Learning Recommenders for Implicit Feedback with Importance Resampling

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

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

Binbin Jin
Huawei Cloud Computing Technologies Co., Ltd.
jinbinbin1@huawei.com

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

Enhong Chen
University of Science and Technology of China
cheneh@ustc.edu.cn

ABSTRACT

Recommendation is prevalently studied for implicit feedback recently, but it seriously suffers from the lack of negative samples, which has a significant impact on the training of recommendation models. Existing negative sampling is based on the static or adaptive probability distributions. Sampling from the adaptive probability receives more attention, since it tends to generate more hard examples, to make recommender training faster to converge. However, item sampling becomes much more time-consuming particularly for complex recommendation models. In this paper, we propose an Adaptive Sampling method based on Importance Resampling (AdaSIR for short), which is not only almost equally efficient and accurate for any recommender models, but also can robustly accommodate arbitrary proposal distributions. More concretely, AdaSIR maintains a contextualized sample pool of fixed-size with importance resampling, from which items are only uniformly sampled. Such a simple sampling method can be proved to provide approximately accurate adaptive sampling under some conditions. The sample pool plays two extra important roles in (1) reusing historical hard samples with certain probabilities; (2) estimating the rank of positive samples for weighting, such that recommender training can concentrate more on difficult positive samples. Extensive empirical experiments demonstrate that AdaSIR outperforms state-of-the-art methods in terms of sampling efficiency and effectiveness.

CCS CONCEPTS

• **Information systems → Recommender systems**.

*This work was done when the author Jin Chen was at University of Science and Technology of China for intern.
†Corresponding authors

WWW '22, April 25–29, 2022, Virtual Event, Lyon, France.

ACM ISBN 978-1-4503-9096-5/22/04…$15.00
https://doi.org/10.1145/3485447.3512075

ACM Reference Format:
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 (WWW '22), April 25–29, 2022, Virtual Event, Lyon, France.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3485447.3512075

1 INTRODUCTION

Recommender systems play a crucial role in addressing information overload, and have created considerable business revenue for many high-tech companies. Since implicit feedback, such as the behavior of click and purchase, is more ubiquitous and more easily accessible, recommendation is more prevalently investigated for implicit feedback [19, 29]. However, the lack of explicit negative samples becomes an obstacle for training recommendation models. Given the large number of items, explicit negative feedbacks are extremely sparse. If we only consider the real negative samples, it will cause biased selection [10]. One simple method is to treat all the unobserved samples as negative [2, 5, 6, 24, 26, 41] but assign them low confidence being negative. The learning efficiency can be only improved by de-contextualizing the negative confidence and only restricting the use of square loss and specified models such as matrix factorization [15, 19, 22]. This significantly restricts the representation capacity of recommendation models. Sampling a few representative items from unobserved data (i.e., negative sampling) is an efficient approach to accommodate any complex models and loss functions [7–9, 14, 17, 25, 32, 34, 36, 37, 42].

Many negative sampling methods are usually based on the static distributions, such as the uniform distribution and popularity-based distribution [7, 17, 32]. Although they only take $O(1)$ time to sample items, they are not adaptive to the update of recommendation models. As a consequence, the sampled items are becoming easier to distinguish from positive samples, slowing down the convergence of recommender training [31]. More effective samplers adapt both to context and recommender models during the training [17, 25, 31, 42], so that high-scored items by recommenders, which can contribute more to the gradients, are more likely to be drawn. Since scores can be positive or negative, the sampling probability distribution is usually defined by a softmax of recommendation scores over items [30, 34]. However, it is time-consuming to directly sample items from the softmax distribution, which is

contextualized and dynamically changing with the training of recommenders. Assuming the scores are computed by inner product, sampling from the softmax distribution can be approximated by rank mixture [31] or a quadratic-kernel based distribution [4]. However, the latter distribution does not well approximate the softmax distribution when the scores are negative, and also suffers from a large memory footprint due to the feature mapping of the quadratic kernel. More seriously, both methods are difficult to adapt to nonlinear scores, such as neural recommendation models [14].

One promising efficient approach to approximate the softmax of nonlinear scores is the two-pass sampling [1, 11, 42], which first samples a fixed size of items, i.e., sample pool, from the simple static distribution, and then selects the item with the maximal recommendation score from the sample pool. However, in these samplers, many questions are remained to answer. First, how do these samplers approximate the softmax distribution? Second, what is the probability of each item being sampled? Finally, how are other efficient proposals accommodated to the two-pass sampling?

To this end, we propose an Adaptive Sampling method based on Importance Resampling (AdaSIR for short), which is not only almost equally accurate for any recommender model, but also can robustly accommodate arbitrary proposal distributions. More concretely, AdaSIR maintains a fixed-size sample pool for each context with importance resampling based on recommendation scores. The sample pool is updated after each training epoch according to the latest model, so that highly-informative samples are likely to be kept for reuse in the next epoch of training. When training, items are only uniformly sampled from the sample pool to pair with positive samples. In spite of simplicity, AdaSIR can be proved to provide approximately accurate adaptive sampling under some conditions. Moreover, when the proposal is the uniform distribution, it is possible to reuse the sample pool for estimating the rank of positive samples. This could benefit to the training of recommenders, since it could concentrate more on difficult positive samples by assigning larger weights to them. According to time complexity analysis, AdaSIR only takes linear time w.r.t. pool size to sample an item, which is much efficient for any complex recommender model. AdaSIR is then evaluated with three real-world datasets, whose results demonstrate the superiority of AdaSIR in terms of sampling efficiency and effectiveness.

To summarize, the main contributions are three-fold:

- We propose an efficient two-pass sampling approach for implicit feedback in recommendation system, where the importance resampling is exploited to approximate the softmax distribution.
- We design the sampling pool with fixed size, where high informative historical items would be reused and the rank of positive samples are efficiently accessible.
- Experiments conducted on the three public datasets demonstrate the efficiency and effectiveness of the proposed sampling method, showing superior performance over the state-of-the-art samplers.

2 PRELIMINARIES

2.1 Recommendation for Implicit Feedback

Recommender for implicit feedback aims to learn a scoring function $r(c, i|\boldsymbol{\theta})$ to predict scores of unobserved items in a context c and recommend the top-ranked items. The predicted score between a context $c \in C$ and an item $i \in I$ reflects the preference for the item, where a larger value indicates the higher preference. In the personalized ranking algorithms, given the observed data $\mathcal{D}$, the objective function can be formulated as:

$$\min_{\boldsymbol{\theta}} \sum_{(c,i)\in\mathcal{D}} \mathbb{E}_{j\sim P_{ns}(j|c)} \left[-\ln \sigma\left(r(c,i|\boldsymbol{\theta}) - r(c,j|\boldsymbol{\theta})\right)\right] + \lambda \|\boldsymbol{\theta}\|^2,$$

where $\boldsymbol{\theta}$ denotes the model parameters, $\sigma(x) = \frac{1}{1+\exp(-x)}$ is the sigmoid function, λ is a coefficient for the ℓ_2 regularization. P_{ns} denotes the proposal distribution, from which negative items are drawn. Minimizing the loss function encourages the positive items to be ranked higher than others. In the following, $r(c, i|\boldsymbol{\theta})$ is denoted as r_{ci} and $r(c, i|\boldsymbol{\theta}) - r(c, j|\boldsymbol{\theta})$ is denoted as r_{cij} for short.

