Pop	0.1479±9.1δ	0.0764±3.2δ	0.0625±1.5δ	0.1609±1.9δ	0.0547±5.6δ
MNS	0.1486±8.9δ	0.0781±4.9δ	0.0634±1.4δ	0.1648±2.5δ	0.0561±4.9δ
G-Tower	0.1500±8.6δ	0.0764±4.0δ	0.0496±8.7δ	0.1600±5.1δ	0.0429±2.1δ
BIR	0.1523±7.9δ	0.0833±2.7δ	0.0675±1.3δ	0.1682±3.4δ	0.0590±3.2δ
χIR	0.1543±1.1δ	0.0877±3.4δ	0.0730±1.1δ	0.1842±2.4δ	0.0658±2.2δ
	RECALL@10	RECALL@10	RECALL@10	RECALL@10	RECALL@10
SSL	0.1105±4.1δ	0.0753±4.5δ	0.0375±7.5δ	0.1350±3.6δ	0.0354±3.6δ
SSL-Pop	0.1124±4.4δ	0.0777±4.3δ	0.0506±9.2δ	0.1502±2.4δ	0.0544±6.1δ
MNS	0.1130±4.2δ	0.0796±6.1δ	0.0514±8.5δ	0.1537±2.9δ	0.0558±5.4δ
G-Tower	0.1176±6.8δ	0.0798±5.8δ	0.0415±7.2δ	0.1504±5.1δ	0.0440±1.8δ
BIR	0.1157±4.6δ	0.0848±3.5δ	0.0549±8.6δ	0.1568±3.4δ	0.0588±3.1δ
χIR	0.1169±6.1δ	0.0895±4.1δ	0.0589±6.1δ	0.1703±2.5δ	0.0651±2.8δ

[2]Implementation for BIR and χIR: https://anonymous.4open.science/r/IR-CC7F



(a) Amazon (b) Tmall

Figure 1: NDCG@10 vs. number of epochs on three datasets

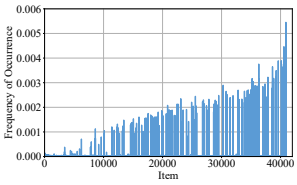


Figure 2: Item occurrence. Items are sorted in descending order of popularity.

Finding 1: Reducing the selection bias within the mini-batch by item popularity is beneficial to reducing the approximation bias from the softmax distribution. SSL, which does not adjust the proposal distribution, performs the worst among the baseline methods in terms of both NDCG and RECALL. Thus, such a method raises inconsistency between the real sampling distribution and the proposal distribution, resulting in poor performance of the sampled softmax loss.

Finding 2: Introducing negative items outside of the mini-batch expands the candidate set, facilitating the learning of retrievers. The result, that MNS outperforms SSL-Pop with an average 1.84% relative improvement of NDCG@10 and χIR outperforms BIR with a 7.12% improvement, demonstrates that the model has a better resolution towards long-tailed items with the additional sampled items.

Finding 3: Caching the frequently sampled items helps improve the retriever performance, indicating that the frequently sampled items are more informative than uniformly sampled items. χIR significantly improves the performance of BIR, where historical informative items have a tendency to remain in the cache with more training.

Furthermore, we investigate the recommendation performance during training epochs with different inbatch-sampling-based methods to verify the effectiveness of the BIR and χIR. The changing curve of NDCG@10 on the Amazon and Tmall datasets are shown in Figure 1. Both BIR and χIR contribute to faster convergence compared to the baselines. The model's NDCG quickly reaches a relatively high level under BIR and χIR, particularly on the Amazon and Tmall datasets, implying that more informative items are selected for training.

4.3 Item Distribution in Cache

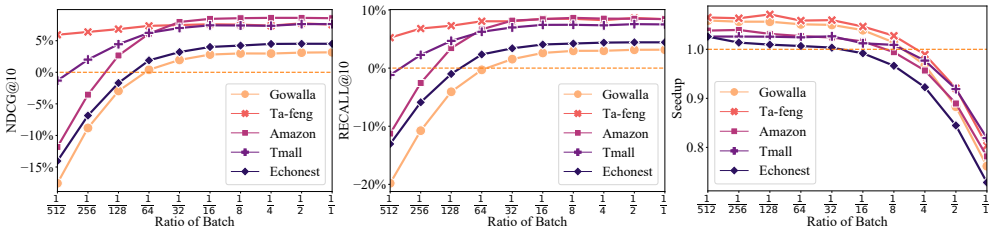
In order to intuitively figure out the item distribution in the cache, experiments are conducted to summarize the occurrence vector, since the cache is updated by sampling based on it. We summarize the occurrence after 100 epochs on the Gowalla dataset and normalize the item occurrence, as shown in Figure 2, where items are sorted by their popularity in descending order.

