Simplify and Robustify Negative Sampling for Implicit Collaborative Filtering

Jingtao Ding[1]* **Yuhan Quan**[1] **Quanming Yao**[2] **Yong Li**[1] **Depeng Jin**[1]
[1]Tsinghua University, [2]4Paradigm Inc (Hong Kong)

Abstract

Negative sampling approaches are prevalent in implicit collaborative filtering for obtaining negative labels from massive unlabeled data. As two major concerns in negative sampling, efficiency and effectiveness are still not fully achieved by recent works that use complicate structures and overlook risk of false negative instances. In this paper, we first provide a novel understanding of negative instances by empirically observing that only a few instances are potentially important for model learning, and false negatives tend to have stable predictions over many training iterations. Above findings motivate us to simplify the model by sampling from designed memory that only stores a few important candidates and, more importantly, tackle the untouched false negative problem by favouring high-variance samples stored in memory, which achieves efficient sampling of true negatives with high-quality. Empirical results on two synthetic datasets and three real-world datasets demonstrate both robustness and superiorities of our negative sampling method.

1 Introduction

Collaborative filtering (CF), as the key technique of personalized recommender systems, focuses on learning user preference from the observed user-item interactions [27, 33]. Today's recommender systems also witness the prevalence of implicit user feedback, such as purchases in E-commerce sites and watches in online video platforms, which is much easier to collect compared to the explicit feedback (such as ratings) on item utility. In above examples, each observed interaction normally indicates a user's interest on an item, *i.e.*, a positive label, while the rest unobserved interactions are unlabeled. As for learning an implicit CF model from this positive-only data, a widely adopted approach is to select a few instances from the unlabeled part and treat them as negative labels, also known as negative sampling [10, 33]. Then, the CF model is optimized to give positive instances higher scores than those given to negative ones [33].

Similar to other related applications in representation learning of text [26] or graph data [29], negative sampling in implicit CF also has two major concerns, i.e., efficiency and effectiveness [10, 45]. First, the efficient sampling process is required, as the number of unobserved user-item interactions can be extremely huge. Second, the sampled instances need to be high-quality, so as to learn useful information about user's negative preference. However, since implicit CF is an application-driven problem where user behaviors play an important role, it may be unrealistic to assume that unobserved interactions are all negative, which introduces false negative instances into training process [20, 25, 48]. For example, an item may be ignored because of its displayed position and form, not necessarily the user's dislike. Therefore, false negative instances naturally exist in implicit CF.

Previous works of negative sampling in implicit CF mainly focus on replacing the uniform sampling distribution with another proposed distribution, so as to improve the quality of negative samples. Similar to the word-frequency based distribution [26] and node-degree based distribution [29]

---

*The first three authors have equal contributions.

used in other domains, an item-popularity based distribution that favours popular items is usually adopted [10, 41]. In terms of sample quality, the strategy emphasizing hard negative samples has been proven to be more effective [28], as it can bring more information for model training. Specifically, this is achieved by either assigning higher probability to instances with large prediction score [32, 45] or leveraging techniques of adversarial learning [11, 28, 37]. Nevertheless, the above hard negative sampling approaches cannot simultaneously meet the requirements on efficiency and effectiveness. On the one hand, several state-of-the-art solutions [11, 28] use complicate structures like generative adversarial network (GAN) [17] for generating negative instances, which has posed a severe challenge on model efficiency. On the other hand, all these methods overlook the risk of introducing false negative instances and instead only focus on hard ones, making the sampling process less robust for training an effective CF model with false negatives.

Different from above works, this paper formulates the negative sampling problem as efficient learning from unlabeled data with the presence of noisy labels, *i.e.*, false negative instances. We propose to simplify and robustify the negative sampling for implicit CF, which has three main challenges:

- **How to capture the dynamic distribution of true negative instances with a simple model?** In the implicit CF problem, true negative instances are hidden inside the massive unlabeled data, along with false negative instances. Although negative instances in other domains follow a skewed distribution and can be modeled by a simple model [26, 46], it remains unknown if this prior knowledge can be applied in the implicit CF problem that expects true negative instances only.
- **How can we reliably measure the quality of negative samples?** Given the risk of introducing false negative instances, the quality of negative samples needs to be measured in a more reliable way. However, it is non-trivial to design a discriminative criterion that can help to accurately identify true negative instances with high quality.
- **How can we efficiently sample true negative instances of high-quality?** Although learning effective information from unlabeled and noisy data is related to general machine learning approaches including positive-unlabeled leaning [22] and instance re-weighting [31], these methods are not suitable for implicit CF problem, where the huge number of unobserved user-item interactions requires an efficient modeling. Instead, our proposed method needs to maintain both efficiency, by sampling, and effectiveness, by considering samples' informativeness and reliability simultaneously. This has not been tackled before in both implicit CF and other similar problems.

Solving above three challenges calls for a deep and fundamental understanding of different negative instances in implicit CF problem. In this paper, we empirically find that negative instances with large prediction scores are important for the model learning but generally rare, *i.e.*, following a skewed distribution. A more novel finding is that false negative instances always have large scores over many iterations of training, *i.e.*, a lower variance, which provides a new angle on tackling false negative problem remained in existing approaches. Motivated by above two findings, we propose a novel simplified and robust negative sampling approach, named SRNS, that 1) captures the dynamic distribution of negative instances with a memory-based model, by simply maintaining the promising candidates with large scores, and 2) leverages a high-variance based criterion to reliably measure the quality of negative samples, reducing the risk of false negative instances effectively. Above two designs are further combined into a two-step sampling scheme that constantly alternates between score-based memory update and variance-based sampling, so as to efficiently sample true negative instances with high-quality. Experiment results on two synthetic datasets demonstrate the robustness of our SRNS under various levels of noisy circumstances. Further experiments on three real-world datasets also empirically validates its superiorities over state-of-the-art baselines, in terms of effectiveness and efficiency.

2 Background

Training an implicit CF model generally involves three main steps, *i.e.*, choosing scoring function r, objective function L and negative sampling distribution p_{ns}. The scoring function $r(\mathbf{p}_u, \mathbf{q}_i, \boldsymbol{\beta})$ calculates the relevance between a user $u \in \mathcal{U}$ and an item $i \in \mathcal{I}$ based on u's embedding $\mathbf{p}_u \in \mathbb{R}^F$ and i's embedding $\mathbf{q}_i \in \mathbb{R}^F$, with a learnable parameter $\boldsymbol{\beta}$. It can be chosen among various candidates including matrix factorization (MF) [23], multi-layer perceptron (MLP) [19], graph neural network (GNN) [3, 39], etc. For example, the generalized matrix factorization (GMF) [19] is: $r(\mathbf{p}_u, \mathbf{q}_i, \boldsymbol{\beta}) = \boldsymbol{\beta}^\top (\mathbf{p}_u \odot \mathbf{q}_i)$, where the learnable parameter of r is a vector $\boldsymbol{\beta}$ and $\odot$ denotes element-wise product. A large value of $r(\mathbf{p}_u, \mathbf{q}_i, \boldsymbol{\beta})$ indicates u's strong preference on i, denoted

Table 1: Comparison of the proposed SRNS with closely related works, where $\text{rk}(j|u)$ is the (u, j)'s rank sorted by score, pop_j is the j's item popularity, B is the mini-batch size, T is the time complexity of computing an instance score, E is the epoch of lazy-update, and $\mathcal{F}$ denotes false negative.

