Locality-Sensitive State-Guided Experience Replay Optimization for Sparse Rewards in Online Recommendation

Xiaocong Chen
xiaocng.chen@unsw.edu.au
University of New South Wales
Sydney, NSW, Australia

Lina Yao
lina.yao@unsw.edu.au
University of New South Wales
Sydney, NSW, Australia

Julian McAuley
jmcauley@eng.ucsd.edu
University of California, San Diego
CA, USA

Weili Guan
Weili.Guan@monash.edu
Monash University
Melbourne, VIC, Australia

Xiaojun Chang
xiaojun.chang@uts.edu.au
University of Technology Sydney
Sydney, NSW, Australia

Xianzhi Wang
xianzhi.wang@uts.edu.au
University of Technology Sydney
Sydney, NSW, Australia

ABSTRACT

Online recommendation requires handling rapidly changing user preferences. Deep reinforcement learning (DRL) is an effective means of capturing users’ dynamic interest during interactions with recommender systems. Generally, it is challenging to train a DRL agent in online recommender systems because of the sparse rewards caused by the large action space (e.g., candidate item space) and comparatively fewer user interactions. Leveraging experience replay (ER) has been extensively studied to conquer the issue of sparse rewards. However, they adapt poorly to the complex environment of online recommender systems and are inefficient in learning an optimal strategy from past experience. As a step to filling this gap, we propose a novel state-aware experience replay model, in which the agent selectively discovers the most relevant and salient experiences and is guided to find the optimal policy for online recommendations. In particular, a locality-sensitive hashing method is proposed to selectively retain the most meaningful experience at scale and a prioritized reward-driven strategy is designed to replay more valuable experiences with higher chance. We formally show that the proposed method guarantees the upper and lower bound on experience replay and optimizes the space complexity, as well as empirically demonstrate our model’s superiority to several existing experience replay methods over three benchmark simulation platforms.

CCS CONCEPTS

• Information systems → Recommender systems; • Computing methodologies → Reinforcement learning.

KEYWORDS

Recommender Systems, Deep Reinforcement Learning, Experience Replay

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

SIGIR '22, July 11–15, 2022, Madrid, Spain.
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-8732-3/22/07...
https://doi.org/10.1145/3477495.3532015

1 INTRODUCTION

Online recommendation aims to learn users’ preferences and recommend items dynamically to help users find desired items in highly dynamic environments [37]. Deep reinforcement learning (DRL) naturally fits online recommendation as it learns policies through interactions with the environment via maximizing a cumulative reward. DRL has been widely applied to sequential decision-making (e.g., in Atari [23] and AlphaGo [32]). A considerable amount of literature has been published on deep reinforcement learning-based dynamic recommender systems [2, 4, 39].

DRL-based recommender systems cover three categories of methods: deep Q-learning (DQN), policy gradient, and hybrid methods. DQN aims to find the best step via maximizing a Q-value over all possible actions. Zheng et al. [42] introduced DRL into recommender systems for news recommendation; Chen et al. [4] introduced a robust reward function to Q-learning, which stabilized the reward in online recommendations. Despite the capability of fast indexing in selecting a discrete action, Q-learning-based methods conduct the “maximize” operation over the action space (i.e., all available items) and suffer from the stuck agent problem—a considerable amount of literature has been published on deep reinforcement learning-based dynamic recommender systems [2, 4, 39].

DQN and policy gradient are more suitable for small action and state spaces [20, 34] in a recommendation context, hybrid methods [3, 8, 13, 40] have the capability to map large high-dimensional discrete state spaces into low-dimensional continuous spaces via combining the advantages of Q-learning and policy gradient. A typical hybrid method is the actor-critic method [19], which adopts policy gradient on an actor-network and Q-learning on a critic network to achieve Nash equilibrium on both networks. Actor-critic networks have been widely applied to DRL-based recommender
We present a novel experience replay structure, Locality-Sensitive Experience Replay (LSER), to address the above challenges. Differing from existing approaches, which apply random or uniform sampling, LSER samples experiences based on expected states. Inspired by collaborative filtering (which measures the similarity between users and items to make recommendations) and AER [33], LSER only replays experience from similar states to improve the sampling efficiency. Specifically, we introduce a ranking mechanism to prioritize replays and promote higher-reward experiences. We further use the $\epsilon$-greedy method to avoid replaying high-reward states excessively.

