Localilty-Sensitive Experience Replay for Online Recommendation

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

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

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

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

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, due to large state space (e.g., user-item rating matrix and user profiles), action space (e.g., candidate items), and sparse rewards. Existing studies leverage experience replay (ER) to let an agent learn from past experience. However, they adapt poorly to the complex environment of online recommender systems and are inefficient in determining an optimal strategy from past experience. To address these issues, we design a novel state-aware experience replay model, which selectively selects the most relevant, salient experiences, and recommends the agent with the optimal policy for online recommendation. In particular, the model uses locality-sensitive hashing to map high dimensional data into low-dimensional representations and a prioritized reward-driven strategy to replay more valuable experience at a higher chance. Experiments on three online simulation platforms demonstrate our model’s feasibility and superiority to several existing experience replay methods.

KEYWORDS

Recommender System, Deep Reinforcement Learning, Experience Replay

1 INTRODUCTION

Online recommendation aims to learn users’ preferences and recommend items dynamically to help users find desired items in highly dynamic environments [35]. Deep reinforcement learning (DRL) naturally fits online recommendation as it learns policies through interactions with the environment via maximizing a cumulative reward. Besides, DRL has been widely applied to sequential decision-making (e.g. in Atari [22] and AlphaGo [31]) and achieved remarkable progress. Therefore, it is increasing applied for enhancing online recommender systems [2, 4, 37].

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. As the representatives, Zheng et al. [40] 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 recommendation. 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 [8]—the “maximize” operation becomes unfeasible when the action space has high dimensionality (e.g., 100,000 items form a 10k-dimensional action space) [3]. Policy-gradient-based methods use the average reward as guideline to mitigate the stuck agent problem [3]. However, they are prone to converge to sub-optimality [25]. While both DQN and policy gradient are more suitable for small action and state spaces [19, 33] in a recommendation context, hybrid methods [3, 8, 13, 38] has the capability to map large high-dimensional discrete state spaces into low-dimensional continuous spaces via combines the advantages of Q-learning and policy gradient. A typical hybrid method is the actor-critic network [18], 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 systems [5, 20].

Existing DRL-based recommendation methods except policy-gradient-based ones rely heavily on experience replay to learn from previous experience, avoid re-traversal of the state-action space, and stabilize the training on large, sparse state and action spaces [34]. They generally require long training time, thus suffering from the training inefficiency problem. Further more, in contrast to the larger, diverse pool of continuous actions required in recommendation tasks, existing experience replay methods are mostly designed for games with a small pool of discrete actions. Therefore, a straightforward application of those methods may result in strong biases during the policy learning process [12], thus impeding the generalization of optimal recommendation results. For example, Schaul et al. [28] assume that not every experience is worth replaying and propose a prioritized experience replay (PER) method to replay only the experience with the largest temporal difference error. Sun et al. [32] propose attentive experience replay (AER), which introduces similarity measurement into PER to boost the efficiency of finding similar states’ experience, but attention mechanisms cause inefficiency on large sized state and action spaces [17].

Existing DRL-based recommendation methods except policy-gradient-based ones rely heavily on experience replay to learn from previous experience, avoid re-traversal of the state-action space, and stabilize the training on large, sparse state and action spaces [34]. They generally require long training time, thus suffering from the training inefficiency problem. Further more, in contrast to the larger, diverse pool of continuous actions required in recommendation tasks, existing experience replay methods are mostly designed for games with a small pool of discrete actions. Therefore, a straightforward application of those methods may result in strong biases during the policy learning process [12], thus impeding the generalization of optimal recommendation results. For example, Schaul et al. [28] assume that not every experience is worth replaying and propose a prioritized experience replay (PER) method to replay only the experience with the largest temporal difference error. Sun et al. [32] propose attentive experience replay (AER), which introduces similarity measurement into PER to boost the efficiency of finding similar states’ experience, but attention mechanisms cause inefficiency on large sized state and action spaces [17].
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 [32], LSER only replays experience from similar states to improve the sampling efficiency. Specifically, we introduce a ranking mechanism to prioritize replays and promote the higher reward experiences. We further use ε-greedy method to avoid replaying high-rewards states excessively.