	$p_{\text{ns}}(j|u)$	Optimization	Time Complexity	Robustness
Uniform [33]	Uniform($\{j \notin \mathcal{R}_u\}$)	SGD (from scratch)	$O(BT)$	×
NNCF [10]	$\propto (\text{pop}_j)^{0.75}$	SGD (from scratch)	$O(B^2 T)$	×
AOBPR [32]	$\propto \exp(-\text{rk}(j|u)/\lambda)$	SGD (from scratch)	$O(BT)$	×
IRGAN [37]	learned $\bar{p}_{\text{ns}}(j|u)$ (GAN)	REINFORCE (pretrain)	$O(B|\mathcal{I}|T)$	×
AdvIR [28]	learned $\bar{p}_{\text{ns}}(j|u)$ (GAN)	REINFORCE (pretrain)	$O(BS_1 T)$	×
SRNS (proposed)	variance-based (see (4))	SGD (from scratch)	$O(\frac{B}{E}(S_1 + S_2)T)$	$\checkmark$

by r_{ui} for simplicity. Each observed instance between u and the interacted item $i \in \mathcal{R}_u$, i.e., (u, i), can be seen as a positive label. As for the rest unobserved interactions, i.e., $\{(u, j)|j \notin \mathcal{R}_u\}$, the probability of (u, j) being negative is

$$P_{\text{neg}}(j|u, i) = \text{sigmoid}(r_{ui} - r_{uj}), \tag{1}$$

which approaches to 1 when $r_{ui} \gg r_{uj}$. In other words, when learning user preference in implicit CF, we care more about the pairwise ranking relation between an observed interaction $(u, i) \in \mathcal{R}$ and another unobserved interaction (u, j), instead of absolute values of r_{ui} and r_{uj}. The learning objective can be formulated as minimizing following loss function [33]:

$$L(\{\mathbf{p}_u\}, \{\mathbf{q}_i\}, \boldsymbol{\beta}) = \sum_{(u,i) \in \mathcal{R}} \left[\mathbb{E}_{j \sim p_{\text{ns}}(j|u)} \left[-\log P_{\text{neg}}(j|u, i) \right] \right], \tag{2}$$

where the negative instance (u, j) is sampled according to a specific distribution $p_{\text{ns}}(j|u)$. Learning above objective is equivalent to maximizing the likelihood of observing such pairwise ranking relations $r_{ui} > r_{uj}$, which can be replaced by other objectives used in implicit CF problems, such as marginal hinge loss [43] and binary cross-entropy loss [19].

The most widely used $p_{\text{ns}}(j|u)$ is the uniform distribution [33], suffering from low quality of samples. To solve this, previous works [11, 28, 32] propose to sample much harder instances, containing more information. Among them, state-of-the-arts [11, 28] simultaneously learn a parameterized $\bar{p}_{\text{ns}}(j|u)$ to maximize above loss function in (2), based on GAN. Therefore, the sampled negative instance (u, j) corresponds to a low $P_{\text{neg}}(j|u, i)$ and a high r_{uj}, which is generally hard for CF model to learn. In other words, (u, j) has a high probability of being positive, denoted as $P_{\text{pos}}(j|u, i) = 1 - P_{\text{neg}}(j|u, i)$. Different choices of $p_{\text{ns}}(j|u)$ in previous works are listed in Table 1. Since none of them have enough robustness to handle false negative instances, and GAN-based model structure is much more complicate, our goal is to propose a more robust and simplified negative sampling method.

3 SRNS: the Proposed Method

To improve robustness and efficiency for negative sampling in implicit CF, we seek for a deep understanding of different negative instances, including false negative instances and negative instances obtained by uniform sampling or hard negative sampling. We then describe the proposed method based on these understandings.

3.1 Understanding False Negative Instances

In previous works [28, 32], the positive-label probability P_{pos} (or the prediction score) is widely used as the sampling criterion, as it is proportional to the sample difficulty. Therefore, in Figure 1 (details on setup are in Appendix C.2), we have a closer look at the negative instances' distribution w.r.t. P_{pos} and further analyze the possibility of using P_{pos} to discriminate true negative instances and false negative instances. Besides, we are also curious about the model's prediction uncertainty regarding to different negative instances, and investigate the variance of P_{pos} in Figure 1(d).

Based on above analysis of negative instances in implicit CF, we have following two findings:

1) The score distribution of negative instances is highly skew. Regardless of the training, only a few negative instances have large scores (Figure 1(a)).

(a) Distribution of P_{pos}.　(b) Comparing P_{pos} of neg.　(c) Comparing LER.　(d) Comparing $\frac{Std}{Mean} P_{pos}$.

Figure 1: Analysis of negative instances on ML-100k. D: difficulty level; Label Error Ratio (LER): $= \text{(\# of false negative samples)} / \text{(\# of all selected negative samples)}$; CCDF: complementary cumulative distribution function, $p50$: median value among a set of negative instances, $Std/Mean$: normalized variance).

2) Both false negative instances and hard negative instances have large scores (Figure 1(b)), making it hard to discriminate them (harder negative samples are more likely be false negative, Figure 1(c)). However, the false negative instances have lower prediction variance comparatively (Figure 1(d)).

The first finding demonstrates the potential of just capturing a part of the full distribution corresponding to those large-scored negative instances, which are more likely to be high-quality. Similar observations have also been discussed in graph representation learning, suggesting a skewed negative sampling distribution that focuses on hard ones [42, 46].

As for the second finding, it provides us a reliable way of measuring sample quality based on prediction variance, sharing the same intuition with [8] that improves stochastic optimization by emphasizing high variance samples. Specifically, we prefer those negative samples with both large scores and high variances, avoiding false negative instances that always have large scores over many iterations of training. In terms of robustifying negative sampling process, none of above works in implicit CF and other domains have tackled this problem, except for a simple workaround that only selects hard negative samples but avoids the hardest ones [43, 46].

3.2　SRNS Method Design

As above, on the one hand, we are motivated to use a small amount of memory for each user, storing hard negative instances that have large potential of being high-quality. This largely simplifies the model structure, by focusing on a partial set of instances, which thus improves efficiency. On the other hand, we propose a variance-based sampling strategy to effectively obtain samples that are both reliable and informative. Our simplified and robust negative sampling (SRNS) approach addresses the remaining challenges on model efficiency and robustness. Algorithm 1 shows the implicit CF learning framework, *i.e.*, minimizing loss function in (2), based on SRNS.

Algorithm 1: The proposed Simplified and Robust Negative Sampling (SRNS) method.

Input : Training set $\mathcal{R} = \{(u, i)\}$, embedding dimension F, scoring function r with learnable parameter $\boldsymbol{\beta}$, and memory $\{\mathcal{M}_u | u \in \mathcal{U}\}$, each with size S_1;

Output : Final user embeddings $\{\mathbf{p}_u | u \in \mathcal{U}\}$ and item embeddings $\{\mathbf{q}_i | i \in \mathcal{I}\}$, and r;

1　Initialize $\{\mathbf{p}_u | u \in \mathcal{U}\}$ and $\{\mathbf{q}_i | i \in \mathcal{I}\}$, $\boldsymbol{\beta}$ and $\{\mathcal{M}_u | u \in \mathcal{U}\}$;

2　**for** $t = 1, 2, ..., T$ **do**

3　　Sample a mini-batch $\mathcal{R}_{batch} \in \mathcal{R}$ of size B;

4　　**for** *each* $(u, i) \in \mathcal{R}_{batch}$ **do**

5　　　Get the candidate items from u-related memory $\mathcal{M}_u$;

6　　　Sample the item j from $\mathcal{M}_u$, based on the variance-based sampling strategy (4);

7　　　Uniformly sample S_2 items from $\{k | k \notin \mathcal{R}_u\}$ ($\bar{\mathcal{M}}_u$), and merge with original $\mathcal{M}_u$;

8　　　Update $\mathcal{M}_u$ based on the score-based updating strategy (3);

9　　　Update embeddings $\{\mathbf{p}_u, \mathbf{q}_i, \mathbf{q}_j\}$ and parameters $\boldsymbol{\beta}$ based on gradient *w.r.t.* L (2).

10　　**end**

11　**end**

The learning process of the SRNS is carried out in mini-batch mode, and alternates between two main steps. First, according to the high-variance based criterion, a negative instance for each training instance (u, i) is sampled from u's memory $\mathcal{M}_u$ (line 6), which already stores S_1 candidates with high potential. To improve efficiency, all positive instances of a same user u is designed to share one memory $\mathcal{M}_u$. Second, as the model is constantly changing during the training process, $\mathcal{M}_u$ requires

4

a dynamic update so as to keep track of the promising candidates for negative sampling. Specifically, this is completed by first extending it into $\mathcal{M}_u \cup \bar{\mathcal{M}}_u$ with additional S_2 instances that was uniformly sampled (line 7), and then choosing S_1 hard candidates to obtain a new $\mathcal{M}_u$ (line 8). A similar two-step scheme is adopted by a related work that focuses on negative sampling for knowledge graph embeddings [46]. However, unlike SRNS leveraging the instance's variance in the sampling step, it uniformly chooses an instance from memory, which cannot enhance model's robustness effectively.