2.2 Negative Samplers in Recommenders

Static Sampler. The most widely-used sampling distribution is the uniform distribution [32]. It enjoys high-efficient sampling but all items are equally probable to be drawn. When the model is better trained, the sampled items are easier to be distinguished from positive samples, i.e., $\sigma(r_{cij}) \to 1$, so that they contribute little to the gradients, i.e., $\left(1 - \sigma(r_{cij})\right) \frac{\partial r_{cij}}{\partial \boldsymbol{\theta}} \to \mathbf{0}$. This would lead to slow convergence of recommender training. The popularity distribution [3] is another commonly-used static sampling distribution, where popular items are more likely to be sampled. This is reasonable to a certain extent from a crowd's perspective because the popular items are more easily exposed to users so that they are not preferred by users if users do not have interactions with them. However, the sampler can not be personalized, contextualized, and more importantly, adaptive to model updating. Therefore, in spite of sampling efficiency, they are far from the ideal negative sampler.

Adaptive Sampler. The adaptive samplers, which assume the high-scored items by recommenders should be more likely to be picked, show better performance in both recommendation and other tasks. Since models' outputs are not constrained, the probability of an item i being sampled given a user is usually defined as [3, 25]:

$$P^{\star}_{ns}(i|c) = \frac{\exp(r_{ci})}{\sum_{j\in I} \exp(r_{cj})}.$$

However, sampling from $P^{\star}_{ns}(i|c)$ is extremely time-consuming due to personalization and frequent model updating. Softmax sampling can be approximated by exploiting the grid or tree structure [4, 23, 28], achieving $O(\sqrt{N})$ or $O(\log N)$ time sampling if N is the number of items. The recent work utilizes the quadratic kernel [4] for approximation such that the softmax probability can be efficiently computed through a balanced binary tree in a divide and conquer way. However, these methods are challenging to be extended to nonlinear scores instead of inner-product scores, such as the ones used in GMF, MLP [14]. This problem could be mitigated by the efficient two-pass sampler [1, 42], where a collection of items are first sampled from all candidate items according to a static distribution and one item with the largest scores is then selected from the collection as a negative one.

3 LEARNING RECOMMENDERS WITH IMPORTANCE RESAMPLING

As aforementioned, the two-pass sampler is efficient and effective for any complex recommendation model. However, it is unclear

what is the underlying sampling distribution and how it approximates the softmax distribution. Moreover, the sampler in the first stage is usually the uniform distribution, it is also unclear how they can accommodate other static or simple samplers. To answer these questions, we propose an Adaptive Sampling method based on Importance Resampling (AdaSIR for short). Technical details will be elaborated below.

3.1 Generating Contextualized Sample Pools with Importance Resampling

The key step of AdaSIR is to generate a fixed-size sample pool for each context c, from which a few negative items can be drawn for each training step. Initially, the items in the pool, denoted by $\mathcal{K}_c$, are sampled from a static distribution Q. In order to make these samples follow the ideal softmax distribution, items in the pool should be resampled to form the sample pool $\mathcal{R}_c$. According to the importance sampling theory, the resampling probability should follow the categorical distribution over a size-$|\mathcal{K}_c|$ sample space, where each sampled item i is resampled with a probability $w(i|c) = \frac{\exp(r_{ci} - \log Q(i))}{\sum_{j \in \mathcal{K}_c} \exp(r_{cj} - \log Q(j))}$. Note that their sizes are set to be the same for simplicity (i.e. $|\mathcal{K}_c| = |\mathcal{R}_c|$). When Q is the uniform distribution, this means that the sampled items with larger scores r_{ci} are more likely to be resampled. Before resampling, items rarely appear more than one time, but after resampling, items with larger scores r_{ci} probably appear multiple times. The underlying frequency distribution can approximate the softmax distribution based on the following theorem.

Theorem 3.1. *When $|\mathcal{K}_c| \to \infty$, items in the sample pool $\mathcal{R}_c$ follow the softmax distribution. That is, for any given set of items $\mathcal{S} \subseteq \mathcal{I}$, assuming these items totally appear $N_{\mathcal{S}}$ times in the pool $\mathcal{R}_c$, then when $|\mathcal{K}_c| \to \infty$, we have $\frac{N_{\mathcal{S}}}{|\mathcal{K}_c|} = P^{\star}_{ns}(S|c)$, where $P^{\star}_{ns}(\mathcal{S}|c) = \sum_i P^{\star}_{ns}(i|c)$.*

Proof.

$$\begin{aligned}
\frac{N_{\mathcal{S}}}{|\mathcal{K}_c|} \approx \sum_{i \in \mathcal{K}_c \cap \mathcal{S}} w(i|c) &= \sum_{i \in \mathcal{K}_c \cap \mathcal{S}} \frac{\exp(r_{ci} - \log Q(i))}{\sum_{j \in \mathcal{K}_c} \exp(r_{cj} - \log Q(j))} \\
&= \frac{\frac{1}{|\mathcal{K}_c|} \sum_{i \in K_c} \mathbb{I}[i \in \mathcal{S}] \exp(r_{ci} - \log Q(i))}{\frac{1}{|\mathcal{K}_c|} \sum_{j \in \mathcal{K}_c} \exp(r_{cj} - \log Q(j))} \\
&\approx \frac{\mathbb{E}_{i \sim Q}\left[\mathbb{I}[i \in \mathcal{S}] \exp(r_{ci} - \log Q(i))\right]}{\mathbb{E}_{j \sim Q}[\exp(r_{cj} - \log Q(j))]} \\
&= \frac{\sum_{i \in \mathcal{S}} \exp(r_{ci})}{\sum_{j \in \mathcal{I}} \exp(r_{cj})} = P^{\star}_{ns}(\mathcal{S}|c)
\end{aligned}$$

where $\mathbb{I}[\cdot]$ is an indicator function which equals 1 if the condition is true and equals 0 otherwise. The approximately equal sign $\approx$ turns to the equal sign $=$ when $|\mathcal{K}_c| \to \infty$. □

Note that when the proposal Q becomes more informative, the divergence between them could be more reduced according to the theorem. When the sample pool is ready, a set of negative items, denoted by $\mathcal{J}_{ci}$, are uniformly sampled from $\mathcal{R}_c$ for each positive pair (c, i), each of which is assigned with the resampling weight $w(\cdot|c)$, indicating the sampling probability of the two-pass sampler. The uniform sampling from $\mathcal{R}_c$ can be considered equivalent to the softmax sampling according to Lemma 3.2.