Considering the high-dimensionality of vectorized representations of states, We convert similarity measurement 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 in this paper:

- 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 recommend 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 \( \mathcal{U} = \{u_0, u_1, \ldots, u_n\} \), a set of items \( \mathcal{I} = \{i_0, i_1, \ldots, i_m\} \) and user’s demographic information \( \mathcal{D} = \{d_0, d_1, \ldots, d_n\} \), 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 \( \mathcal{I} \) and its corresponding user information) \( A \) denotes the action space, which represents 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 recommended item or not), and \( \gamma \) is a discount factor \( \gamma \in [0, 1] \) for the trade-off between future and current rewards.

Given a user \( u \) and an initial state \( s_0 \) observed by the agent (or the recommender system), which includes a subset of item set \( I \) and user’s profile information \( d_0 \), a typical recommendation iteration for the user goes as follows: first, the agent takes an action \( a_0 \) based on the recommend policy \( \pi_0 \) under the observed state \( s_0 \) and receives the corresponding reward \( r_0 \)—the reward \( r_0 \) is the numerical representation for user’s behavior such as click through or not; then, the agent generates a new policy \( \pi_1 \) based on the received reward \( r_0 \) and determines the new state \( s_1 \) based on the probability distribution \( p(s_{new}|s_0, a_0) \in \mathcal{P} \). The cumulative reward (denoted by \( r_c \)) after \( k \) iterations from the initial state is as follows:

\[
r_c = \sum_{k=0}^{\infty} \gamma^k r_k
\]

DRL-based recommender systems uses a replay buffer to store and replay old experience for training. Given the large state and action space in a recommender system, not every experiences are worth to replay [6]—replaying experience that does not contain useful information will increase the training time significantly and introduce extra uncertainty to convergence. Hence, it is reasonable to prioritize replay important experience for DRL recommender systems.

The ideal criterion for measuring the importance of a transition in RL is the amount of knowledge learnable from the transition in its current state [11, 28]. State-of-the-art methods like AER are unsuitable for recommendation tasks that contain large, higher dimensional state and action spaces as their sampling strategies may not work properly. Thus, we propose a new experience replay method named Locality-sensitive experience replay (LSER) for online recommendation, which uses hashing for dimension reduction when sampling and storing the experiences.

2.2 Locality-sensitive Experience Replay

We formulate the storage and sampling issue in LSER as a similarity measure problem, where LSER stores similar states into the same buckets and samples similar experiences based on state similarities. A popular way of searching similar high-dimensional vectors in Euclidean space is Locality-Sensitive Hashing (LSH), which follows the idea of Approximate Nearest Neighbor (ANN) while allocating similar items into the same buckets to measure the similarity. However, standard LSH conducts bit-sampling on the Hamming space; it requires time-consuming transformation between the Euclidean space to the Hamming space, liable to lose information. Aiming at measuring the similarity between high-dimensional vectors without losing significant information, we propose using \( p \)-stable distribution [23] to conduct dimensionality reduction while preserving the original distance. This converts high-dimensional vectors (states) into low-dimensional representations easier to be handled by the similarity measure.

To address possible hash collision (i.e., dissimilar features may be assigned into the same bucket and recognized as similar), we introduce the formal definition of the collision probability for LSH. Then, we theoretically analyze the collision probability for \( p \)-stable distribution to prove that our method has a reasonable boundary for collision probability.
**Figure 1:** The proposed LSER with DDPG. The environment provides the current state $s_t$, with the policy $\pi$ learned by the DDPG model; the action $a_t$ can be obtained by $a_t = \pi(s_t)$. $r_t$ will be provided by the user (e.g. click or not). LSER takes $s_t$ as the input and encodes it on the projective space. Given the encoded states, LSER will return the most similar experience for DDPG to update he parameters. After that, this transition $h(s_t) : (s_t, a_t, s_{t+1}, r_t)$ will be stored.