Score-based memory update. In this part, we propose a memory-based model to simply capture the dynamic distributions of true negative instances. Specifically, a memory $\mathcal{M}_u = \{(u, k_1), ..., (u, k_{S_1})\}$ of size S_1 is assigned to each user u, storing the negative instances that are available to u in sampling. To ensure only those informative instances are maintained, we design a score-based strategy to dynamically update the $\mathcal{M}_u$, which tends to involve more hard negative instances. For an extended memory that merges the old $\mathcal{M}_u$ and a set of uniformly sampled instances $\bar{\mathcal{M}}_u$, *i.e.*, $\mathcal{M}_u \cup \bar{\mathcal{M}}_u$, the new $\mathcal{M}_u$ is updated by sampling S_1 instances according to the following probability distribution:

$$\bar{\Psi}(k|u, \mathcal{M}_u \cup \bar{\mathcal{M}}_u) = \exp(^{r_{uk}}/\tau)/\sum\nolimits_{k' \in \mathcal{M}_u \cup \bar{\mathcal{M}}_u} \exp(^{r_{uk'}}/\tau), \quad (3)$$

where a lower temperature $\tau \in (0, +\infty)$ would make $\bar{\Psi}$ focus more on large-scored instances.

Variance-based sampling. As we have demonstrated in finding 2, oversampling hard negative instances may increase the risk of introducing false negatives, making above score-based updating strategy less robust. Motivated by the observed low-variance characteristic of false negatives, we propose a robust sampling strategy that can effectively avoid this noise by favouring those high-variance candidates. Given a positive instance (u, i) and u's memory $\mathcal{M}_u$, for each candidate $(u, k) \in \mathcal{M}_u$, we maintain $P_{\text{pos}}(k|u, i)$ values at tth training epoch as $[P_{\text{pos}}(k|u, i)]_t$. The proposed variance-based sampling strategy chooses the negative instance (u, j) from $\mathcal{M}_u$ by:

$$j = \arg\max_{k \in \mathcal{M}_u} P_{\text{pos}}(k|u, i) + \alpha_t \cdot \text{std}[P_{\text{pos}}(k|u, i)]. \quad (4)$$

Note that we also consider the instance difficulty, i.e., $P_{\text{pos}}(k|u, i)$, to ensure the informativeness of sampled negative instances, with a hyper-parameter α_t controlling the importance of high-variance at the t-th training epoch. When $\alpha_t = 0$, our proposed sampling approach degenerates into a difficulty-only strategy that follows the similar idea as previous works [11, 28, 32]. Since all instances tend to have high variance at an early training stage, the variance term should not be weighted too much. Therefore, we expect a "warm-start" setting of α_t that reduces the influence of prediction variance at first and then gradually strengthens it (details of α_t will be discussed in Section 4.2).

Bootstrapping. In Algorithm 1, false negative instances are identified by checking their prediction variance. However, this assumes that the CF model has some discriminative ability. There is an important observation that deep models can memorize easy training instances first and gradually adapt to hard instances [1, 18, 44]. Fortunately, we also observe such memorization effect for deep CF models (see Section 4.2), which means that the false negative instances among unlabeled data are generally more difficult and may not be memorized at an early stage. In other words, SRNS can be self-boosted by first learning to discriminate those easy negative instances and then tackling the rest real hard ones with the help of variance-based criterion.

3.3 Complexity Analysis

Here, we analyze the time complexity of SRNS (Algorithm 1) and compare it with related negative sampling approaches in Table 1. Compared with a uniform sampling approach [33], the main additional cost comes from score-based memory update and variance-based sampling. The former requires to compute scores of $S_1 + S_2$ candidates for each positive instance and sample S_1 of them according to a normalized distribution $\bar{\Psi}$ that is based on computed scores. Thus the time complexity is $O((S_1 + S_2)T)$, where T denotes the operation count of score computation. As for the latter, computing $\text{std}[P_{\text{pos}}(k|u, i)]$ of each candidate and choosing the final negative instance in (4) take $O(S_1)$. Thus, the cost is $O((S_1 + S_2)T)$ for each positive instance, which can be reduced to $O((S_1 + S_2)T/E)$ using lazy-update every E epochs. Model parameters in CF consists of two parts, *i.e.*, embeddings and scoring function parameters, and the former is generally much larger. Specifically, SRNS's model complexity is about $(|\mathcal{U}| + |\mathcal{I}|)F$, which can double in those GAN-based state-of-the-arts [28, 37]. As in Table 1, SRNS is not only more simplified (in terms of time complexity), but also can be easily trained from scratch.

4 Experiments

We first conduct controlled experiments with synthetic noise, so as to investigate SRNS's robustness to false negative instances (Section 4.2). Then, we evaluate the SRNS's performance on the implicit CF task, based on real data experiments (Section 4.3).

4.1 Experimental Settings

Dataset. Table 2 summarizes datasets used for experiments, which are popularly used in the literature [16, 19, 28, 37]. We use ML-100k and Ecom-toy for synthetic data experiments and do a 4:1 random splitting for train/test data. To simulate the noise, we randomly select 50%/25% of groundtruth records in the test set of ML-100k/Ecom-toy. The selected records can be regarded as false negative instances during training, denoted as $\mathcal{F}$. As for real data experiments, we use the rest three datasets and adopt leave-one-out strategy, *i.e.*, holding out users' most recent records and second to the last records (sorted *w.r.t.* time-stamp) as the test set and validation set, respectively [19, 33].

Table 2: Statistics of datasets.

Category	Dataset	User	Item	Train	Validation	Test	$\mathcal{F}$ (Noise)
Synthetic noisy dataset	Movielens-100k	942	1,447	44,140	-	11,235	5,509
	Ecommerce-toy	1,000	2,000	60,482	-	14,612	3,246
Real-world dataset	Movielens-1m	6,028	3,533	563,186	6,028	6,028	-
	Pinterest	55,187	9,916	1,390,435	55,187	55,187	-
	Ecommerce	66,450	59,290	1,625,006	66,441	66,450	-

Baselines. We compare SRNS with three types of methods listed in Table 1. First, for methods using a fixed negative sampling distribution, we choose Uniform [33] and NNCF [10]. Second, for methods based on hard negative sampling, we choose AOBPR [32], IRGAN [37], RNS-AS [11] and AdvIR [28], where the last three are GAN-based state-of-the-arts. Finally, we also compare with a non-sampling approach ENMF [9] that regards all the unlabeled data as negative labels.

Hyper-parameter and optimizer. For better performance, we mainly use GMF [19] as the scoring function r, but also experiment on another popular choice, *i.e.*, a MLP with sigmoid activation (Section 4.3). Hyper-parameters of SRNS and baselines are carefully tuned according to validation performance (details are in Appendix B.4). For all experiments, Adam optimizer is used and the mini-batch size is 1024. Specifically, we run each synthetic data experiment 400 epochs and repeat five times. As for real data experiments, we conduct the standard procedure [10, 39], running 400 epochs and terminating training if validation performance does not improve for 100 epochs.

Experimental setup. In the implicit CF, the model is evaluated by testing if it can generate a better ranked item list $\mathcal{S}_u$ for each user u. In the synthetic case, $\mathcal{S}_u$ contains u's test items $\mathcal{G}_u$ and the rest items that are not interacted by u. While in the real-world case with much larger item count $|\mathcal{I}|$, we follow a common strategy [19, 23] to fix the list length $|\mathcal{S}_u|$ as 100, by randomly sampling $100 - |\mathcal{G}_u|$ non-interacted items. We measure $\mathcal{S}_u$'s performance by *Recall* and *Normalized Discounted Cumulative Gain* (NDCG). Specifically, $Recall@k(u) = |\mathcal{S}_u(k) \cap \mathcal{G}_u|/|\mathcal{G}_u|$, $NDCG@k(u) = \sum_{i \in \mathcal{S}_u(k) \cap \mathcal{G}_u} 1/\log_2(p_i + 1)$, where $\mathcal{S}_u(k)$ denotes truncated $\mathcal{S}_u$ at k and p_i denotes i's rank in $\mathcal{S}_u$. Comparatively, NDCG accounts more for the position of the hit by assigning higher scores to hits at top ranks and $NDCG@1(u) = Recall@1(u)$. We choose a rather small k in $\{1, 3\}$, which matters more in applications. The final report Recall/NCDG is the average score among all test users.

4.2 Synthetic Noise Experiments

Synthetic false negative instances are simulated by flipping labels of test records ($\mathcal{F}$ in Table 2). To manually inject this noise, we constantly feed a false negative into each user's memory $\mathcal{M}$ during sampling process. We control this impact by varying the size of available false negative instances in different experiments, randomly sampling $\sigma \times 100$ (%) from $\mathcal{F}$ ($\sigma \in [0, 1]$). Note that $\sigma = 0$ indicates an "ideal" case where $\mathcal{M}$ is not influenced by $\mathcal{F}$. In these experiments, we fix the memory size S_1 as 20 (details on setup are in Appendix C.3).