Lemma 3.2. *Assume a list of items Λ are i.i.d. sampled from a categorical distribution P over a size-K sample spaces, which are denoted by $\mathcal{Y} = \{y_1, \cdots, y_K\}$. When $|\Lambda| \to \infty$, uniformly sampling an item from Λ is equivalent to sampling an item from $\mathcal{Y}$ according to the distribution P.*

Proof. Denote by N_k the number of times y_k appear in Λ such that $\sum_{k=1}^{K} N_k = |\Lambda|$. Let Y be a random variable following the distribution P, then obviously we have $\lim_{|\Lambda| \to \infty} \frac{N_k}{|\Lambda|} = P(Y = y_k)$. When uniformly sampling from Λ, an item k is picked with a probability $\frac{N_k}{|\Lambda|}$, which equals to $P(Y = y_k)$ as proved. □

Finally, based on the self-normalized importance sampling [25], the objective function of the personalized ranking is formulated as:

$$\begin{aligned}
\min_{\boldsymbol{\theta}} \sum_{(c,i) \in \mathcal{D}} \sum_{j \in \mathcal{J}_{ci}} -\hat{w}(j|c) \ln \sigma\left(\hat{r}_{ci} - \hat{r}_{cj}\right) + \lambda \|\boldsymbol{\theta}\|^2, \\
\hat{w}(i|c) = \frac{\exp(\hat{r}_{ci} - \log w(i|c))}{\sum_{j \in \mathcal{J}_{ci}} \exp(\hat{r}_{cj} - \log w(j|c))},
\end{aligned} \tag{1}$$

where $\hat{r}_{c\cdot}$ denotes the score outputted by the current model while $w(i|c)$ is computed with the score $\hat{r}'_{ci}$ by the model in the immediately preceding epoch. Assuming Q is the uniform distribution, $\hat{w}(i|c) \propto \exp(\hat{r}_{ci} - \log w(i|c)) \propto \exp(\hat{r}_{ci} - \hat{r}'_{ci})$. Therefore, items with large increase of scores from last model will contribute more to the loss function and thus the gradients.

3.2 Reusing Informative Historical Samples

Considering the practical situation that the size of sample pool is finite, the proposed samplers may deviate much from the softmax distribution. Although such a bias can be reduced via exploiting all historical sampled items, it suffers from a large memory footprint. Since static sampling can generate diverse items, only informative items (i.e., high-scored items by recommenders) are more valuable to keep. Noting that more informative items appear multiple times, we can update the sample pool for each context by uniformly sampling items from the merging set of the historical pool $\mathcal{R}^t_c$ and the current pool $\mathcal{R}^{t+1}_c$. This simple idea can be supported by the following theoretical results.

Corollary 3.3. *Assume the model does not update a lot between two epochs such that the softmax distribution almost keeps the same. The updated pool of samples are approximately drawn from the softmax distribution.*

Proof. According to theorem 3.1, the items in both $\mathcal{R}^t_c$ and $\mathcal{R}^{t+1}_c$ are considered i.i.d. sampled from the softmax distribution when the pool size is sufficiently large. It directly follows that the union of $\mathcal{R}^t_c$ and $\mathcal{R}^{t+1}_c$ also reach the same results. According to Lemma 3.2, uniformly sampling an item from the union of $\mathcal{R}^t_c$ and $\mathcal{R}^{t+1}_c$ is equivalent to sampling it from the softmax distribution. □

According to the corollary, the proposed pool updating approach can mitigate the divergence caused by limited pool size. This is also different from previous two-pass samplers, which simply fill the sample pool with newly sampled items. The whole procedure of the proposed sampler is detailed in Algorithm 1. It is worth mentioning that the proposed sampler can be fully implemented in the tensor-based deep learning library, remarkably accelerating

Algorithm 1: AdaSIR: Adaptive Sampling method based on Importance Resampling

Input: Traindata $\mathcal{D} = \{(c, i)\}$, Context set C, Epochs E
Output: Model parameters $\boldsymbol{\theta}$

Initialize model parameters $\boldsymbol{\theta}$, fixed-size sampling pool for each context $\mathcal{R}_c = \emptyset$;
for $e = 1, 2, .., E$ **do**
  // Update the sampling pool for each context
  foreach $c \in C$ **do**
    Sample a set of items $\mathcal{K}_c$ following a simple proposal distribution Q;
    Calculate the resampling weight $w_i = \frac{\exp(r_{ci} - \log Q(i))}{\sum_{j \in \mathcal{K}_c} \exp(r_{cj} - \log Q(j))}, \forall i \in \mathcal{K}_c$;
    Resample a set of items $\mathcal{R}'_c$ from $\mathcal{K}_c$ based on w_i with replacement;
    // Reuse historical samples
    if $\mathcal{R}_c = \emptyset$ **then**
      $\mathcal{R}_c \leftarrow \{(i, w_i) | i \in \mathcal{R}'_c\}$
    else
      $\mathcal{R}_c \leftarrow$ uniformly sample items from $\mathcal{R}_c \cup \{(i, w_i) | i \in \mathcal{R}'_c\}$
  // Sample negative items and train the model
  foreach $(c, i) \in \mathcal{D}$ **do**
    Uniformly sample a set of negative samples $\mathcal{J}_{ci}$ from $\mathcal{R}_c$;
    Update model parameters based on the objective function $\mathcal{L}$ in Eq (1);
return $\boldsymbol{\theta}$

sampling by GPU computation and reducing the cost of data transmission between GPU and CPU. This is also a significant advantage over index-based approximate sampling.

3.3 Adaptive Rank Estimation

Different positive items can make different contributions to gradients in a mini-batch. Generally speaking, the bottom-ranked positive items can contribute more than top-ranked ones. As a consequence, it can benefit from incorporating the rank of positive items. However, it is challenging to estimate the rank of positive items due to the large number of candidates. WARP [38] is a popular method to approximate the rank based on the geometric distribution. Particularly, it counts the trials V of uniform sampling until a violator occurs, and estimates the rank of the positive item as $rank(c, i) = \lfloor \frac{N-1}{V} \rfloor + 1$ where $N = |\mathcal{I}|$ denotes the number of items. However, when the recommendation models are better trained, it takes more trials to find a violator for top-ranked items. Therefore, a maximum value of V is set to ensure the efficiency but results in an inaccurate estimation of rank. Moreover, such a method can not make full use of the sample pool and thus efficiently integrate with the developed proposals.