**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:

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

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 dimension reduction. We randomly initialize $n_h$ hyperplanes based on normal distribution on the projective space $\mathcal{P}^n$ 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:

$$h_{p \in \mathcal{P}^n}(s) = \{0, 1\}^n \text{with } \begin{cases} 1 & p_i \cdot s_i > 0 \\ 0 & p_i \cdot s_i \leq 0 \end{cases} \quad (2)$$

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

$$P = Pr[h_{p \in \mathcal{P}^n}(u) = h_{p \in \mathcal{P}^n}(v)] = 1 - \frac{\text{Ang}(u, v)}{\pi}$$

where $\text{Ang}(u, v) = \arccos \frac{u \cdot v}{\|u\| \cdot \|v\|}$

Eq.(2) formulates the information loss during the projection, where we use term $\epsilon$ to represent the quantification between the real value $p \cdot v$ and hashed results induced from $h(v)$. Since the relative positions in original space are preserved during the hash transformation with an extra measurement $\epsilon$, the upper bound and lower bound of collision probability boundary in projective space is guaranteed to be intact. That means the more dissimilar states will not receive a higher probability to be allocated into 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 $\epsilon$ 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 $c - \epsilon$ and from below by $Pr[h_{ab}(u) = h_{ab}(v)]$ for $c + \epsilon$.

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

Then, we have the upper bound:

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

$$\int_0^{w/(c-\epsilon)} \left(f_p(q) \left(1 - \frac{qc}{w}\right) + q f_p(q) \left(1 - \frac{qc}{w}\right) dq \leq P$$

\[\boxed{P + \frac{\epsilon}{w} \int_0^{w/(c-\epsilon)} q f_p(q) dq} \leq P + \frac{\epsilon}{c-\epsilon}\]
and the lower bound:

\[
\int_{0}^{w/(c+e)} f_p(q) \left( 1 - \frac{q}{w} \right) dq \leq \left( \sup_{q \in \{0, w/(c+e)\}} q \right) \left\| f_p \right\|_1 \leq \frac{w}{c+e}.
\]

Considering the \( L^\infty \) space, we have:

\[
\int_{0}^{w/(c+e)} f_p(q) dq \leq \left\| f_p \right\|_\infty \int_{0}^{w/(c+e)} dq = \frac{w^2 \left\| f_p \right\|_\infty}{2(c+e)^2}.
\]

We use the similar method in \( L^1 \) to compute the lower bound:

\[
\int_{w/(c+e)}^{w/c} f_p(q) \left( 1 - \frac{q}{w} \right) dq \leq \left( \sup_{q \in \{w/(c+e), w/c\}} q \right) \left\| f_p \right\|_1 \leq 1 - \frac{cw/(c+e)}{w} = \frac{e}{c+e},
\]

and in \( L^\infty \):

\[
\int_{w/(c+e)}^{w/c} f_p(q) \left( 1 - \frac{q}{w} \right) dq \leq \left\| f_p \right\|_\infty \int_{w/(c+e)}^{w/c} \left( 1 - \frac{q}{w} \right) dq \leq \frac{e^2w \left\| f_p \right\|_\infty}{2(c+e)^2}.
\]

The collision probability \( Pr[h_{ab}(u) = h_{ab}(v)] \) is bounded from both ends as follows:

\[
\left( 2e \cdot \frac{e^2w \left\| f_p \right\|_\infty}{2(c+e)^2} \right) \left( 2e \cdot \frac{w^2 \left\| f_p \right\|_\infty}{2(c+e)^2} \right).
\]

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

In the following, we demonstrate from two perspectives that LSER can find the similar states efficiency. First, we show the efficiency of LSER with 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 a 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 \( r_1 \). Based on existing work [15], the LSH family is \((r_1, r_2, p_1, p_2)\)-sensitive, i.e., we can find a distribution \( H \) such that \( p_1 \geq Pr_{H \sim H}[h_{ab}(u) = h_{ab}(v)] \) when \( u \) and \( v \) are similar and \( p_2 \leq Pr_{H \sim H}[h_{ab}(u) = h_{ab}(v)] \) when \( u \) 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 \( O(n^{1+\rho} p_1^{-1}) \) space.

**Proof.** Assume \( r_1, r_2, p_1, p_2 \) are known, \( \rho = \log(1/p_1)/\log(1/p_2) \), and \( k = \log(n)/\log(1/p_2) \) where \( k \) is the number of hash functions, and LSH initializes \( L \) tables. Based on the definition in [15], 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 calculated as \( O(Lnd_s) \) where \( d_s \) is the dimension of state \( s \). It can be written as \( O(n^{1+\rho}/p_1 d_s) \) (by applying \( L = n^\rho/p_1 \) and further simplified into \( O(n^{1+\rho}/p_1) \)).