(a) N@3, ML-100k   (b) R@3, ML-100k   (c) N@3, Ecom-toy   (d) R@3, Ecom-toy

Figure 2: Average test Recall/NDCG of SRNS with different σ on two synthetic datasets over the last 50 epochs. Two sampling strategies are compared, *i.e.*, difficulty-only vs. variance-based (proposed).

Sampling criterion. We first investigate if the high-variance based criterion in SRNS can indeed identify true negative instances which are of high-quality, by comparing with a difficulty-only strategy (*i.e.*, weight $\alpha_t = 0$). Figure 2 shows comparison results *w.r.t.* test Recall and NDCG, under different levels of noisy supervision (σ). Although increasing noisy level can harm model's performance, we can observe a consistent improvement of variance-based strategy with different σ.

Warm-start. Motivated by [18], we propose to linearly increase the value of α_t as epoch number t increases. Specifically, $\alpha_t = \alpha \cdot \min(t/T_0, 1)$, where T_0 denotes the threshold of stopping increase. In Figure 3(a)-(b), we compare this increased setting of α_t with another two competitor, *i.e.*, $\alpha_t = \alpha$ (flat) and $\alpha_t = \alpha \cdot \max(1 - t/T_0, 0)$ (decreased). It can be clearly observed that the increased setting of α_t performs better than others, as the former can better leverage variance-based criterion after false negative instances become more stable.

(a) $\sigma = 1$, ML-100k   (b) $\sigma = 0.5$, Ecom-toy   (c) $\sigma = 1$, ML-100k   (d) $\sigma = 1$, Ecom-toy

Figure 3: (a)-(b) Test NDCG vs. number of epochs on two datasets, with the error bar for STD highlighted as a shade. (c)-(d) Memorization effect of the CF model under extremely noisy supervision.

Bootstrapping. Finally we demonstrate SRNS's self-boosting capability, by illustrating the memorization effects of CF models in Figure 3(c)-(d). To ensure a clear observation, we inject much intenser noise during sampling process, by extending $\mathcal{F}$ to 100% and 40% of the original test set on ML-100k and Ecom-toy, respectively. Under extremely noisy supervision ($\sigma = 1$), though sampling based on difficulty only ($\alpha_t = 0$), the model's test NDCG first reaches a high level and then gradually decreases, indicating that it can avoid the impact of false negative instances at an early stage.

Table 3: Performance comparison *w.r.t.* test NDCG and Recall on three datasets. The last row shows relative improvement in percentage compared with the second best.

Category	Method	Movielens-1m			Pinterest			Ecommerce		
		N@1	N@3	R@3	N@1	N@3	R@3	N@1	N@3	R@3
Non-sampling	ENMF	0.1846	0.3021	0.3882	0.2594	0.4144	0.5284	0.1317	0.2095	0.2670
Fixed Dist. Sampling	Uniform	0.1744	0.2846	0.3663	0.2586	0.4136	0.5276	0.1265	0.2057	0.2640
	NNCF	0.0829	0.1478	0.1971	0.2292	0.3699	0.4735	0.0833	0.1420	0.1855
Hard Negative Sampling	AOBPR	0.1802	0.2905	0.3728	0.2596	0.4165	0.5319	0.1293	0.2108	0.2710
	IRGAN	0.1755	0.2877	0.3708	0.2587	0.4143	0.5282	0.1275	0.2065	0.2648
	RNS-AS	0.1823	0.2932	0.3754	0.2690	0.4233	0.5359	0.1335	0.2131	0.2714
	AdvIR	0.1790	0.2941	0.3792	0.2689	0.4235	0.5363	0.1357	0.2141	0.2719
Proposed	SRNS	**0.1933**	**0.3070**	**0.3912**	**0.2891**	**0.4391**	**0.5486**	**0.1471**	**0.2256**	**0.2833**
		4.71%	1.62%	0.77%	7.47%	3.68%	2.29%	8.40%	5.37%	4.19%

7

| (a) Time, ML-1m | (b) Time, Pinterest | (c) Time, Ecom | (d) S_1, ML-1m |

| (e) S_1, Pinterest | (f) S_1, Ecom | (g) r, ML-1m | (h) r, Ecom |

Figure 4: (a)-(c) Validation NDCG vs. wall-clock time (in seconds) on three datasets. (d)-(f) Test NDCG vs. SRNS's memory size S_1, using different sampling strategies on three datasets. (g)-(h) Test NDCG and Recall of Uniform and SRNS, using two r (GMF and MLP), on ML-1m and Ecom.

4.3 Real Data Experiments

Performance comparison. As shown in Table 3, we compare SRNS with seven baselines *w.r.t.* test NDCG and Recall on three real-world datasets. As can be seen, SRNS consistently outperforms them, achieving a relative improvement of 4.71~8.40% *w.r.t.* NDCG@1. This indicates that SRNS can sample high-quality negative instances and thus helps to learn a better CF model that ranks items more accurately. Specifically, we have following three observations. First, among all baselines, hard negative sampling approaches perform more competitively. By considering both informativeness and reliability of negative instances, our SRNS outperforms two state-of-the-art baselines, *i.e.*, RNS-AS and AdvIR, that generate hard negatives based on adversarial sampling. Second, approaches using a fixed sampling distribution perform poorly, especially NNCF that directly adopts a power distribution based on item popularity. Third, by improving sample quality, sampling-based approaches can be more effective than the non-sampling counterpart that models the whole unlabeled data. For example, ENMF performs worse than RNS-AS and AdvIR on Pinterest and Ecom.

Besides effectiveness, we also compare performance in terms of efficiency, by illustrating validation NDCG vs. wall-clock time in Figure 4(a)-(c). We observe that SRNS can converge much faster and is more stable than RNS-AS and AdvIR that use GAN based structure. For fair efficiency comparison, here we also start training SRNS from the same pretrained model as in RNS-AS and AdvIR.

Robustness of variance-based sampling. With the score-based updating strategy, increasing memory size S_1 makes SRNS more prone to the false negatives. Therefore, we further test robustness of the variance-based sampling (in (4)), by evaluating performance under different S_1. As illustrated in Figure 4(d)-(f), variance-based SRNS (orange line) performs stably, indicating that emphasizing high-variance can reliably obtain high-quality samples. Comparatively, difficulty-only strategy ($\alpha_t = 0$, blue line) suffers from dramatic degradation (S_1=32 or 64). Another strategy for avoiding false negatives is to randomly select a sample from memory [46], which performs less effectively than our approach. Note that the necessity of variance-based sampling depends on the specific real-world data and, for example, Ecom may not need this *w.r.t.* overall best NDCG@1 (Figure 4(f)). Generally, our SRNS is flexible enough to switch between different situations, by controlling importance of variance-based sampling criterion (*i.e.*, α_t).

Varying scoring function. Finally we test SRNS's effectiveness on different r including GMF and MLP [19]. As illustrated in Figure 4(g)-(h), we observe similar performance improvement of SRNS over Uniform [33] when using the above two r, indicating SRNS's capability of combining with different r. We are also interested in exploring more choices like GNN-based r [43] in future study. Note that embedding dimension F is set as 32 (ML-1m), 16 (Pinterest) and 8 (Ecom), respectively, as we observe similar results with different $F \in \{8, 16, 32, 64\}$ (details are in Appendix C.4).

5 Conclusion

In this paper, we propose a simplified and robust negative sampling approach SRNS for implicit CF, which can efficiently sample true negative instances that are of high-quality. Motivated by the empirical evidence on different negative instances, our score-based memory design and variance-based sampling criterion achieve efficiency and robustness, respectively, in negative sampling. Experimental results on both synthetic and real-world datasets demonstrate SRNS's robustness and superiorities. Finally, one interesting future works would be studying the theoretical convergence guarantee of the proposed method. We will attempt to address this issue by learning from importance sampling methods [8, 47] in stochastic optimization.

References

[1] Devansh Arpit, Stanisław Jastrzębski, Nicolas Ballas, David Krueger, Emmanuel Bengio, Maxinder S Kanwal, Tegan Maharaj, Asja Fischer, Aaron Courville, Yoshua Bengio, et al. A closer look at memorization in deep networks. In *ICML*, pages 233–242, 2017.

[2] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *ICML*, pages 41–48, 2009.

[3] Rianne van den Berg, Thomas N Kipf, and Max Welling. Graph convolutional matrix completion. *arXiv preprint arXiv:1706.02263*, 2017.

[4] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In *NIPS*, pages 2787–2795, 2013.