Therefore, in this section, we consider how to estimate the rank of positive items given the sample pool. Suppose items in $\mathcal{K}_c$ are drawn from the uniform distribution, the rank of the positive item can be approximately estimated by the proportion of higher-scored items than positive items in $\mathcal{K}_c$. Formally, given a positive item i, its rank at the context c is defined as $rank(c, i) = \sum_{j \in \mathcal{I}} \mathbb{I}[\hat{r}_{cj} > \hat{r}_{ci}] + 1$. In the first stage of uniform sampling, higher-scored items than item i are picked with a probability $p = \frac{rank(c,i)-1}{N-1}$. Then the random variable x which counts the number of higher-scored items than the item i in the sample pool follows a binomial distribution of parameter $|\mathcal{K}_c|$ and p. Then $\mathbb{E}[x] = |\mathcal{K}_c| \frac{rank(c,i)-1}{N-1}$. This suggests that the rank of item i can be approximated by

$$rank(c, i) \approx \lfloor \frac{(N-1) \times \sum_{j \in \mathcal{K}_c} \mathbb{I}[\hat{r}_{cj} > \hat{r}_{ci}]}{|\mathcal{K}_c|} \rfloor + 1, \tag{2}$$

where $\mathbb{I}[\cdot]$ is an indicator function which equals 1 if the condition is true and equals 0 otherwise. Compared to WARP whose approximation of rank is based on the expectation of the geometric distribution, our proposed approximation is based on the expectation of the binomial distribution. Therefore, it is possible to derive error bound of rank estimation.

Theorem 3.4. *Suppose $x = \sum_{j \in \mathcal{K}_c} \mathbb{I}[\hat{r}_{cj} > \hat{r}_{ci}]$ is a random variable which counts the number of higher-scored items than the item i in the sample pool. $\overline{x}$ is the empirical mean over n trials which is defined as $\overline{x} = \frac{1}{n} \sum_{k=1}^{n} x^{(k)}$. Then, the deviation between its empirical mean and the expected value has an upper bound as follows:*

$$P(|\overline{x} - \mathbb{E}[\overline{x}]| \geq t) \leq \exp(-\frac{2nt^2}{|\mathcal{K}_c|^2}), \quad s.t. \quad t \geq 0.$$

Proof. $x^{(1)}, x^{(2)}, \cdots, x^{(n)}$ i.i.d. follows Binomial($|K_c|, p$), where $p = \frac{rank(c,i)-1}{N-1}$ and $x^{(k)}$ is strictly bounded by $[0, \mathcal{K}_c]$. The result is then derived by directly applying the Hoeffding's inequality [16]. □

The important goal is to estimate the $rank(c, i)$, so we can transform the inequality with the probability p, that is, by setting $t = |\mathcal{K}_c|\varepsilon$ and defining $\overline{p} = \frac{\overline{x}}{|\mathcal{K}_c|}$

$$P(|\overline{p} - p| \geq \varepsilon) \leq 2\exp(-2n\varepsilon^2), \quad s.t. \quad \varepsilon \geq 0.$$

The results show that given the fixed pool size $|\mathcal{K}_c|$, the error of approximating p with $\overline{p}$ can be exponentially reduced by generating more sample pools for each context. This can be considered as equivalent to increasing the size of the sample pool.

We also provide another explanation for this inequality by applying the Hoeffding's inequality for Bernoulli random variables. This is because the Binomial distribution represents the number of successes (higher-scored than the item i) in $|\mathcal{K}_c|$ independent Bernoulli trials. Then we have

$$P(|\frac{x}{|\mathcal{K}_c|} - p| \geq \varepsilon) \leq 2\exp(-2|\mathcal{K}_c|\varepsilon^2), \quad s.t. \quad \varepsilon \geq 0.$$

From this inequality, we directly observe with the increasing size of the sample pool, the approximate error can decay exponentially.

According to the theorem, we can observe that setting a larger pool size can reduce the error bound so that the estimation of the rank can be more accurate. Following WARP, the adaptive weight with respect to the rank is defined as $J(k) = \sum_k \frac{1}{k}$. This function

pays more attention on the top of item list[1], such that the bottom-ranked positive items are assigned with more importance. Finally, the objective function is then formulated as follows:

$$\min_{\boldsymbol{\theta}} \sum_{(c,i)\in\mathcal{D}} J(rank(c,i)) \sum_{j\in\mathcal{J}_{ci}} -\hat{w}(j|c) \ln \sigma\left(\hat{r}_{ci} - \hat{r}_{cj}\right) + \lambda\|\boldsymbol{\theta}\|^2 \quad (3)$$

To sum up, AdaSIR provides a good approximation to the softmax distribution and can accommodate any proposal distribution. In addition, AdaSIR has two extra advantages. On one hand, instead of discarding the previously sampled items in the classic two-pass sampler, AdaSIR reuses more informative items to improve recommender training. On the other hand, the sampler can efficiently estimate the rank of positive items such that the bottom-ranked items are more emphasized during the training process.

3.4 Complexity Analysis

Time Complexity. The sampling and training phases dominate the majority of running time. As for the sampling procedure, generating the sampling pool for each context c takes $O(|\mathcal{K}_c| \cdot (T_Q + T_r))$, where T_Q is the time of sampling an item from the proposal distribution, particularly $O(1)$ for uniform sampling and popularity-based sampling, T_r denotes the time for calculating the score function. Another important step for updating the sampling pool takes $O(|\mathcal{R}_c|)$. For each training pair (c, i), it takes $O(|\mathcal{J}_{ci}|)$ to sample items. Thus, the time complexity for sampling procedure during each training epoch is $O(M \cdot |\mathcal{K}_c| \cdot (T_Q + T_r) + |\mathcal{D}| \cdot |\mathcal{J}_{ci}|)$, where M is the number of users and $|\mathcal{D}|$ denotes the number of user-item pairs for training. As for the training procedure, it takes $O(|\mathcal{D}| \cdot |\mathcal{J}_{ci}| \cdot T_r)$ for inference. Since $|\mathcal{D}| >> M$, the time complexity of sampling is less than that of training. The ranking estimation takes $O(|\mathcal{D}| \cdot |\mathcal{K}_c| \cdot T_r)$, which is also greater than the sampling complexity.

Space Complexity. Apart from the dataset and model parameters, the main memory cost is the storage of the sampling pool which takes $O(M \cdot |\mathcal{R}_c|)$ to save the sampling pool and their resampling weights respectively. It also takes $O(M \cdot |\mathcal{K}_c|)$ to store $\mathcal{K}_c$ to estimate the ranking scores of positive items. Totally, the space complexity is $O(M \cdot (|\mathcal{K}_c| + |\mathcal{R}_c|))$.