Considering the high-dimensionality of vectorized representations of states [6], we convert similarity measurements for high-dimensional data into a hash key matching problem and employ locality-sensitive hashing to transform states into low-dimensional representations. Then, we assign similar vectors the same hash codes (based on the property of locality-sensitive hashing). Such a transformation reduces all the states into low dimension hash keys.

In summary, we make the following contributions:

- We propose a novel experience replay method (LSER) for reinforcement-learning-based online recommendation. It employs a similarity measurement to improve training efficiency.
- LSER replays experience based on the similarity level of the given state and the stored states; the agent thus has a higher chance to learn valuable information than it does with uniform sampling.
- The experiments on three platforms, VirtualTB, RecSim and RecoGym, demonstrate the efficacy and superiority of LSER to several state-of-the-art experience replay methods.

2 METHODOLOGY

In this section, we will briefly introduce the proposed LSER method with theoretical analysis. The overall structure of using LSER in DRL RS can be found in Figure 1.

2.1 Overview

Online Recommendation aims to find a solution that best reflects real-time interactions between users and the recommender system and apply the solution to the recommendation policy. The system needs to analyze users’ behaviors and update the recommendation policy dynamically. In particular, reinforcement learning-based recommendation learns from interactions through a Markov Decision Process (MDP).

Given a recommendation problem consisting of a set of users $U = \{u_0, u_1, \cdots, u_k\}$, a set of items $I = \{i_0, i_1, \cdots, i_l\}$ and user attributes $D = \{d_0, d_1, \cdots, d_m\}$, MDP can be represented as a tuple $(S, A, P, R, \gamma)$, where $S$ denotes the state space (i.e., the combination of the subsets of $I$ and its corresponding user information) $A$ denotes the action space, which represents the agent’s selection during recommendation based on the state space $S$, $P$ denotes the set of transition probabilities for state transfer based on the action received, $R$ is a set of rewards received from users, which are used to evaluate the action taken by the recommender system (each reward is a binary value to indicate whether user has clicked the...
We formulate the storage and sampling issue in LSER as a similarity measure. A popular way of searching for similar high-dimensional vectors is to introduce the formal definition of the collision probability for LSH.

Then, we theoretically analyze the collision probability for the $p$-stable distribution to prove that our method has a reasonable boundary for collision probability.

**Definition 1** (Collision probability for LSH in $p$-stable distribution). Given an LSH function $h_{ab} \in \mathcal{H}$ and the probability density function (PDF) of the absolute value of the $p$-stable $(p \in [1, 2])$ distribution $f_p(t)$ in $L^p$ space, the collision probability for vectors $u$ and $v$ is represented by:

$$P = Pr[h_{ab}(u) = h_{ab}(v)] = \int_0^w \frac{1}{c} f_p\left(\frac{t}{c}\right) \left(1 - \frac{t}{w}\right) dt$$

where $c = \|u - v\|_p$ and $w$ is a user-defined fixed distance measure.

Here, we use a 2-state distribution, i.e., normal distribution for dimensionality reduction. We randomly initialize $n_h$ hyperplanes based on the normal distribution on the projective space $P^n$ to get the hash representation for a given state $s$, where $n$ is the dimension of the state. The hashing representation $h(s)$ for the given state $s$ is calculated as follows:

$$h_{p \in P^n}(s) = \{0, 1\}^n \text{ with } \begin{cases} 0 & p_i \cdot s_i > 0 \\ 1 & p_i \cdot s_i \leq 0 \end{cases}$$

The collision probability of the above method can be represented as:

$$P = Pr[h_{p \in P^n}(u) = h_{p \in P^n}(v)] = 1 - \frac{\text{Ang}(u, v)}{\pi} \text{ where } \text{Ang}(u, v) = \arccos \frac{|u \cap v|}{\sqrt{|u||v|}}.$$  

Equation (2) formulates the information loss during the projection, where we use term $e$ to represent the quantification between the real value $p \cdot v$ and hashed results induced from $h(v)$. Since the relative positions in the original space are preserved during the hash transformation with an extra measurement $e$, the upper bound and lower bound of collision probability boundary in projective space is guaranteed to be maintained by introducing an extra parameter $e$. That means the more dissimilar states will not receive a higher probability to be allocated to the same hash result.

**Lemma 1.** Given an arbitrary hash function $h_{ab} \in \mathcal{H}$, the collision probability for a given vector $u$ and $v$ is bounded at both ends.