The distribution of items in the cache performs significantly different from the popularity-based distribution. The huge bias from the popularity-based distribution suggests that the cache can alleviate the long-tailed items becoming more popular according to the corrected sampling probability, i.e., the popularity. Moreover, the unpopular items have a greater chance of being sampled, whereas more informative items are encouraged to be more sampled, rather than treating full items as equivalent.

4.4 Adaptive Query-Dependent Estimation

Compared with MNS and G-Tower, where the items within the mini-batch are shared between user queries, the proposed BIR selects different samples, being able to approximate the softmax distributions for different queries and capture the dynamic changing of softmax distribution during the training process. In this chapter, we conduct experiments by resampling smaller numbers of items from the mini-batch and compare the performances with SSL-Pop. We report the relative improvements in terms of NDCG@10 and RECALL@10 in Figure 3(a) and 3(b).

From the figures, resampling nearly $\frac{1}{64}$ of items in the batch can achieve comparable performances of SSL-Pop with full batch items. Specifically, for Tmall, Amazon and Ta-feng datasets, it takes fewer samples to surpass SSL-Pop. This demonstrates the superiority of distinguished negative samples



(a) NDCG@10 (b) RECALL@10 (c) Gradient Summarized Time

Figure 3: Performances vs. $|\mathcal{R}_u|$

over different user queries, which performs a query-dependent approximation for softmax distribution and benefits the model training.

This series of experiments provides the potential for more efficient training with much fewer samples. Thus, we further compare the running time of summarizing the gradient under various sizes of the resampled item set in Figure 3(c). We run for 10 epochs for each setting and report the average time compared with SSL-Pop. Since data IO and feature embedding take considerably computational cost, we just compare the calculation on the gradient. As fewer items are selected, the summarized gradient receives fewer calculations.

4.5 Introducing Features into Item Tower

In online recommender retrievers, two tower models with deep networks offer the capacity of capturing complicated features. Thus, we further encode the categorical features *SellerID* and *CateID* in the Tmall dataset and build the item tower as two fully-connected layers with 32 and 16 units, using the Relu as the active function. As shown in Table 4, BIR and χIR outperform SSL-Pop, achieving a relative 3.06% and 4.87% improvements in terms of NDCG@10 respectively. This indicates the effectiveness of the proposed importance resampling based methods given more features, implying the possibility of deploying deeper retrieval towers.

Figure 4: Performances on Tmall

	N@10	R@10	N@20	R@20
SSL	0.0311	0.0329	0.0524	0.0568
SSL-Pop	0.0720	0.0712	0.0999	0.1022
BIR	0.0742	0.0735	0.1032	0.1057
χIR	0.0755	0.0750	0.1067	0.1099

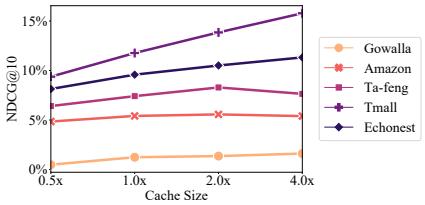


Figure 5: Performances vs. cache size

4.6 Effect of Cache Size

To figure out the influence of the cache size, we conduct experiments with various sizes, including 0.5x, 1x, 2x and 4x of the batch size (2048). We report the relative improvements compared with BIR in Figure 5. As the batch size gets larger, more items participate in the approximation of softmax, achieving a more accurate estimation and a better performance of recommendation quality.

5 Conclusion

In this paper, we propose a cache-augmented inbatch importance resampling for training recommender retrievers, which achieves a more accurate estimation of softmax for different queries using the inbatch items. The proposed BIR resamples items based on the similarity scores with popularity-based debias and thus different queries are trained with different sampled items. χIR is further proposed to encourage historical informative items to be more trained, which augments the inbatch items and thus helps retriever training. Experiments are finally conducted on five real-world datasets to validate the effectiveness of the proposed BIR and χIR.

References

[1] Yu Bai, Sally Goldman, and Li Zhang. Tapas: Two-pass approximate adaptive sampling for softmax. *arXiv preprint arXiv:1707.03073*, 2017.

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

[3] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[4] Guy Blanc and Steffen Rendle. Adaptive sampled softmax with kernel based sampling. In *International Conference on Machine Learning*, pages 590–599. PMLR, 2018.

[5] Qi Chen, Bing Zhao, Haidong Wang, Mingqin Li, Chuanjie Liu, Zengzhong Li, Mao Yang, and Jingdong Wang. Spann: Highly-efficient billion-scale approximate nearest neighborhood search. *Advances in Neural Information Processing Systems*, 34:5199–5212, 2021.

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

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

[8] Jingtao Ding, Yuhan Quan, Xiangnan He, Yong Li, and Depeng Jin. Reinforced negative sampling for recommendation with exposure data. In *IJCAI*, pages 2230–2236, 2019.