4 EXPERIMENT

The proposed AdaSIR is evaluated on three public sparse datasets in recommender systems to figure out the following questions: (1) Can the proposed AdaSIR outperform the competitive recommendation models? (2) How efficiently the AdaSIR sample items? (3) What is the accuracy of estimating the softmax distribution with AdaSIR?

4.1 Dataset

Three real-world datasets are utilized, as shown in Table 1, to validate the proposed sampler. They vary in scale and sparsity. The **Gowalla**[2] dataset is the collection of user check-in histories. The **Yelp**[3] dataset records the point of interest. The **Amazon**[4] dataset includes the set of ratings for Amazon books and is sparser than the other two datasets. Considering the ratings in the Amazon dataset are integers ranging from 1 to 5, the books with ratings above 4 are regarded as the positive ones. We filter out the users and items with less than 5 interactions to guarantee the test. For each user, 80% of items are randomly selected as the training set and the left 20% of items are the test set. 10% ratings of the training data are collected for validation. The models are trained on the training set and evaluated on the test set.

Table 1: Summary of Datasets

Dataset	#User	#Item	#Train	#Test	Sparsity
Gowalla	29,858	40,988	822,358	205,106	99.9160%
Yelp	77,277	45,638	1,684,846	419,049	99.9403%
Amazon	130,380	128,939	1,934,404	481,246	99.9856%

As for the metrics, we focus on the quality of recommended item list and select the widely-used metrics of ranking evaluation, NDCG and Recall at a cutoff k. A higher top-ranked item rewards a higher value of NDCG. The Recall is the fraction of positive items in the top-k item list over the whole positive items in the test data. The cutoff k is set to 50 by default.

4.2 Baselines

To verify the effectiveness of the proposed sampler, we choose the following competitive algorithms. Unless otherwise specified, we exploit matrix factorization as the recommender models.

- **BPR** [32], the Bayesian personalized ranking, is the classical personalized ranking model for implicit feedback. It utilizes the pair-wise loss and samples negative items with the static uniform distribution.
- **AOBPR** [31] improves the sampling method in BPR through the proposed adaptive oversampling. We use the version in librec[5], and λ for the geometric distribution is set to 500.
- **WARP** [38] exploits the weighted approximate-rank pair-wise loss for implicit feedback. Given a positive item, it counts the uniform sampling over all items util a negative item is collected to estimate the rank. We use the public implementation in LightFM[6].
- **IRGAN** [34] generates the negative data with the GAN style model. It uses the generative network that generates items for the user and sends the items into a discriminative network to judge whether the sample is from the real data. We utilize the code released by the authors[7].
- **DNS** [42] is the dynamic negative sampler which samples a set of items uniformly and chooses the item with the highest score according to the current model parameters to update the model. The candidate size is set to 10 by default.
- **SRNS** [11] is the state-of-the-art sampling method to select informative negative items by considering the variance of the predicted score during the recent epochs. We use the released code[8].
- **PRIS** [25] utilizes the importance sampling to the pair-wise ranking loss for personalized ranking and assigns the sampling weight to the sampled items. We adopt the uniform and popularity-based distribution as the proposal distribution for training models, denoted as PRIS(U) and PRIS(P) respectively.

[1] CML [18] utilizes an approximation of the formula with $\log(k+1)$.
[2] **Gowalla**: http://snap.stanford.edu/data/loc-gowalla.html
[3] **Yelp**: https://www.yelp.com/dataset
[4] **Amazon**: http://jmcauley.ucsd.edu/data/amazon/
[5] https://github.com/guoguibing/librec
[6] https://github.com/lyst/lightfm
[7] https://github.com/geek-ai/irgan
[8] https://github.com/dingjingtao/SRNS

Table 2: Overall performance w.r.t. NDCG@50 and Recall@50.

	Gowalla		Yelp		Amazon	
	NDCG@50	RECALL@50	NDCG@50	RECALL@50	NDCG@50	RECALL@50
BPR	0.1214±0.0017	0.1985±0.0023	0.0520±0.0017	0.1070±0.0030	0.0473±0.0024	0.1108±0.0050
AOBPR	0.1385±0.0020	0.2417±0.0016	0.0677±0.0015	0.1346±0.0009	0.0563±0.0023	0.1303±0.0011
WARP	0.1248±0.0052	0.2240±0.0011	0.0636±0.0004	0.1332±0.0009	0.0542±0.0010	0.1267±0.0025
IRGAN	0.1443±0.0019	0.2242±0.0018	0.0695±0.0012	0.1367±0.0020	0.0627±0.0018	0.1395±0.0032
DNS	0.1412±0.0015	0.1839±0.0025	0.0693±0.0016	0.1425±0.0029	0.0615±0.0015	0.1378±0.0043
SRNS	0.1317±0.0007	0.2152±0.0013	0.0493±0.0006	0.1017±0.0009	0.0344±0.0017	0.0774±0.0011
PRIS(U)	0.1334±0.0032	0.2217±0.0024	0.0639±0.0015	0.1273±0.0030	0.0607±0.0013	0.1377±0.0014
PRIS(P)	0.1385±0.0032	0.2282±0.0027	0.0673±0.0016	0.1342±0.0029	0.0697±0.0025	0.1463±0.0033
Kernel	0.1399±0.0024	0.2264±0.0025	0.0658±0.0004	0.1315±0.0012	0.0700±0.0013	0.1495±0.0018
AdaSIR(U)	0.1489±0.0012	0.2500±0.0011	0.0732±0.0024	0.1523±0.0048	0.0731±0.0026	0.1505±0.0052
AdaSIR(P)	**0.1519±0.0016**	0.2516±0.0021	0.0731±0.0027	0.1525±0.0043	0.0740±0.0024	0.1534±0.0054
AdaSIR-W	0.1503±0.0009	**0.2543±0.0005**	**0.0761±0.0012**	**0.1529±0.0041**	**0.0795±0.0027**	**0.1655±0.0053**

- **Kernel** [4]-based sampler approximates the softmax distribution with the non-negative quadratic kernel. It constructs a tree structure for fast sampling.

We test three versions of AdaSIR[9] to validate the recommendation quality of implicit collaborative filtering. AdaSIR(U) and AdaSIR(P) respectively sample candidate items from the uniform and popularity distribution and minimize the loss function in Eq. (1). AdaSIR-W draws items from the uniform distribution and estimates the rank of positive items, minimizing the loss function in Eq. (3).