**Proof.** Since $Pr[h_{ab}(u) = h_{ab}(v)]$ monotonically decreases in $c$ for any hash function from the LSH family $\mathcal{H}$, the collision probability is bounded from above by $Pr[h_{ab}(u) = h_{ab}(v)]$ for $c - e$ and from below by $Pr[h_{ab}(u) = h_{ab}(v)]$ for $c + e$.

$$P = \int_0^w \frac{1}{c} f_p\left(\frac{t}{c}\right) \left(1 - \frac{t}{w}\right) dt = \int_0^{w/c} f_p(q) \left(1 - \frac{qc}{w}\right) dq \text{ with } q = \frac{t}{c}.$$

Then, we have the upper bound:

$$\int_0^{w/(c-e)} f_p(q) \left(1 - \frac{(c-e)q}{w}\right) dq \text{ with } q = \frac{t}{c-e}$$

$$= \int_0^{w/(c-e)} f_p(q) \left(1 - \frac{q}{w}\right) + \frac{qe f_p(q)}{w} dq$$

$$\leq P + \frac{e}{w} \int_0^{w/(c-e)} q f_p(q) dq \leq P + \frac{e}{c-e}.$$
and the lower bound:
\[
\int_0^{w/(c+\epsilon)} f_p(q) \left(1 - \frac{(c + \epsilon)q}{w}\right) dq \text{ with } q = \frac{t}{c + \epsilon} \\
= \int_0^{w/(c+\epsilon)} f_p(q) \left(1 - \frac{qc}{w}\right) dq \text{ for } q \geq \frac{t}{c + \epsilon} \\
= -P - \frac{e}{w} \int_0^{w/(c+\epsilon)} q f_p(q) dq - \int_{w/(c+\epsilon)}^{w/c} f_p(q) \left(1 - \frac{qc}{w}\right) dq \\
\geq P - \frac{e}{c + \epsilon} - \left(1 - \frac{c}{c + \epsilon}\right) = P - \frac{2e}{c + \epsilon}
\]

We compute the upper bound based on Hölder's inequality in $L^1$ space:
\[
\int_0^{w/(c-\epsilon)} q f_p(q) dq \leq \left(\sup_{q \in [0,w/(c-\epsilon)]} q\right) \|f_p\|_1 \leq \frac{w}{c - \epsilon}
\]

Considering the $L^\infty$ space, we have:
\[
\int_0^{w/(c-\epsilon)} q f_p(q) dq \leq \|f_p\|_\infty \int_0^{w/(c-\epsilon)} q dq = \frac{w^2 \|f_p\|_\infty}{2(c - \epsilon)^2}
\]

We use the similar method in $L^1$ to compute the lower bound:
\[
\int_{w/(c+\epsilon)}^{w/c} f_p(q) \left(1 - \frac{qc}{w}\right) dq \leq \left(\sup_{q \in [w/(c+\epsilon),w/c]} \left(1 - \frac{qc}{w}\right)\right) \|f_p\|_1
\leq 1 - \frac{cw/(c + \epsilon)}{w} = \frac{e}{c + \epsilon}
\]
and in $L^\infty$:
\[
\int_{w/(c+\epsilon)}^{w/c} f_p(q) \left(1 - \frac{qc}{w}\right) dq \leq \|f_p\|_\infty \int_{w/(c+\epsilon)}^{w/c} \left(1 - \frac{qc}{w}\right) dq
\leq \frac{e^2 w \|f_p\|_\infty}{2(c + \epsilon)^2}
\]

The collision probability $\Pr\{h_{ab}(u) = h_{ab}(v)\}$ is bounded from both ends as follows:
\[
P = \min \left(\frac{2e}{c + \epsilon}, \frac{e^2 w \|f_p\|_\infty}{2(c + \epsilon)^2}\right) + \min \left(\frac{e}{c - \epsilon}, \frac{w^2 \|f_p\|_\infty}{2(c - \epsilon)^2}\right)
\]

Note that, when calculating the lower and upper bounds, $q$ represents $\frac{t}{c + \epsilon}$ and $\frac{t}{c - \epsilon}$, respectively. The algorithm of LSER is shown in Algorithm 1.