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 \( L = [s] + [m] \) 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 in a high 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

\[
(1 - p_1^{[k]} \cdot |n|)(1 - p_1^{[k]} \cdot |m|) \leq (1 - p_1^{[k]} |n|)(1 - p_1^{[k]} |m|) (5)
\]

\[
\leq e^{-np_1^{[k]} |m|} (1 - p_1^{[k]} - 1) \leq e^{-np_1^{[k]} - np_1^{[k]} |m|} (1 - p_1^{[k]} - 1) \leq e^{-1 - (1 - p_1^{[k]} - 1)} \leq 0
\]

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

\[ \square \]

### 2.3 Store and Sampling Strategy

Existing experience replay methods in DRL research assume that the 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 experience 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 \( t \) : \((s_t, a_t, s_{t+1}, r_t)\) will be stored in the replay buffer based on the value \( h_{p\rho} = h_{p\rho}(t, s_t) \). If the replay buffer is full, the transition with the same value of \( h_{p\rho} = h_{p\rho}(t, s_t) \) but lower reward will be replaced. In practice, an indicator \( m_t \) is stored in the transition as well to indicate when the recommendation should terminate.
We use a similar method to with uniform sampling, our strategy has a higher chance to replay the space is split into six hash areas. We can find that, states \( p \in [0, 1] \), then uses reward-driven sampling if the probability less than a threshold, \( \epsilon_{\text{max}} \), and random sampling otherwise. The threshold allows LSER to replay low priority experience to fulfill the exploration requirement.

**Non-existence dilemma.** When split 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 \( h_{p \in P} \). 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 \( h_{p \in P} \)) 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) [19] as the training backbone. We choose an actor-critic network as the agent and train two parts of the actor-critic network simultaneously. The critic network aims to minimize the following loss function:

\[
    l(\theta) = \frac{1}{N} \sum_{j=1}^{N} (r + \gamma \xi - \psi_{\phi}(s_t, a_t))^2
\]

where \( \xi = \psi_{\phi}(s_{t+1}, \phi'_{\phi}(s_{t+1})) \)

where \( \theta \) and \( \phi \) are the critic and actor parameters, \( N \) is the size of the mini-batch from the replay buffer, \( \psi_{\phi} \) and \( \phi'_{\phi} \) 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 \( \tau \).

### 3 EXPERIMENTS

#### 3.1 Online Simulation Platform Evaluation

We conduct experiments on three widely used public simulation platforms: VirtualTB [30], RecSim [14] and RecoGym [27], which mimic online recommendations in real-world applications. VirtualTB is a real-time simulation platform for recommendation, where the agent recommend items based on users’ dynamic interests. VirtualTB uses a pre-trained generative adversarial imitation learning (GAIL) to generate different users who have both static interest and dynamic interest. It’s worth to mention that, the GAIL is pre-trained by using the real-world from Taobao, which is one of the largest online retail platforms in China. Moreover, the interactions between users and items are generated by GAIL as well. Benefit from that, VirtualTB can provide a large number of users and the corresponding interactions to simulate the real-world scenario.

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) encourages the agent to interact with content characterized by the level of ‘clickbaitiness.’ Generally, clickbaity items lead to more engagement yet lower long-term satisfaction, while non-clickbaity items

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 \( p \in [0, 1] \) are encoded into a binary string e.g., [111, 101, 011, 001, 000]
Algorithm 1: LSH memory by using dictionary

\begin{algorithm}
\begin{algorithmic}
\State \textbf{input}: Transition for storage $\tau : (s_t, a_t, r_{t+1}, s_{t+1}, r_t)$, capacity $c$, state dimension $d_s$, batch size $b$, Hash bits $n_h$, state $s$ for sampling, epsilon threshold $\epsilon_{\text{max}}$.
\State Initialize $n_h$ hyperplanes on projection space $\mathcal{P}_{d_s}$;
\State Initialize empty dictionary $M$;
\Function{encode}{$s$}
\For{$p$ in $\mathcal{P}_{d_s}$}
\State Calculate hash bits by using Eq.2;
\EndFor
\State return ".".join(hash bits)
\EndFunction
\Function{Push}{$\tau$}
\State $v_h = \text{encode}(\tau, s_t)$ // get the hash code
\If{$\text{size} < \text{capacity}$}
\If{$v_h$ in $M$}
\State $M[v_h].\text{append}(\tau)$;
\Else
\State $M[v_h] = \tau$;
\EndIf
\Else
\If{$v_h$ in $M$ and $r_t > M[v_h][0].r$}
\State // replace tuple has the minimal reward
\State $M[v_h][0] = r$;
\EndIf
\EndIf
\State Sort($M[v_h]$) based on reward in ascending order;
\EndFunction
\Function{Sample}{$s, b$}
\State $v_h = \text{encode}(s)$;
\State $p = \text{random}.\text{random}();$
\State Find two most similar hash values $v_1, v_2$ based on $v_h$;
\If{$v_h$ in $M$}
\If{$p < \epsilon_{\text{max}}$}
\State result = $M[v_h][-b :]$;
\Else
\State result = random.sample($M[v_h], b$);
\EndIf
\Else
\If{$p < \epsilon_{\text{max}}$}
\State result = $M[v_1][-b :] + M[v_2][-b :]$;
\Else
\State result = random.sample($M[v_1], b$) + random.sample($M[v_2], b$);
\EndIf
\EndIf
\State \Return result;
\EndFunction
\end{algorithmic}
\end{algorithm}