4.3 Implementation Details

The algorithm is implemented based on PyTorch in a Linux operating system (2.10 GHz Intel Xeon Gold 6230 CPUs and a Tesla V100 GPU). We apply the Adam optimizer to optimize all parameters. In this paper, we first choose the matrix factorization model, i.e. $r_{ci} = \boldsymbol{p}_c^T \boldsymbol{q}_i$. We additionally choose the GMF as the scoring function for extension. Since we do not focus on the feature modeling of the users and items, we do not utilize other contextual information of users. Without the loss of generality, the dimension of the user and item embedding is fixed to 32. The batch size is fixed to 4096 for all three datasets and the learning rate is set to 0.001 by default. The number of training epochs is fixed to 200 for fair comparisons. The coefficient of the regularization of all algorithms is tuned over $\{0.1, 0.01, 0.001, 0.0001\}$. For each positive pair (c, i), we sample 5 negative items for the pair-wise ranking loss. The size of the sample pool for each user is set to 200, 200, 600 for Gowalla, Yelp and Amazon respectively. That is to say, nearly 0.5% of items are drawn during each epoch to update the sample pool. As for the popularity-based distribution for PRIS and AdaSIR, we choose $\log(1 + f_i)$ [27] to calculate the popularity score, where f_i is the number of occurrences of item i in the training data.

The recommendation models are trained on the training data and evaluated on the test data. 10% of the training data is used for validation. The hyperparameters of baselines are carefully tuned depending on NDCG@50. The learning rate for all baselines is fixed to 0.001 by default.

[9]Implementation for AdaSIR: https://github.com/HERECJ/AdaSIR

4.4 Performance comparison

Experiments are conducted on all the real-world datasets, where each algorithm is run for 5 competitions with different random seeds. We report the average value with the standard deviation in Table 2. Obviously, our proposed AdaSIR consistently outperforms all baselines. Compared with the best baseline, AdaSIR achieves a relative 5.2%, 9.8% and 11.81% improvements respectively in terms of NDCG@50 on three datasets. This superior performance demonstrates the effectiveness of the proposed methods, where informative items are drawn and benefit to the convergence of recommendation models. In addition, we have the following observations.

Observation 1: The adaptive samplers show better performances than the static samplers. For example, the best sampler with the static proposal distribution only has a performance of 0.1385 in terms of NDCG@50 on the Gowalla dataset while almost all the dynamic samplers outperform the baselines. This observation implies the necessity of capturing the dynamics of the model and thus more informative items can be sampled to strengthen the learning.

Observation 2: The adoption of the popularity-based distribution has slight improvements for AdaSIR. Utilizing the popularity distribution as the proposal for AdaSIR leads to an average relative 1.04% improvement in terms of NDCG@50 compared with the uniform distribution on three datasets. The relative improvements for the PRIS achieve an average of 8.7% which is larger than that of AdaSIR. The popularity distribution benefits to figuring out more informative items, but the performance of our proposed sampler is not greatly affected by the initial proposal distribution, indicating the robustness of the AdsSIR with different proposal distributions.

Observation 3: The rank estimation of positive items helps distinguish the positive items, leading to the better performance. WARP and AdaSIR-W both assign different importance to the positive items and achieve better performances compared with BPR and AdaSIR(U). The weights correspond with the ranking of the positive pair where the higher ranked items are high quality positive data for training. This implies paying more attention to the bottom-ranked positive items benefits to the model training.

Observation 4: The AdaSIR performs better on sparser datasets. When the dataset is more sparse, it becomes more difficult to select

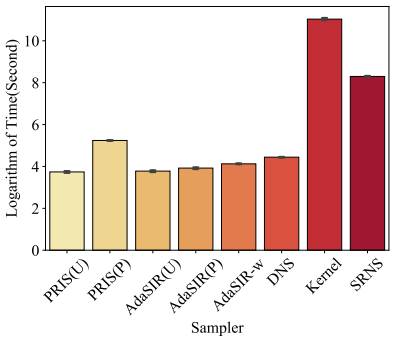


Figure 1: Time Consumption.

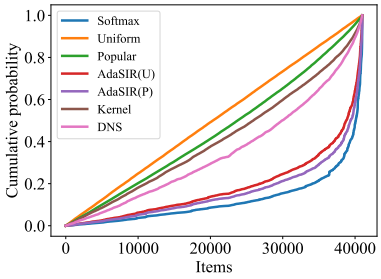


Figure 2: Approximate distributions of different sampling methods.

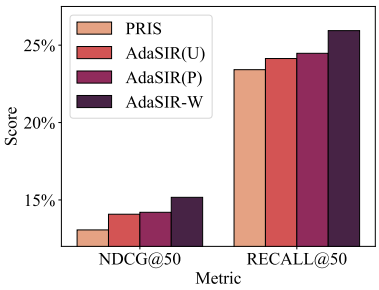


Figure 3: Extension on GMF.

the informative items. Among the experiments over three datasets, the performance on the Amazon dataset has been improved the most with AdaSIR, achieving a relative 13.6% improvement on NDCG@50 compared with the best baseline. Hopefully, the AdaSIR may achieve promising performance on much sparse large-scale data. This demonstrates the good performance of approximation for the softmax distribution to select hard negatives from numerous candidates and demonstrates the effectiveness of capturing the difference between positive user-item pairs.

4.5 Running Time Comparison

Another major concern in negative sampling is the sampling efficiency and we conduct experiments to compare the running time of different samplers. We further conduct experiments to compare the different running time of the sampling procedure and the training procedure to figure out the efficiency of the sampling algorithm. We choose PRIS, DNS, Kernel and SRNS to report the results. The IRGAN and SRNS are implemented in TensorFlow and more time-consuming than our proposed method. Specifically, the SRNS takes over 3 minutes training the Gowalla dataset each epoch with with a Tesla V100 GPU. Each algorithm is run for 20 epochs on the Gowalla dataset and we report the average and the std. over 5 times experiments in Figure 1. We report the time in logarithmic scale.

All the proposed AdaSIR-based methods show the superior performance of the running time, even being comparable to the uniformly sampling method. The Kernel based sampler costs more time on training since it requires $O(D^2 \log N)$ to sample one item, where D denotes the embedding size and N denotes the item number.

The high computational overload of PRIS(P) is the competitive multinomial sampling procedure for each positive pair and the time complexity is $O(n|\mathcal{D}|T_Q)$ for one epoch where n is the number of sampled items, T_Q represents the time of sampling from the multinomial distribution, such as $O(1)$ with the Vose-Alias method [33] or $O(\log N)$ with the binary search. AdaSIR(P) is our proposed method which also regards the popularity as the proposal. However, its sampling procedure is only performed once for each epoch and the time complexity (i.e. $O(|\mathcal{K}_c|T_Q)$) is related to the size of sample pool rather than the data size so that it achieves remarkable speedup. Overall, we provide an extremely efficient sampling method with the simple proposal and significantly enhance the accuracy of approximating the adaptive softmax distribution, even taking almost the same time as the uniformly sampling method.