In the following, we demonstrate from two perspectives that LSER can find the efficiency of a similar state. First, we show the efficacy of LSER with a theoretical guarantee, i.e., similar states can be sampled given the current state. We formulate "the sampling of similar states" as a neighbor-finding problem in the projective space and provide theoretical proof of the soundness of LSER. Given a set of states $S$, and a query $q$, LSER can quickly find a state $s \in S$ within distance $r_2$ or determine that $S$ has no states within distance $r_2$. Based on existing work [16], the LSH family is $(r_1,r_2,p_1,p_2)$-sensitive, i.e., we can find a distribution $H$ such that $p_1 \geq Pr_{h \sim H}\{h_{ab}(u) = h_{ab}(v)\}$ whenever $u$ and $v$ are similar and $p_2 \leq Pr_{h \sim H}\{h_{ab}(u) = h_{ab}(v)\}$ whenever $u$ and $v$ are dissimilar.

Theorem 2. Let $H$ be $(r_1,r_2,p_1,p_2)$-sensitive. Suppose $p_1 > 1/n$ and $p_2 > 1/n$, where $n$ is the size of data points. There exists a solution for the neighbor finding problem in LSER within $O(n^\rho p_1^{-1} \log n)$ query time, and $O(n^{1+p_2^{-1}})$ space.

Proof. Assume $r_1, r_2, p_1, p_2$ are known, $\rho = \frac{\log(1/p_1)}{\log(1/p_2)}$, and $k = \frac{\log(n)}{\log(1/p_3)}$ where $k$ is the number of hash functions, and LSH initializes $L$ tables. Based on the definition in [16], we have:
\[
kL = k \left[p_1^{-k}\right] \leq k(e^{\log(1/p_1) - k} + 1) \leq k(n^\rho/p_1 + 1) = O(n^\rho/p_1 \log n)
\]

The space complexity is straightforward and can be calculated as $O(Ln_{ds})$ where $d_s$ is the dimension of state $s$. It can be written as $O(n^{1+p_2^{-1}}/p_1)$ and further simplified into $O(n^{1+p_2^{-1}}/p_1)$.

Then, we prove LSER can find similar neighbors. The $L$ table can be classified into two categories: similar and dissimilar. Given a state $s$, the similar category gives similar states while the dissimilar category provides dissimilar states. We split the two categories such that $L = [n] + [m]$ and its corresponding $[k], [k]$. Given any state $s \in S$ in the distance $r_1$, LSER must be able to find the most similar states with higher probability—the query and the data need to share the same hash-bucket in one of the tables. The probability of their not sharing the same hash-bucket is:
\[
(1 - p_1^k) \{1 - p_1^k\}^m \leq (1 - p_1^k)^{n-1} (1 - p_1^k)^m
\]

\[
\leq e^{-\alpha n 1 - p_1^k} (1 - p_1^k)^{-1} = e^{-\alpha n 1 - p_1^k} (1 - p_1^k)^{-1}
\]

\[
= e^{-1} (1 - p_1^k)^{-1}
\]

where $\alpha = [k] - k$. We have applied the definitions $p_1^k = p_2^k = n^{-\rho}$ for step 6 to step 7 and $np_1^{1+\alpha} + mp_2^\alpha = n^\rho$ for step (7) to (8). Finally, we get the probability of LSER’s getting the similar states as follows:
\[
P \geq 1 - e^{-1} (1 - p_1^k)^{-1} > 0
\]

Recall that $p_1 > 1/n$. Therefore, we conclude that LSER can find the most similar states.

2.3 Storage and Sampling Strategy

Existing experience replay methods in DRL research assume that recent experience is more informative than older experience. Therefore, they simply replace the oldest experience with the newest experience to update the experience buffer in DRL-based recommender systems without further optimization.

As such, some valuable experiences might be discarded, i.e., catastrophic forgetting. In contrast, we design a state-aware reward-driven experience storage strategy, which removes the experience with the lowest reward—instead of following the First-In-First-Out (FIFO) strategy—when the replay buffer is full. Formally speaking, a transition $t_i = (s_{i}, a_i, r_{i+1}, s_{i+1})$ will be stored in the replay buffer based on the value $h_{\psi}(s_{i})$. If the replay buffer is full, the transition with the same value of $h_{\psi}(s_{i})$ but lower reward will be replaced. In practice, an indicator $m_i$ is stored in the transition as well to indicate when the recommendation should terminate.
The sampling strategy is another crucial component of LSER, determining which experience should be selected for the agent to optimize in LSER. We propose a state-aware reward-driven sampling strategy that only replays the experience with the top-N highest rewards in the same hashing area; this way, the agent can quickly find the correct direction for optimization. We call our sampling strategy ‘state-aware’ because we use a hash key to encode the state and replay the experience based on the hash key. Our strategy has a higher chance of replaying the correct experience than uniform sampling. Here, we illustrate how to address three related challenges faced by our sampling strategy: exploitation-vs-exploration dilemma, bias annealing and non-existence dilemma.