[5] Avishek Joey Bose, Huan Ling, and Yanshuai Cao. Adversarial contrastive estimation. In *ACL*, pages 1021–1032, 2018.

[6] Hongyun Cai, Vincent W Zheng, and Kevin Chen-Chuan Chang. A comprehensive survey of graph embedding: Problems, techniques, and applications. *IEEE Transactions on Knowledge and Data Engineering*, 30(9):1616–1637, 2018.

[7] Liwei Cai and William Yang Wang. Kbgan: Adversarial learning for knowledge graph embeddings. In *NAACL*, pages 1470–1480, 2018.

[8] Haw-Shiuan Chang, Erik Learned-Miller, and Andrew McCallum. Active bias: Training more accurate neural networks by emphasizing high variance samples. In *NIPS*, pages 1002–1012, 2017.

[9] Chong Chen, Min Zhang, Yongfeng Zhang, Yiqun Liu, and Shaoping Ma. Efficient neural matrix factorization without sampling for recommendation. *ACM Transactions on Information Systems*, 38(2):1–28, 2020.

[10] Ting Chen, Yizhou Sun, Yue Shi, and Liangjie Hong. On sampling strategies for neural network-based collaborative filtering. In *KDD*, pages 767–776, 2017.

[11] 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.

[12] Marthinus C Du Plessis, Gang Niu, and Masashi Sugiyama. Analysis of learning from positive and unlabeled data. In *NIPS*, pages 703–711, 2014.

[13] Marthinus C Du Plessis, Gang Niu, and Masashi Sugiyama. Class-prior estimation for learning from positive and unlabeled data. *Machine Learning*, 106(4):463, 2017.

[14] Charles Elkan and Keith Noto. Learning classifiers from only positive and unlabeled data. In *KDD*, pages 213–220, 2008.

[15] Hongchang Gao and Heng Huang. Self-paced network embedding. In *KDD*, pages 1406–1415, 2018.

[16] Xue Geng, Hanwang Zhang, Jingwen Bian, and Tat-Seng Chua. Learning image and user features for recommendation in social networks. In *ICCV*, pages 4274–4282, 2015.

[17] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *NIPS*, pages 2672–2680, 2014.

[18] Bo Han, Quanming Yao, Xingrui Yu, Gang Niu, Miao Xu, Weihua Hu, Ivor Tsang, and Masashi Sugiyama. Co-teaching: Robust training of deep neural networks with extremely noisy labels. In *NeurIPS*, pages 8527–8537, 2018.

[19] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering. In *WWW*, pages 173–182, 2017.

[20] José Miguel Hernández-Lobato, Neil Houlsby, and Zoubin Ghahramani. Probabilistic matrix factorization with non-random missing data. In *ICML*, pages 1512–1520, 2014.

[21] Lu Jiang, Zhengyuan Zhou, Thomas Leung, Li-Jia Li, and Li Fei-Fei. Mentornet: Learning data-driven curriculum for very deep neural networks on corrupted labels. In *ICML*, pages 2304–2313, 2018.

[22] Ryuichi Kiryo, Gang Niu, Marthinus C du Plessis, and Masashi Sugiyama. Positive-unlabeled learning with non-negative risk estimator. In *NIPS*, pages 1675–1685, 2017.

[23] Yehuda Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *KDD*, pages 426–434, 2008.

[24] M Pawan Kumar, Benjamin Packer, and Daphne Koller. Self-paced learning for latent variable models. In *NIPS*, pages 1189–1197, 2010.

[25] Dawen Liang, Laurent Charlin, James McInerney, and David M Blei. Modeling user exposure in recommendation. In *WWW*, pages 951–961, 2016.

[26] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *NIPS*, pages 3111–3119, 2013.

[27] Rong Pan, Yunhong Zhou, Bin Cao, Nathan N Liu, Rajan Lukose, Martin Scholz, and Qiang Yang. One-class collaborative filtering. In *ICDM*, pages 502–511, 2008.

[28] Dae Hoon Park and Yi Chang. Adversarial sampling and training for semi-supervised information retrieval. In *WWW*, pages 1443–1453, 2019.

[29] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *KDD*, pages 701–710, 2014.

[30] Harish Ramaswamy, Clayton Scott, and Ambuj Tewari. Mixture proportion estimation via kernel embeddings of distributions. In *ICML*, pages 2052–2060, 2016.

[31] Mengye Ren, Wenyuan Zeng, Bin Yang, and Raquel Urtasun. Learning to reweight examples for robust deep learning. In *ICML*, pages 4334–4343, 2018.

[32] Steffen Rendle and Christoph Freudenthaler. Improving pairwise learning for item recommendation from implicit feedback. In *WSDM*, 2014.

[33] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. In *UAI*, pages 452–461, 2009.

[34] Emanuele Sansone, Francesco GB De Natale, and Zhi-Hua Zhou. Efficient training for positive unlabeled learning. *IEEE transactions on pattern analysis and machine intelligence*, 41(11):2584–2598, 2018.

[35] Jun Shu, Qi Xie, Lixuan Yi, Qian Zhao, Sanping Zhou, Zongben Xu, and Deyu Meng. Meta-weight-net: Learning an explicit mapping for sample weighting. In *NeurIPS*, pages 1917–1928, 2019.

[36] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: Large-scale information network embedding. In *WWW*, pages 1067–1077, 2015.

[37] 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 *SIGIR*, pages 515–524, 2017.

[38] Quan Wang, Zhendong Mao, Bin Wang, and Li Guo. Knowledge graph embedding: A survey of approaches and applications. *IEEE Transactions on Knowledge and Data Engineering*, 29(12):2724–2743, 2017.

[39] Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. Neural graph collaborative filtering. In *SIGIR*, pages 165–174, 2019.

[40] Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. Knowledge graph embedding by translating on hyperplanes. In *AAAI*, pages 1112–1119, 2014.

[41] Ga Wu, Maksims Volkovs, Chee Loong Soon, Scott Sanner, and Himanshu Rai. Noise contrastive estimation for one-class collaborative filtering. In *SIGIR*, pages 135–144, 2019.

[42] Zhen Yang, Ming Ding, Chang Zhou, Hongxia Yang, Jingren Zhou, and Jie Tang. Understanding negative sampling in graph representation learning. *arXiv*, pages arXiv–2005, 2020.

[43] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *KDD*, pages 974–983, 2018.

[44] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. In *ICLR*, 2016.

[45] Weinan Zhang, Tianqi Chen, Jun Wang, and Yong Yu. Optimizing top-n collaborative filtering via dynamic negative item sampling. In *SIGIR*, pages 785–788, 2013.

[46] Yongqi Zhang, Quanming Yao, Yingxia Shao, and Lei Chen. Nscaching: Simple and efficient negative sampling for knowledge graph embedding. In *ICDE*, pages 614–625, 2019.

[47] Peilin Zhao and Tong Zhang. Stochastic optimization with importance sampling for regularized loss minimization. In *ICML*, pages 1–9, 2015.

[48] Qian Zhao, Martijn C Willemsen, Gediminas Adomavicius, F Maxwell Harper, and Joseph A Konstan. Interpreting user inaction in recommender systems. In *Recsys*, pages 40–48, 2018.

A Comparison Between Different Approaches

A.1 General Machine Learning Approaches

Learning an implicit CF model from the positive-only data is also related to Positive-Unlabeled (PU) learning and learning from noisy labels, as the rest unobserved instances are unlabeled and noisy. Motivated by these general machine learning approaches, this paper formulates the negative sampling problem as efficient learning from unlabeled data with the presence of noisy labels, and pays more attention on those true negative instances hidden inside the massive unlabeled data. The following table and review on literatures discuss the differences between different approaches that can be adapted for this problem.

Approaches	Learning from Positive-Unlabeled Data	Learning from Noisy Labels	Negative Sampling
Positive/Negative Class Prior	known or estimated from data [13, 30]	unknown	unknown
Assumption on Unlabeled Data	positive or negative labels [14, 22]	negative labels with noise [21, 31]	unobserved [26, 29, 33]
Handling Uncertainty of Unlabeled Data	minimizing the empirical risk estimator [12, 22]	manually designing [2, 24] or automated learning the instance weight [21, 31, 35]	sampling unobserved instances as negative labels [26, 29, 33]

Learning from Positive-Unlabeled Data. Since implicit feedback data contain positive instances only, the implicit CF problem is also related to learning from positive-unlabeled (PU) data. PU learning formulates the problem as a binary classification, accounting for the fact that both positive and negative labels exist in the unlabeled data [12, 14, 22]. However, it normally requires an accurate estimation of the class-prior, which is challenging in real-world data [13, 30]. Moreover, a direct optimization on the whole unlabeled data is generally inefficient, especially for implicit CF, where an efficient training approach supporting large-scale data is necessary [22, 34]. In our proposed solution, above issues are avoided by efficiently sampling negative instances from the unlabeled data and, motivated by the idea of PU learning, we carefully distinguish those true negative instances from others.