Table 3: Running Time of one epoch (Seconds) w.r.t. Samplers of seperate phases on the Gowalla dataset.

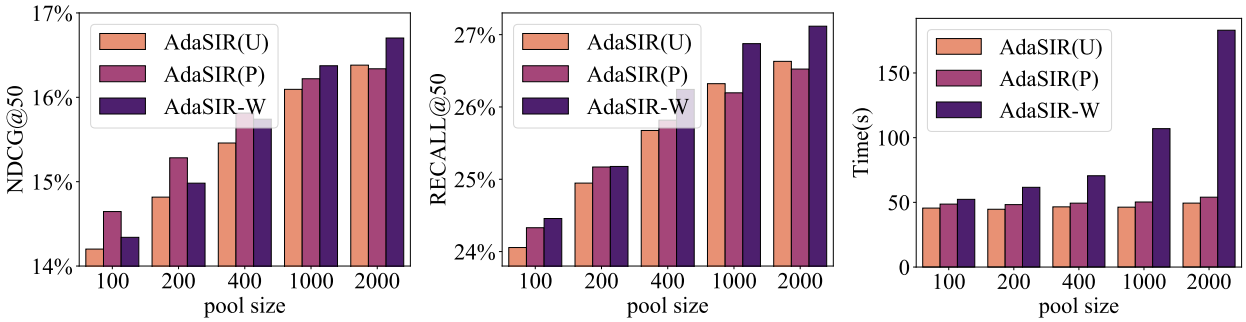
Phase	AdaSIR(U)	AdaSIR(P)	AdaSIR-W
Sampling	0.0741±0.0003	0.0734±0.0003	0.0637±0.0004
Training	0.7572±0.0277	0.7475±0.0180	1.3787±0.0180

To demonstrate the efficiency of the proposed sampling approach, we validate the different running time of the training phase and sampling phase. We conduct experiments on the Gowalla dataset and report the actual running time (in seconds) for one epoch. We run the experiments for 5 epoches and provide the average results with the standard derivation, as shown in Table 3. The results illustrate high efficiency of the proposed sampler, since sampling takes substantially less time than training.

4.6 Approximation performance

Since the sampling distribution plays an important role in improving the quality of items, we conduct experiments to figure out the divergence between the proposal distribution and the softmax distribution. We choose the user embeddings and item embeddings from the well-trained models on the Gowalla dataset and keep them fixed. Then, we randomly pick a user from the user set and compute his/her softmax distribution. Afterwards, we draw 10 items for 5,000 times. Regarding the proposed AdaSIR, the sample pool with a fixed size of 200 is updated each time. The occurrence numbers of the items are approximated as the proposal distribution. We report the cumulative probability distribution of the softmax and proposal distributions in Figure 2. The items are ordered by the popularity.

AdaSIR-based samplers are much closer to the softmax distribution than the other samplers. This indicates the good performance of the proposed sampler in reducing the bias between the proposal distribution and the softmax distribution. Although the sampler of DNS requires low time complexity, there exists a huge deviation from the softmax distribution. When the number of the candidates increases, the popular items are more likely to be sampled which will exacerbate the long tail effect. The kernel-based sampler also has a huge divergence between the proposal and softmax distribution. A reason may lie in that the embedding mapped with the quadratic kernel to ensure the non-negative scores leads to the bad estimation. Those items with negative scores have larger probabilities to be sampled but contain less information. These superior



(a) NDCG@50 (b) Recall@50 (c) Running Time

Figure 4: Comparison results of AdaSIR w.r.t. the pool size on the Gowalla dataset.

performances of the proposed samplers when approximating the softmax distribution empowers the strong capability of capturing the dynamics during model training and high-quality items can be sampled for better convergence.

4.7 Extensions with Various scoring function

Previous works propose adaptive samplers for simple score functions, which are designed for matrix factorization but cannot fit in arbitrary recommendation models. In this section, we additionally adopt the GMF as the score function to validate the effectiveness of AdaSIR with different types of recommendation models. The experiments are conducted on the Gowalla. The dimension of the embeddings is set to 32. The experiments are run for 200 epochs.

As shown in Figure 3, AdaSIR-based samplers show better performances than PRIS, the competing and latest baseline method, achieving at least a relative 13.06% improvement in terms of NDCG@50. We also observe similar improvements with the matrix factorization models, where This indicates the adaptability of the proposed samplers for complicated scoring functions. In the future, we will explore AdaSIR with more complex models (e.g., NCF, GNN-based models) which enhance the expressiveness of embeddings and improve the recommendation quality.

4.8 Varying the pool size

To figure out the influence of different sizes of the sample pool, we evaluate the recommendation quality and running time of AdaSIR on the Gowalla dataset. We vary the pool size in {100, 200, 400, 1000, 2000} for the AdaSIR-based methods.

Figure 4(a) and 4(b) report the performance of the recommendation quality. With the increase of the pool size, all AdaSIR based methods perform better. Since more items are collected, the proposal distribution gets closer to the softmax distribution and thus more informative items can be sampled. Overall, the AdaSIR-W competes with the other two methods in terms of NDCG@50 and Recall@50 when the pool size is 1000. The AdaSIR-W consistently shows good performance on Recall@50. The reason may lie in that the rank estimation gets more accurate when the pool size becomes large so that the high-quality positive items can be distinguished during training. The higher Recall@50 implies the good performance of distinguishing the ground-truth positives in testing.

Figure 4(c) shows the running time of different sizes of sample pool. The AdaSIR-W estimates the rank of the positive item by enumerating the negative items in $\mathcal{K}_c$, which has the same size as the sample pool, so that it takes almost linear time to perform the rank estimation. An exciting point is that the AdaSIR(U) and AdaSIR(P) take almost the same time when the size of sample pool increases. The two-pass sampling procedure dramatically reduces the computational overload and shows a superior advantage in the great number of the sample pool.

5 RELATED WORK

The main concern of recommendation from implicit feedback is the lack of the negative items. Despite the existence of the expilict negative behaviors, such as non-clicks and low ratings, considering the enormous number of candidate items, these feedbacks are extremely sparse. If we only consider these feedbacks as negative samples, it will cause selection bias [10]. Fortunately, the community agrees on the importance of unobserved items when learning the recommendation models from both academic [10, 12] and industrial researchers [13, 20, 35, 39, 40]. The proposed algorithms for item recommendation differ in how to exploit the unobserved items. WRMF [19] and OCCF [29], which are proposed more than ten years, treat all the unobserved items as negative feedbacks. A series of works [2, 5, 22] design implicit regularizers to penalize the high ratings of the unobserved items. However, these algorithms get trapped in the high computational cost given great number of items. Negative sampling is then proposed and has been investigated in the recommender systems [21, 25, 31, 32, 42]. These samplers differ in the the proposal distribution and the re-weights of the items.