[9] Jingtao Ding, Yuhan Quan, Quanming Yao, Yong Li, and Depeng Jin. Simplify and robustify negative sampling for implicit collaborative filtering. *arXiv preprint arXiv:2009.03376*, 2020.

[10] Carlos A Gomez-Uribe and Neil Hunt. The netflix recommender system: Algorithms, business value, and innovation. *ACM Transactions on Management Information Systems (TMIS)*, 6(4):1–19, 2015.

[11] Ruiqi Guo, Philip Sun, Erik Lindgren, Quan Geng, David Simcha, Felix Chern, and Sanjiv Kumar. Accelerating large-scale inference with anisotropic vector quantization. In *International Conference on Machine Learning*, pages 3887–3896. PMLR, 2020.

[12] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. Session-based recommendations with recurrent neural networks. *arXiv preprint arXiv:1511.06939*, 2015.

[13] Jui-Ting Huang, Ashish Sharma, Shuying Sun, Li Xia, David Zhang, Philip Pronin, Janani Padmanabhan, Giuseppe Ottaviano, and Linjun Yang. Embedding-based retrieval in facebook search. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2553–2561, 2020.

[14] Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry Heck. Learning deep structured semantic models for web search using clickthrough data. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, pages 2333–2338, 2013.

[15] Sébastien Jean, Kyunghyun Cho, Roland Memisevic, and Yoshua Bengio. On using very large target vocabulary for neural machine translation. *arXiv preprint arXiv:1412.2007*, 2014.

[16] Binbin Jin, Defu Lian, Zheng Liu, Qi Liu, Jianhui Ma, Xing Xie, and Enhong Chen. Sampling-decomposable generative adversarial recommender. *Advances in Neural Information Processing Systems*, 33:22629–22639, 2020.

[17] Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with gpus. *IEEE Transactions on Big Data*, 7(3):535–547, 2019.

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

[19] Erik Lindgren, Sashank Reddi, Ruiqi Guo, and Sanjiv Kumar. Efficient training of retrieval models using negative cache. *Advances in Neural Information Processing Systems*, 34, 2021.

[20] Dae Hoon Park and Yi Chang. Adversarial sampling and training for semi-supervised information retrieval. In *The World Wide Web Conference*, pages 1443–1453, 2019.

[21] Ankit Singh Rawat, Jiecao Chen, Felix Xinnan X Yu, Ananda Theertha Suresh, and Sanjiv Kumar. Sampled softmax with random fourier features. *Advances in Neural Information Processing Systems*, 32, 2019.

[22] Steffen Rendle. Item recommendation from implicit feedback. In *Recommender Systems Handbook*, pages 143–171. Springer, 2022.

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

[24] Jinpeng Wang, Jieming Zhu, and Xiuqiang He. 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*, pages 1632–1636, 2021.

[25] Jun Wang, Lantao Yu, Weinan Zhang, Yu Gong, Yinghui Xu, Benyou Wang, Peng Zhang, and Dell Zhang. Irgan: A minimax game for unifying generative and discriminative information retrieval models. In *Proceedings of the 40th International ACM SIGIR conference on Research and Development in Information Retrieval*, pages 515–524. ACM, 2017.

[26] Jason Weston, Samy Bengio, and Nicolas Usunier. Wsabie: Scaling up to large vocabulary image annotation. In *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.

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

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

[29] Weinan Zhang, Tianqi Chen, Jun Wang, and Yong Yu. 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*, pages 785–788, 2013.

[30] Chang Zhou, Jianxin Ma, Jianwei Zhang, Jingren Zhou, and Hongxia Yang. Contrastive learning for debiased candidate generation in large-scale recommender systems. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 3985–3995, 2021.

[31] Qiannan Zhu, Haobo Zhang, Qing He, and Zhicheng Dou. A gain-tuning dynamic negative sampler for recommendation. In *Proceedings of the ACM Web Conference 2022*, pages 277–285, 2022.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] See the abstract and the introduction.
   (b) Did you describe the limitations of your work? [Yes] We propose the adaptive sampler to under the paradigm of recommender retrievers and inbatch sampling. There still exists bias from the ideal softmax distribution in this case.
   (c) Did you discuss any potential negative societal impacts of your work? [No] Our work does not have any negative societal impacts
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes] We states the assumptions for theoretical results.
   (b) Did you include complete proofs of all theoretical results? [Yes] All the proofs can be attached in Appendix.
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] The links of datasets and codes are accessible in Section 4.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] Data splits can be referred in Section 4. Details about hyperparameters can be attached in Appendix. The provided codes covers the whole process and all baselines.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] Each algorithm is run for 5 times and we report the error bar(standard deviation) in Table 2.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] We provide the empirical environment in 'Implementation details' in Appendix.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] We cite the resources in Section 4
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [No]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]