Learning from Noisy Labels. By regarding unobserved instances as a combination of true negative labels and noisy labels, another choice is adapting the implicit CF into learning from noisy labels. Typical learning approaches include curriculum learning [2], self-pace learning [24] and instance re-weighting [21, 31, 35]. The first two approaches prefer easier instances during training process so as to improve robustness, while these easy instances may be ineffective for learning a CF model. Without prior information about the noisy labels, instance re-weighting approach learns the weight of each instance with bi-level optimization on training and validation data [21, 31, 35]. However, the size of unlabeled data in implicit CF can approach to nearly a product of user count and item count, making above non-sampling approach become unaffordable in terms of learning efficiency. Therefore, this work focuses on negative sampling and aims to handle noisy labels correctly at the same time.

A.2 Specific Negative Sampling Approaches

Negative sampling approaches have also been widely adopted in other domains of embedding learning for text, graph, etc. Motivated by these works that tend to leverage a simple model for capturing negative sampling distribution, we design a memory-based model that simply maintains the promising candidates with large scores. More importantly, we propose to robustify negative sampling by emphasizing high-variance samples, which is novel in both CF and other domains. The following table and review on literatures discuss the differences between different approaches.

Negative Sampling in Other Domains. Negative sampling approaches are widely used in many tasks like word embedding [26], graph embedding [6] and knowledge graph embedding [38]. In terms of capturing the distribution of negative instances, these applications generally requires a rather simple model. For example, Word2Vec [26] sets the negative sampling distribution proportional to the 3/4 power of word frequency, which favours those frequent words. Later works on graph

Domain	Text	Graph	Knowledge Graph	Collaborative Filtering
Learning Objective	semantic word relationships	node proximities	fact composed of head/tail entity and relation	user preferences among items
Vanilla Sampling Strategy	frequency-based [26]	degree-based [29, 36]	uniform [4], bernoulli [40]	uniform [33], popularity-based [10, 41]
Improving Sample Quality	GAN [5]	self-paced learning, GAN [15]	score-based [46], GAN [5, 7]	score-based [32, 45], GAN [11, 28]
Leveraging Skewness in Distribution	favouring frequent words [26]	favouring high-degree nodes [29, 36] or positive-alike nodes [42]	favouring large-scored instances [46]	none
Handling False Negative	none	none	none	avoiding the hardest instances [43]

embedding [29, 36] readily keep this skewed distribution by adapting it to the node degree. Similarly in knowledge graph, it has been observed that negative instances with large scores are important but rare and focusing on this partial set makes the model much simpler [46]. Another recent work on negative sampling of graph representation learning proposes that the negative sampling distribution should be positively but sub-linearly correlated to their positive sampling distribution [42]. However, in terms of avoiding false negative instances, none of them have tackled this problem by designing a robust sampling approach. Since the implicit CF is a different problem where the reliability of sampled negative instances is much harder to guarantee, we propose to reduce this risk by emphasizing high variance samples. Meanwhile, motivated by above examples in other domains, we also leverage a simple model to efficiently capture the distribution of negative instances which are of high-quality.

B Implementation Details

B.1 Running Environment

The experiments are conducted on a single Linux server with AMD Ryzen Threadripper 2990WX@3.0GHz, 128G RAM and 4 NVIDIA GeForce RTX 2080TI-11GB. Our proposed SRNS is implemented in Tensorflow 1.14 and Python 3.7.

B.2 Baselines

We compare the SRNS with following state-of-the-art approaches: (1) Uniform [33], which uniformly selects negative samples from the unlabeled data. (2) NNCF [10], which uses a negative sampling distribution proportional to the 3/4 power of item popularity. A hyper-parameter s is the number of positive samples per item. b is the number of negative samples per positive sample. (3) AOBPR [32], which improves uniform strategy by adaptively oversampling hard instances. A hyper-parameter λ controls the skewness of distribution $\propto \exp(-\text{rk}(j|u)/\lambda)$. (4) IRGAN [37], which uses an adversarial sampler by conducting a minimax game between the recommender and the sampler. A hyper-parameter τ is the temperature in sampling distribution (Eq. (10) in [37]). (5) RNS-AS [11], which leverages adversarial sampling to generate hard negative samples. A hyper-parameter N_s is size of candidate set for sampling and τ is the temperature. (6) AdvIR [28], which exploits both adversarial sampling and training (*i.e.*, adding perturbation) to generate better negative samples. N_s and τ are defined similarly as above. ϵ controls the perturbation size. (7) ENMF [9], as a baseline, we also compare with an non-sampling approach that regards all the unlabeled data as negative labels and carefully assigns instance weights. A hyper-parameter c controls above weight for a negative instance.

B.3 Detail of MLP based r

The MLP based scoring function $r(\mathbf{p}_u, \mathbf{q}_i, \boldsymbol{\beta})$ [19] takes the concatenation of $\mathbf{p}_u$ and $\mathbf{q}_i$, i.e., $\mathbf{z}_0 = [\mathbf{p}_u; \mathbf{q}_i] \in \mathbb{R}^{2F}$, as the input. Then there are H hidden layers, and the lth layer is defined as

$$\mathbf{z}_l = \text{sigmoid}(\mathbf{W}_l \mathbf{z}_{l-1} + \mathbf{b}_l), \tag{5}$$

where $\mathbf{W}_l \in \mathbb{R}^{d_l \times d_{l-1}}$ and $\mathbf{b}_l \in \mathbb{R}^{d_l}$ denote the weight matrix and bias vector in this layer. Specifically, $d_0 = 2F$ and we set $d_l = \frac{1}{2} d_{l-1}$. The last layer outputs the prediction score r_{ui}, defined as

$$r_{ui} = \mathbf{W}_{H+1}^\top \mathbf{z}_H + \mathbf{b}_{H+1}, \tag{6}$$

where $\mathbf{W}_{H+1} \in \mathbb{R}^{d_H}$ and $\mathbf{b}_{H+1} \in \mathbb{R}$. The learnable parameters $\boldsymbol{\beta}$ in this MLP based r are $\{\mathbf{W}_i, \mathbf{b}_i\}(i = 1, ..., H+1)$.

B.4 Hyper-parameter Tuning

Our SRNS's hyper-parameters can be divided into three parts: (1) sampling related part, including memory size S_1, expansion size S_2, temperature τ, variance-based criterion weight α, warm-start epoch number T_0. (2) r related part, including embedding dimension F and hidden layer number H. (3) optimization related part, including learning rate lr and L2 regularization reg.

In synthetic noise experiments, since we do not explicitly split a validation set on synthetic data, we draw two different train/test splits. The hyper-parameters are searched in the first round and afterwards are kept constant in another round. Note that the false negative instances ($\mathcal{F}$) in there two rounds are also independent with each other, as they are simulated by random sampling from the corresponding test set. We run each synthetic data experiment 400 epochs without early stopping and repeat five times. The scoring function r is GMF [19]. The memory size S_1/S_2 are fixed as 20/20. The temperature τ is 1. Adam optimizer with $\beta_1 = 0.9$, $\beta_2 = 0.999$ is used and the mini-batch size is set to 1024. The lazy-update epoch number $E = 1$. The rest hyper-parameters are tuned according to average NDCG@3 in the last 50 training epochs. Specifically, first we use grid search to find the best group of non-sampling related hyper-parameters, i.e., (F, lr, reg), using the vanilla Uniform method [33] as the negative sampling strategy. Then we fix (F, lr, reg) and search the rest sampling related hyper-parameters, i.e., (α, T_0), under different settings of noisy supervision (σ). See Table 4 for detailed information.