6 CONCLUSION

In this work, we propose an Adaptive Sampling method based on the *Importance Resampling*, an efficient sampling approach based on two-stage sampling method. The AdaSIR method provides an accurate approximation of the adaptive distribution and thus samples more informative items, which benefits the model training. The proposed sampler supports the reuse of high-quality items and the rank approximation for positive items through the maintained sampling pool with fixed size. Experiments are conducted on the three real-world datasets, showing superior performance in terms of efficiency and effectiveness.

ACKNOWLEDGMENTS

The work is supported by the National Natural Science Foundation of China (No. 62022077, 61972069, 61836007 and 61832017), and Shenzhen Municipal Science and Technology R&D Funding Basic Research Program (JCYJ20210324133607021).

REFERENCES

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

[2] Immanuel Bayer, Xiangnan He, Bhargav Kanagal, and Steffen Rendle. 2017. A generic coordinate descent framework for learning from implicit feedback. In *Proceedings of the 26th International Conference on World Wide Web*. 1341–1350.

[3] 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.

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

[5] Jin Chen, Defu Lian, and Kai Zheng. 2019. Improving one-class collaborative filtering via ranking-based implicit regularizer. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 37–44.

[6] Jin Chen, Defu Lian, and Kai Zheng. 2020. Collaborative filtering with ranking-based priors on unknown ratings. *IEEE Intelligent Systems* 35, 5 (2020), 38–49.

[7] Ting Chen, Yizhou Sun, Yue Shi, and Liangjie Hong. 2017. On sampling strategies for neural network-based collaborative filtering. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 767–776.

[8] Xuanhao Chen, Yan Zhao, Guanfeng Liu, Rui Sun, Xiaofang Zhou, and Kai Zheng. 2020. Efficient Similarity-aware Influence Maximization in Geo-social Network. *TKDE* (2020).

[9] Yue Cui, Hao Sun, Yan Zhao, Hongzhi Yin, and Kai Zheng. 2021. Sequential-knowledge-aware Next POI Recommendation: A Meta-learning Approach. *TOIS* (2021).

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

[11] Jingtao Ding, Yuhan Quan, Quanming Yao, Yong Li, and Depeng Jin. 2020. Simplify and Robustify Negative Sampling for Implicit Collaborative Filtering. In *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin (Eds.), Vol. 33. Curran Associates, Inc., 1094–1105.

[12] Jingtao Ding, Yuhan Quan, Quanming Yao, Yong Li, and Depeng Jin. 2020. Simplify and Robustify Negative Sampling for Implicit Collaborative Filtering. *arXiv preprint arXiv:2009.03376* (2020).

[13] Miao Fan, Jiacheng Guo, Shuai Zhu, Shuo Miao, Mingming Sun, and Ping Li. 2019. MOBIUS: towards the next generation of query-ad matching in baidu's sponsored search. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2509–2517.

[14] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web*. 173–182.

[15] Xiangnan He, Hanwang Zhang, Min-Yen Kan, and Tat-Seng Chua. 2016. Fast matrix factorization for online recommendation with implicit feedback. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*. 549–558.

[16] Wassily Hoeffding. 1994. Probability inequalities for sums of bounded random variables. In *The Collected Works of Wassily Hoeffding*. Springer, 409–426.

[17] 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.

[18] 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 (WWW '17)*. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, CHE, 193–201. https://doi.org/10.1145/3038912.3052639

[19] Yifan Hu, Yehuda Koren, and Chris Volinsky. 2008. Collaborative filtering for implicit feedback datasets. In *2008 Eighth IEEE International Conference on Data Mining*. Ieee, 263–272.

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

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

[22] Walid Krichene, Nicolas Mayoraz, Steffen Rendle, Li Zhang, Xinyang Yi, Lichan Hong, John Anderson, et al. 2018. Efficient Training on Very Large Corpora via Gramian Estimation. In *International Conference on Learning Representations*.

[23] Xiang Li, Tao Qin, Jian Yang, Xiaolin Hu, and Tie-Yan Liu. 2016. LightRNN: Memory and Computation-Efficient Recurrent Neural Networks. In *NIPS*.

[24] Defu Lian, Jin Chen, Kai Zheng, Enhong Chen, and Xiaofang Zhou. 2021. Ranking-based Implicit Regularization for One-Class Collaborative Filtering. *IEEE Transactions on Knowledge and Data Engineering* (2021).

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

[26] Dawen Liang, Laurent Charlin, James McInerney, and David M Blei. 2016. Modeling user exposure in recommendation. In *Proceedings of the 25th international conference on World Wide Web*. 951–961.

[27] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Distributed representations of words and phrases and their compositionality. *arXiv preprint arXiv:1310.4546* (2013).

[28] Frederic Morin and Yoshua Bengio. 2005. Hierarchical probabilistic neural network language model.. In *Aistats*, Vol. 5. Citeseer, 246–252.

[29] Rong Pan, Yunhong Zhou, Bin Cao, Nathan N Liu, Rajan Lukose, Martin Scholz, and Qiang Yang. 2008. One-class collaborative filtering. In *2008 Eighth IEEE International Conference on Data Mining*. IEEE, 502–511.

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

[31] 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.

[32] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme. 2009. BPR: Bayesian personalized ranking from implicit feedback. In *Proceedings of UAI'09*. AUAI Press, 452–461.

[33] Alastair J Walker. 1977. An efficient method for generating discrete random variables with general distributions. *ACM Transactions on Mathematical Software (TOMS)* 3, 3 (1977), 253–256.

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

[35] 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.

[36] Shoujin Wang, Liang Hu, Yan Wang, Longbing Cao, Quan Z. Sheng, and Mehmet Orgun. 2019. Sequential recommender systems: challenges, progress and prospects. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*. AAAI Press, 6332–6338.

[37] Jason Weston, Samy Bengio, and Nicolas Usunier. 2010. Large scale image annotation: learning to rank with joint word-image embeddings. *Machine learning* 81, 1 (2010), 21–35.

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

[39] 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.

[40] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. 2018. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 974–983.

[41] Hsiang-Fu Yu, Mikhail Bilenko, and Chih-Jen Lin. 2017. Selection of negative samples for one-class matrix factorization. In *Proceedings of the 2017 SIAM International Conference on Data Mining*. SIAM, 363–371.

[42] 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*. 785–788.