**Exploitation vs. exploration dilemma.** The exploitation and exploration dilemma is a well-known dilemma when training an agent for RL, including LSER. While our reward-driven strategy forces the agent to exploit existing high-rewarding experiences, the agent may converge to a sub-optimal policy instead of the globally optimal one. We use a similar method to $\epsilon$-greedy to achieve a trade-off between exploitation and exploration. LSER first draws a random probability $\rho \in [0,1]$ then uses reward-driven sampling if the probability is less than a threshold $\epsilon_{\text{max}}$, and random sampling otherwise. The threshold allows LSER to replay low-priority experiences to fulfill the exploration requirement.

**Bias annealing.** Prioritizing partial experiences among the replay buffer may introduce inductive bias [29, 33]—the training process is highly non-stationary (due to changing policies), even a small bias introduced by the sampling strategy may change the solution that the policy converges to. A common solution is to let the priority anneal periodically so that the agent can visit those less-replayed experiences. By using the threshold, our $\epsilon$-greedy method has a similar effect as annealing on allowing low-priority experiences to be replayed.

**Non-existence dilemma.** When splitting the projective space into areas to initialize hyperplanes, some areas may not have any data points (esp. when the number of hyperplanes is large), causing the ‘non-existence dilemma.’ Consequently, when a new transition comes, the algorithm will stop if no experience can be found on $h_{p \in \mathcal{P}}$. We use the similarity measure to overcome this problem. Specifically, we find the two hash areas that are most similar to each other (based on current $h_{p \in \mathcal{P}}$) and conduct sampling on those two states. We use Jaccard similarity to measure the similarity between hash codes $A, B$. As such, LSER can always replay the relevant experience.

### 2.4 Training Procedure

We use Deep Deterministic Policy Gradient (DDPG) [20] as the training backbone. DDPG has been dominant in enabling dynamic recommender systems recently [6]. It is worth mentioning that compared to SAC or TD3, DDPG aims to solve the Q-value overs-estimation problem in a more sample-efficient way, which makes it more suitable for recommender systems where interaction data are usually highly sparse.

In our experiments, we choose an actor-critic network as the agent and train two parts of the actor-critic network simultaneously. We can simply incorporate LSER to any actor-critic based DRL RS if LSER works properly with DDPG. The critic network aims to minimize the following loss function:

$$I(\theta_\psi) = \frac{1}{N} \sum_{j=1}^{N} ((r + \gamma \xi) - \psi_\theta_\psi(s_t, a_t))^2$$

where $\xi = \psi_\theta_\psi(s_{t+1}, \phi_\theta_\psi(s_{t+1}))$

where $\theta_\psi$ and $\theta_\phi$ are the critic andactor parameters, $N$ is the size of the mini-batch from the replay buffer, $\psi_\theta_\psi$ and $\phi_\theta_\psi$ are the target critic and target actor network, respectively. We apply the Ornstein-Uhlenbeck process in the action space to introduce perturbation; this encourages the agent to explore. The target network will be updated based on the corresponding hyper-parameter $\tau$. 

![Figure 2: Given a high dimensional space, three random hyper-planes are initialized based on normal distribution. Each hyper-plane splits the space into two hash areas 0 and 1. The space is split into six hash areas. We can find that, states are encoded into a binary string e.g.,[111, 101, 011, 001, 000].](image)

![Figure 3: The structure of the DDPG which will be used as the main algorithm for our DRL agent in this work.](image)
3 EXPERIMENTS