Table 4: SRNS's hyper-parameter exploration in synthetic noise experiments (Section 4.2)

Hyper-parameter	Tuning Range	Opt. (Ecom-toy)	Opt. (ML100k)
lr	$[5, 10, 50] \times 10^{-4}$	0.001	0.001
reg	$[0, 1, 10] \times 10^{-3}$	0.0	0.001
F	$[8, 16, 32]$	32	8
α	$[5.0, 10.0, 20.0, 50.0]$	-	-
T_0	$[50, 100]$	-	-

In real data experiments, we conduct the standard procedure to split train/validation/test set. We run 400 epochs and terminate training if validation performance does not improve for 100 epochs, which has also been repeated five times. Both GMF and MLP (defined in Appendix B.3) are tested. Adam optimizer with $\beta_1 = 0.9$, $\beta_2 = 0.999$ is used and the mini-batch size is set to 1024. The lazy-update epoch number $E = 1$. The embedding dimension F is set as 32 (ML-1m), 16 (Pinterest) and 8 (Ecom), respectively. We further show similar results with different $F \in \{8, 16, 32, 64\}$ in Appendix C.4. The rest hyper-parameters are tuned according to the best NDCG@1 on the validation set. Specifically, first we use grid search to find the best group of non-sampling related hyper-parameters, i.e., (lr, reg), using the vanilla Uniform method [33] as the negative sampling strategy (For MLP based r we also search H). Then we fix them and search the rest sampling related hyper-parameters, i.e., $(\tau, \alpha, T_0, S_1, S_2/S_1)$. To ease the tuning process, we first fix α and T_0 as 0 (difficulty-only sampling), then search the best $(\tau, S_1, S_2/S_1)$. After that we fix them and search the best group of (α, T_0) (variance-based sampling). Also, we repeat above step by changing memory size S_1 to its next or previous value. For example, if current best S_1 is 16, we further test 8 and 32. Note that we do not search sampling related hyper-parameters when using MLP based r, by directly using those for GMF. See Table 5 for detailed information.

15

Table 5: SRNS's hyper-parameter exploration in real data experiments (Section 4.3)

	Para.	Tuning Range	Opt. (Ecom)	Opt. (ML1m)	Opt. (Pinterest)
SRNS GMF	lr	$[5, 10, 50, 100] \times 10^{-4}$	0.001	0.001	0.001
	reg	$[0, 1, 10, 100] \times 10^{-4}$	0.001	0.01	0.0
	τ	$[0.5, 1.0, 2.0, 10.0]$	2.0	10.0	10.0
	α	$[0.1, 1.0, 2.0, 5.0, 10.0, 20.0, 50.0]$	0.0	5.0	5.0
	T_0	$[25, 50, 100]$	25	50	50
	S_1	$[2, 4, 8, 16, 32]$	8	8	16
	S_2/S_1	$[1, 2, 4, 8]$	2	8	4
SRNS MLP	lr	$[5, 10, 50] \times 10^{-4}$	0.001	0.001	-
	reg	$[0, 1, 10, 100] \times 10^{-4}$	0.001	0.01	-
	H	$[0, 1, 2, 3]$	3	3	-

Table 6: Baselines' hyper-parameter exploration in real data experiments (Section 4.3)

Method	Para.	Tuning Range	Opt. (Ecom)	Opt. (ML1m)	Opt. (Pinterest)
Uniform	lr	$[5, 10, 50, 100] \times 10^{-4}$	0.001	0.001	0.001
	reg	$[0, 1, 10, 100] \times 10^{-4}$	0.001	0.01	0.0
NNCF	lr	$[5, 10, 50, 100] \times 10^{-4}$	0.001	0.001	0.001
	reg	$[0, 1, 10, 100] \times 10^{-4}$	0.0	0.0	0.0
	b	$[32, 64, 128, 256, 512, 1024, 2048]$	32	2048	2048
	s	$[1, 2, 4]$	2	2	2
ENMF	lr	$[5, 10, 50, 100] \times 10^{-4}$	0.01	0.01	0.005
	reg	$[0, 1, 10, 100] \times 10^{-4}$	0.001	0.0001	0.0
	c	$[0.01, 0.03, 0.05, 0.07, 0.1, 0.3, 0.5, 0.7]$	0.1	0.3	0.01
AOBPR	lr	$[5, 10, 50, 100] \times 10^{-4}$	0.0005	0.0005	0.0005
	reg	$[0, 1, 10, 100] \times 10^{-4}$	0.001	0.01	0.0
	λ	$[5, 10, 20, 50, 100, 200, 500, 1000, 2000]$	10	1000	2000
IRGAN	lr	$[5, 10, 50, 100] \times 10^{-4}$	0.0005	0.0005	0.0005
	reg	$[0, 1, 10, 100] \times 10^{-4}$	0.001	0.001	0.001
	τ	$[0.5, 1.0, 2.0]$	2.0	1.0	1.0
RNS-AS	lr	$[5, 10, 50, 100] \times 10^{-4}$	0.001	0.0005	0.0005
	reg	$[0, 1, 10, 100] \times 10^{-4}$	0.0	0.001	0.01
	τ	$[0.5, 1.0, 2.0, 10.0]$	1.0	0.5	0.5
	N_s	$[10, 20, 30, 40]$	10	10	10
AdvIR	lr	$[5, 10, 50, 100] \times 10^{-4}$	0.0005	0.0005	0.0005
	reg	$[0, 1, 10, 100] \times 10^{-4}$	0.0	0.0001	0.001
	ϵ	$[1, 10, 100] \times 10^{-2}$	0.01	0.01	0.01
	τ	$[0.5, 1.0, 2.0, 10.0]$	1.0	1.0	1.0
	N_s	$[10, 20, 30, 40]$	10	10	10

As for the baselines listed in Appendix B.2, except Uniform [33] that have been tuned as above, others have also been carefully tuned according to their validation NDCG@1. For IRGAN, RNS-AS and AdvIR using GAN-based structure, we use a pretrained model (*i.e.*, trained under Uniform) to initialize. See Table 6 for detailed information.

B.5 Evaluation Metrics

As defined in Section 4.1, our used metrics, *i.e.*, Recall and NDCG, can provide a comprehensive evaluation of model performance. The former measures whether the ground truth item is presented on the ranked list, while the latter measures the performance at a finer granularity by accounting for the position of hit. The two datasets (ML-100k and Ecom-toy) used in synthetic noise experiments are rather small, with the item count $|\mathcal{I}|$ between 1000~2000, while the rest three in real data experiments are much larger, with the highest value as 59290 (Ecom). Thus in real data experiments, we follow

a common strategy [19, 23] to fix the list length $|\mathcal{S}_u|$ as 100, by randomly sampling $100 - |\mathcal{G}_u|$ non-interacted items, because ranking the whole item set for each user is too time-consuming during evaluation. When reporting NDCG@k and Recall@k, we choose a rather small value of truncated length $k \in \{1, 3\}$, because of following two reasons: (1) In real applications of implicit CF like recommender systems, users tend to browse the items at first few positions of a list, making the accuracy of rest recommended items less important. (2) In real data experiments we fix the length of a ranked list as 100, thus choosing a large k may make this task too easy.

B.6 Variance Computation

To calculate the prediction variance $\text{std}[P_{\text{pos}}(k|u, i)]$ (Eq. (4)) of each candidate instance (u, k) stored in the memory $\mathcal{M}_u$, we directly use the prediction results from previous iterations, without any extra forward or backward passes in the r. In our implementation, we consider the prediction probability in the latest few epochs, which is due to following two reasons: (1) prediction history near the beginning period of training process is not stable for all kinds of negative instances, and thus can be excluded from the computation. (2) this implementation makes the overhead constant ($O(1)$) for each sampling operation. In our experiments, we determine the uncertainty only based on the latest 5 epochs. Specifically, at tth training epoch,

$$
\begin{aligned}
\text{std}[P_{\text{pos}}(k|u, i)] &= \sqrt{\textstyle\sum_{s=t-5}^{t-1}\big[[P_{\text{pos}}(k|u,i)]_s - \text{Mean}[P_{\text{pos}}(k|u,i)]\big]^2 \big/ 5}, \\
\text{Mean}[P_{\text{pos}}(k|u, i)] &= \textstyle\sum_{s=t-5}^{t-1}[P_{\text{pos}}(k|u,i)]_s \big/ 5.
\end{aligned}
\tag{7}
$$

In real data experiments where the datasets are much larger, it is time-consuming to compute the prediction probability (P_{pos}) for all user-item pairs ($|\mathcal{U}| \cdot |\mathcal{I}|$) at each epoch. Thus we prune the item space for each user's memory update process, so as to avoid logging P_{pos} for all items. Specifically, for u at tth training epoch, the newly extended candidates in $\mathcal{M}_u$ can only be randomly sampled from an item set, denoted as var_set_u, which has already been generated at $(t - 5)$th epoch. At the mean time, for $v \in var\_set_u$, we log $P_{\text{pos}}(v|u, i)$ values at the subsequent 5 epochs. Therefore, among u's memory $\mathcal{M}_u$, besides the original items that have been maintained from previous epochs, the newly added items also have the P_{pos} history in the latest 5 epochs, which supports the variance computation above. Note that var_set_u is also generated by random sampling from u's non-interacted items, and its size is larger than memory size S_1, but much smaller than item count $|\mathcal{I}|$. We fix $|var\_set_u|$ as 3000 (ML-1m) and 600 (Pinterest, Ecom), respectively.