Algorithm 1: LSH memory by using dictionary

```python
input: Transition for storage \( r : (s_t, a_t, s_{t+1}, r_t) \)
    capacity \( c \), state dimension \( d_s \), batch size \( b \), Hash bits \( n_h \), state \( s \) for sampling, epsilon threshold \( \epsilon_{max} \).

1. Initialize \( n_h \) hyperplanes on projection space \( \mathcal{P}_{d_s} \).
2. Initialize empty dictionary \( \mathcal{M} \).

3. Function \( encode(s) \) is
   for \( p \) in \( \mathcal{P}_{d_s} \), do
      Calculate hash bits by using Eq.2;
   end

4. return **"\text{join}(hash bits)\"**

5. Function \( \text{Push}(r) \) is
   if \( T.size < T.capacity \) then
      if \( v_h \) in \( \mathcal{M} \) then
         \( \mathcal{M}[v_h].append(r); \)
      else
         \( \mathcal{M}[v_h] = r \);
      end
   else
      if \( v_h \) in \( \mathcal{M} \) and \( r.r_t > \mathcal{M}[v_h][0].r \) then
         \( \mathcal{M}[v_h][0] = r; \)
      end
   end

7. Sort(\( \mathcal{M}[v_h] \)) based on reward in ascending order;

8. end

9. Function \( \text{Sample}(s,b) \) is
   \( v_h = encode(s); \)
   \( p = \text{random}.random(); \)
   Find two most similar hash values \( v_1, v_2 \) based on \( v_h; \)
   if \( v_h \) in \( \mathcal{M} \) then
      if \( p < \epsilon_{max} \) then
         \( \text{result} = \mathcal{M}[v_1] \backslash [−b :]; \)
      else
         \( \text{result} = \text{random}.sample(\mathcal{M}[v_h], b); \)
      end
   else
      if \( p < \epsilon_{max} \) then
         \( \text{result} = \mathcal{M}[v_1] \backslash [−b :] + \mathcal{M}[v_2] \backslash [−b :]; \)
      else
         \( \text{result} = \text{random}.sample(\mathcal{M}[v_1], b) + \text{random}.sample(\mathcal{M}[v_2], b); \)
      end
   end

10. return result;

end
```

3.1 Online Simulation Platform Evaluation

We conduct experiments on three widely used public simulation platforms: VirtualTB [31], RecSim [15] and RecoGym [28], which mimic online recommendations in real-world applications.

**VirtualTB** is a real-time simulation platform for recommendations, where the agent recommends items based on users’ dynamic interests. VirtualTB uses pre-trained generative adversarial imitation learning (GAIL) to generate different users with both static and dynamic interests. Moreover, the interactions between users and items are generated by GAIL. Benefitting from that, VirtualTB can provide a large number of users and the corresponding interactions to simulate real-world scenarios.

**RecSim** is a configurable platform for authoring simulation environments that naturally supports sequential interaction with users in recommender systems. RecSim differs from VirtualTB in containing different, simpler tasks but fewer users and items. There are two different tasks from RecSim, namely interest evolution and long-term satisfaction. The former (interest evolution) encourages the agent to explore and fulfill the user’s interest without further exploitation; the latter (long-term satisfaction) depicts an environment where a user interacts with a content characterized by the level of ‘clickbaitiness.’ Generally, clickbaity items lead to more engagement yet lower long-term satisfaction, while non-clickbaity items have the opposite effect. The challenge lies in balancing the two to achieve a long-term optimal trade-off under the partially observable dynamics of the system, where satisfaction is a latent variable that can only be inferred from the increase/decrease in engagement. We follow the same way as mentioned in [14] to evaluate models.

**RecoGym** is a small Open AI gym-based platform where users have no long-term goals. Different from RecSim and VirtualTB, RecoGym is designed for computational advertising. Similar to RecSim, RecoGym uses clicks (or not) to represent the reward signal. Moreover, similar to RecSim, users in those two environments do not contain any dynamic interests.

Considering RecoGym and RecSim have limited data points and do not consider users’ dynamic interests, we select VirtualTB as the main platform for evaluations. Our model is implemented in Pytorch [27], and all experiments are conducted on a server with two Intel Xeon CPU E5-2697 v2 CPUs with 6 NVIDIA TITAN X Pascal GPUs, 2 NVIDIA TITAN RTX, and 768 GB memory. We use two two-hidden-layer neural networks with 128 hidden units as the actor network and the critic network. \( r, y, \) and \( c \) are set to 0.001, 0.99 and \( 1e^4 \), respectively, during experiments.

3.2 Evaluation Metrics and Baselines

The evaluation metrics are environment-specific. For VirtualTB and RecoGym, click-through rate is used as the main evaluation metric. For RecSim, we use the built-in metric, which is a quality score, as the main evaluation metric.

The overall evaluations are two-fold. We first compare our proposed LSER with the mainstream ER variants under DDPG frameworks. Then we show the efficacy of LSER by replacing the original ER in existing DRL-based recommender system methods: TPGR [3], KGRL [5] and PGPR [35] with LSER. Firstly, we compare our method
Figure 4: Result comparison with four baseline methods on VirtualTB, RecSim and RecoGym. The experiments are repeated five times, and mean values are reported. 95% confidence intervals are shown. (a) is the result for VirtualTB; (b) is the result for long-term satisfaction in RecSim; (c) is the result for interest evolution in RecSim; (d) is the result for RecoGym; (e) is the result for \( \varepsilon \) study; (f) is the ablation study to show the effectiveness of each component.

with the following ER baselines and the backbone experiments are shown in Section 3.7

- Prioritized Experience Replay (PER) [29]: an experience replay method for discrete control, which uses TD-error to rank experience and a re-weighting method to conduct the bias annealing.
- Dynamic Experience Replay (DER) [22]: an experience replay method designed for imitation learning, which stores both human demonstrations and previous experience. Those experiences are selected randomly without any priority.
- Attentive Experience Replay (AER) [33]: an experience replay method that uses attention to calculate the similarity for boosting sample efficiency with PER.
- Selective Experience Replay (SER) [17]: an experience replay method for lifelong machine learning, which employs LSTM as the experience buffer and selectively stores experience.
- Hindsight Experience Replay (HER) [1]: an experience replay method that replays two experiences (one successful, one unsuccessful) each time.

For AER, PER, SER, and HER, we use the same training strategy as LSER. For DER, we use its original structure to run experiments without human demonstrations. The size of the replay buffer is set to 1,000,000 for VirtualTB and 10,000 for RecSim and RecoGym. The number of episodes for our experiments is set to 90,000 for VirtualTB and 1,000 for RecSim and RecoGym. Only PER, AER, and SER contain a prioritized operation to rank or store experience.

3.3 Results and Evaluation

Results for the three platforms (Fig 4) demonstrate our method (LSER) outperforms the baselines: LSER yields significant improvements on VirtualTB, which is a large and sparse environment; while AER, DER, PER, and SER find a correct policy within around 50,000 episodes, ours takes around 30,000 episodes; HER does not perform well because it introduces too much failed experience and has a slow learning process; DER introduces the human demonstration into the vanilla ER, which is hard to acquire for recommendation tasks.

Applying PER to DDPG slightly outperforms applying DER to DDPG, which is consistent with observations by previous work [25, 33]. As PER was originally designed for Deep Q-learning, it uses the high TD-error to indicate highly informative experience for the value network. When applying PER into DDPG, an actor-critic-based algorithm, the sampled experience is also used to update the policy network. Those experiences with high TD-error normally diverge far from the current policy and harm the updates of the policy-network. In contrast, LSER selects experience according to the similarity with the current state. This preference for on-distribution states tends to discard experiences that contain old states and stabilize the training process of the policy network. AER
The attention mechanism allows agents to focus on the most relevant information for their decision-making. This mechanism is often used in recommendation systems to match users with items that are most similar to their preferences.

The attention mechanism is particularly useful in scenarios where the number of possible actions is large, such as in recommendation systems. By focusing on the most relevant aspects of the user's preferences, the attention mechanism can help agents make more accurate decisions.

The attention mechanism is also useful in scenarios where the state space is large and complex. By focusing on the most relevant aspects of the state, the attention mechanism can help agents make more accurate decisions. This is particularly useful in recommendation systems, where the state space can be large and complex due to the high number of possible actions.

The attention mechanism is also useful in scenarios where the state space changes over time. By focusing on the most relevant aspects of the state, the attention mechanism can help agents make more accurate decisions. This is particularly useful in recommendation systems, where the state space can change over time as users' preferences change.

The attention mechanism is also useful in scenarios where the state space is high-dimensional. By focusing on the most relevant aspects of the state, the attention mechanism can help agents make more accurate decisions. This is particularly useful in recommendation systems, where the state space can be high-dimensional due to the large number of possible actions.

The attention mechanism is also useful in scenarios where the state space is uncertain. By focusing on the most relevant aspects of the state, the attention mechanism can help agents make more accurate decisions. This is particularly useful in recommendation systems, where the state space can be uncertain due to the variability of user preferences.