C Experiment Details

C.1 Dataset Description

We choose following four raw datasets and build five datasets for performance evaluation.

- **Movielens (ML)-100k**[2]. This is a widely used movie-rating dataset containing 100,000 ratings on movies from 1 to 5. We follow the common preprocessing to convert it into implicit feedback data, regarding those high-rated records ($4 \sim 5$) as positive labels [28, 37].

- **Movielens (ML)-1m**[3]. Similarly to ML-100k, this large dataset contains 1,000,000 ratings. After similar converting procedure, we filter out users with less than 5 records.

- **Pinterest**[4]. This implicit feedback dataset is constructed by [16] for a task of image recommendation, and has been used for evaluating the implicit CF task [19].

- **Ecommerce (Ecom)**. This implicit feedback dataset is a subset of users' item-click records in a real-word E-commerce website between 2017/06 and 2017/07. For data preprocessing, we filter out users/items with less than 4 records, so as to overcome the problem of high sparsity. After that, we further obtain a toy dataset, denoted as **Ecom-toy**, by retaining top 1,000 users and 2,000 items sorted by number of records.

---

[2]https://grouplens.org/datasets/movielens/100k
[3]https://grouplens.org/datasets/movielens/1m
[4]https://pinterest.com

C.2 Details of Figure 1

The experiment is conducted on ML-100k dataset, using the same train/test split as synthetic noise experiments. We use GMF as the r and Uniform [33] as the negative sampling strategy. By flipping labels of groundtruth records in the test set, we are able to obtain a set of false negative instances (FN) that are in fact positive labeled but unobserved during the negative sampling process. Besides uniformly sampling negative instances (UN) to update the model, we simultaneously obtain a series of hard negative instances (HN) with different difficulty D. In following analysis, we adopt a simple yet effective strategy to control D of a obtained HN: 1) uniformly sample D candidates from $\{(u,j)|j \notin \mathcal{R}_u\}$; 2) select the negative instance with the highest value of r_{uj}. When D gets higher, HN becomes much harder. UN is the same as HN with $D = 1$.

As in Figure 1(a), we have a closer look at the negative instances' distribution in terms of their positive-label probabilities P_{pos} that are proportional to the prediction scores. This is motivated by [46] that has observed a skewed distribution of negative instances when learning knowledge graph embeddings. Specifically, (a) is the distribution of negative instances $\{(u,j)|u \in \mathcal{U}, j \notin \mathcal{R}_u\}$ at 5 timestamps. We measure the *complementary cumulative distribution function* (CCDF) $F(x) = P(P_{\text{pos}} \geq x)$ to show the proportion of negative instances that satisfy $P_{\text{pos}} \geq x$. Since hard negative instances generally have large P_{pos}, we compare them with those false negative instances *w.r.t.* P_{pos} (Figure 1(b)). We use the median value ($p50$) to represent each set. Then in Figure 1(c), we further analyze the possibility of using P_{pos} to discriminate above two sets of negative instances. Specifically, under different hard negative sampling strategies, we calculate the *label error ratio* in each mini-batch, *i.e.*, $LER = (\#\ of\ false\ negative\ samples)/(\#\ of\ all\ selected\ negative\ samples)$. Unlike others, false negative instances follow the similar distribution as those positive instances in training data. Thus the model can ideally become more and more confident about predicting them as positive instances, and the corresponding variance of P_{pos} is low. Finally, to validate this, we compare P_{pos}'s variance between different types of negative instances in Figure 1(d). The normalized variance is measured by the ratio between standard deviation and mean value.

C.3 Synthetic Noise Experiments

To control the impact of false negative instances on the sampling process, we manually inject noisy labels by slightly modifying each user's memory $\mathcal{M}$ that stores S_1 candidate negative instances. Specifically, for user u, there is always an instance in $\mathcal{M}_u$ that is randomly sampled from u's false negative set $\mathcal{F}_u$, and this instance is also dynamically updated together with $\mathcal{M}_u$. As for the rest $S_1 - 1$ candidates in $\mathcal{M}_u$, they cannot be selected from $\mathcal{F}_u$. To control the noise ratio, we vary the size of false negative set by randomly sampling $\sigma \times 100$ (%) from $\mathcal{F}_u$ ($\sigma \in [0, 1]$). Note that

Figure 5: Detailed results of Figure 3: Test NDCG vs. number of epochs on two datasets, with the error bar for STD highlighted as a shade.

18

$\sigma = 0$ indicates an "ideal" case where $\mathcal{M}_u$ is not influenced by $\mathcal{F}_u$. In these experiments, we fix the memory size S_1 as 20.

Note that in the "ideal" case with no explicit noise, SRNS still largely outperforms in Ecom-toy dataset, which is also reasonable given the fact that $\mathcal{F}$ cannot ideally cover all the false negative instances hidden in unlabeled data.

In each figure of Figure 5, the blue curve represents the result of difficulty-only sampling strategy, while the grey curve and orange curve both represent those of the SRNS, with the difference on whether to linearly increase weight α_t during training process. It can be clearly observed that the "warm-start" setting of α_t performs better than a fixed-value setting, as the former better leverages prediction variance after false negative instances become stable. More detailed investigation on different settings of α_t are shown in following two tables.

Table 7: Detailed investigation of "warm-start" on ML-100k, $\sigma = 1.0$ (Figure 3(a)).

	T_0/α	5	10	20	50
Flat	0	0.3703±0.0033	0.3811±0.0048	0.3876±0.0054	0.4004±0.0112
Increased	50	0.3734±0.0045	0.3931±0.0097	0.3924±0.0050	0.3965±0.0099
	100	0.3725±0.0111	0.3850±0.0075	**0.4062±0.0073**	0.3844±0.0078
Decreased	50	0.3631±0.0066	0.3677±0.0049	0.3700±0.0064	0.3623±0.0108
	100	0.3620±0.0063	0.3650±0.0039	0.3710±0.0062	0.3719±0.0055

Table 8: Detailed investigation of "warm-start" on Ecom-toy, $\sigma = 0.5$ (Figure 3(b).

	T_0/α	5	10	20	50
Flat	0	0.2449±0.0052	0.2557±0.010	0.2525±0.0063	0.2343±0.0019
Increased	50	0.2574±0.0051	0.2702±0.0048	0.2515±0.0053	0.2329±0.0090
	100	0.2464±0.0051	**0.2581±0.0072**	0.2636±0.0091	0.2267±0.0092
Decreased	50	0.2037±0.0064	0.2351±0.0081	0.2367±0.0091	0.2365±0.0110
	100	0.2120±0.0029	0.2351±0.0062	0.2348±0.0051	0.2513±0.0053

C.4 Real Data Experiments

Figure 6 shows test NDCG of Uniform and SRNS approaches using different embedding size F. The scoring function r is GMF. Again we can observe consistent improvement of SRNS over Uniform when $F \in \{8, 16, 32, 64\}$. Although increasing F should have improved performance, we observe instead that $F = 16$ performs the best on Pinterest dataset, which conforms to a previous work (Figure 4 in [19]).

Figure 7 shows supplementary results, *w.r.t.* NDCG@3 and Recall@3, of Figure 4(d)-(f), which are similar to those findings *w.r.t.* NDCG@1.

(a) F, N@1, Pinterest   (b) F, N@3, Pinterest   (c) F, R@3, Pinterest

(d) F, N@1, Ecom   (e) F, N@3, Ecom   (f) F, R@3, Ecom

Figure 6: Varying embedding dimension F: Test NDCG/Recall of Uniform and SRNS approaches, using different embedding size F, on Pinterest and Ecom, respectively.

(a) S_1, N@3, ML-1m   (b) S_1, R@3, ML-1m   (c) S_1, N@3, Pinterest

(d) S_1,R@3, Pinterest   (e) S_1, N@3, Ecom   (f) S_1, R@3, Ecom

Figure 7: Detailed results of Figure 4(e) and (f): Test NDCG@3/Recall@3 vs. SRNS's memory size S_1, using different sampling strategies on three datasets.