The attention mechanism is also useful in scenarios where the state space is non-stationary. By focusing on the most relevant aspects of the state, the attention mechanism can help agents make more accurate decisions. This is particularly useful in recommendation systems, where the state space can be non-stationary due to the changing preferences of users.

The attention mechanism is also useful in scenarios where the state space is complex. By focusing on the most relevant aspects of the state, the attention mechanism can help agents make more accurate decisions. This is particularly useful in recommendation systems, where the state space can be complex due to the high number of possible actions.

The attention mechanism is also useful in scenarios where the state space is dynamic. By focusing on the most relevant aspects of the state, the attention mechanism can help agents make more accurate decisions. This is particularly useful in recommendation systems, where the state space can be dynamic due to the changing preferences of users.

The attention mechanism is also useful in scenarios where the state space is high-dimensional. By focusing on the most relevant aspects of the state, the attention mechanism can help agents make more accurate decisions. This is particularly useful in recommendation systems, where the state space can be high-dimensional due to the large number of possible actions.

The attention mechanism is also useful in scenarios where the state space is uncertain. By focusing on the most relevant aspects of the state, the attention mechanism can help agents make more accurate decisions. This is particularly useful in recommendation systems, where the state space can be uncertain due to the variability of user preferences.

The attention mechanism is also useful in scenarios where the state space is non-stationary. By focusing on the most relevant aspects of the state, the attention mechanism can help agents make more accurate decisions. This is particularly useful in recommendation systems, where the state space can be non-stationary due to the changing preferences of users.
Vanilla ER is used to help the agent learn from past experience. The vanilla ER, which uniformly samples experiences from the replay buffer. Among them, Zhao et al. [41] apply DQN to online recommendation and RNN to generate state embeddings; Chen et al. [4] point out that DQN receives unstable rewards in dynamic environments such as online recommendation and may harm the agent; Chen et al. [3] found those traditional methods like DQN become intractable when the state becomes higher-dimensional; DPG addresses the intractability by mapping high-dimensional discrete state into low-dimensional continuous state [5, 38].

Intuitively, some instances are more important than others. So a better experience replay strategy is to sample experiences according to how much the current agent can learn from each of them. While such a measure is not directly accessible, proxies propose to retain experiences in the replay buffer or to sample experiences from the buffer.

In simple continuous control tasks, experience replay should contain experiences that are not close to the current policy to prevent fitting to local minima, and the best replay distribution is in between an on-policy distribution and uniform distribution [7]. However, De Bruin et al. [7] also note that such a heuristic is unsuitable for complex tasks where policies are updated for many iterations. In DRL problems, when the rewards are sparse, the agent can learn from failed experiences by replacing the original goals with states in reproduced successful artificial trajectories [1].

For complex control tasks, PER [29] measures the importance of experiences using the TD-error and designs a customized importance sampling strategy to avoid the effect of bias. Based on that, Ref-ER [25] actively enforces the similarity between policy and the experience in the replay buffer, considering on-policy transitions are more useful for training the current policy. AER [33] is an experience replay method that combines the advantages of PER and Ref-ER. It uses an attention score to indicate state similarity and replays those experiences awarded high similarity with high priority. All the work above focuses on optimizing the sampling strategy, aiming to select the salient and relevant agent’s experiences in the replay buffer. Selective experience replay (SER) [17], in contrast, aims to optimize the storing process to store only valuable experiences. The main idea is to use a Long-short term memory (LSTM) network to store only useful experiences.

5 CONCLUSION

This paper proposes state-aware reward-driven experience replay (LSER) to address the sub-optimality and training instability issues with reinforcement learning for online recommender systems. Instead of focusing on improving the sample efficiency for discrete tasks, LSER considers online recommendation as a continuous task; it then uses locality-sensitive hashing to determine state similarity and reward for efficient experience replay. Our evaluation of LSER against several state-of-the-art experience-replay methods on three benchmarks (VirtualTB, RecSim, and RecoGym) demonstrate LSER’s feasibility and superior performance.

In the future, we will explore new solutions for improving stability, such as better optimizers to help the agent get rid of saddle points, new algorithms to stabilize the training for DDPG, and trust region policy optimization to increase training stability [30]. Moreover, more advanced reinforcement learning algorithms could be used to replace the DDPG, such as soft actor-critic (SAC) [10] or Twin Delayed Deep Deterministic (TD3) [9].

REFERENCES