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.}

RecoGym is a small OpenAI gym-based platform, where users have no long-term goals. Different from RecSim and VirtualTB, RecoGym is designed for computational advertising. Similar with RecSim, RecoGym uses the click or not to represent the reward signal. Moreover, similar with 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 [26] 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 unit as the actor network and the critic network, respectively. $\tau, \gamma,$ and $\epsilon$ are set to 0.001, 0.99 and $10^{-6}$, 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. We compare our method with the following baselines.

- Prioritized Experience Replay (PER) [28]: 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) [21]: an experience replay method designed for imitation learning, where stores both human demonstrations and previous experience. Those experiences are selected randomly without any priority.
- Attentive Experience Replay (AER) [32]: an experience replay method that uses attention to calculate the similarity for boosting sample efficiency with PER.
- Selective Experience Replay (SER) [16]: 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 experience (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. Note that only PER, AER and SER contains a prioritize operation to rank or store the experience.

3.3 Results and Evaluation
Results for the three platforms (Fig 3) demonstrate our method (LSER) outperformed 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
Applying PER to DDPG slightly outperforms applying DER to DDPG, which is consistent with observations by previous work [24, 32]. As PER was originally designed for Deep Q-learning, it uses the high TD-error to indicate the high informative experience for the value-network. When applying PER into DDPG, which is 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 away 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 does not perform as well as PER in VirtualTB because it heavily relies on the attention mechanism to calculate the similarity score between states. LSER’s ε-greedy method can enforce agent to do more exploration when user’s interest shift.

All methods gained similar results on RecSim and RecoGym because all methods can iterate all possible combinations of states and actions. Fig. 3b, 3c and 3d show that LSER is slightly better and more stable than the baselines on RecSim and RecoGym. Since the two platforms are quite small\(^1\), similarity matching and ε-greedy do not significantly improve performance.

### 3.4 Running Time Comparison

We report the running time of the selected experience replay methods in Table 1 to evaluate the efficiency of LSER. LSER outperforms all While performing poorly on RecSim and RecoGym, it is faster than most of the baselines. In comparison, LSER introduces extra running time in small environments (e.g., RecSim and RecoGym) than large environments. For VirtualTB, AER takes much longer time than all other methods, due to attention calculation [17].

### 3.5 Ablation Study

We further investigate the effect of LSER’s store and sampling strategy by replacing our store strategy with the normal strategy and our sampling strategy with random sampling. The results of our ablation study are shown in Fig. 3f, where LSER-P denotes LSER with the replaced store strategy and LSER-S denotes the LSER with the replaced sampling strategy. We found the sample strategy played the most critical role in achieving good performance, as LSER-S underperformed LSER significantly. The store strategy also contributed to the better performance. LSER-P was less stable.

---

\(^1\)RecoGym only contains 100 users and 10 products; RecSim contains 100 users and 20 products.
Table 1: Comparison of running time for DER, PER, SER, AER and LSER coupling with DDPG in three different environments when running the experiments in 90,000 episodes

|                  | RecSim(LTS) | RecSim(IE) | RecoGym | VirtualTB |
|------------------|-------------|------------|---------|-----------|
| DER              | 5.63        | 5.42       | 4.53    | 95.22     |
| PER              | 5.44        | 5.15       | 4.18    | 94.52     |
| SER              | 5.31        | 5.05       | 4.21    | 90.05     |
| AER              | 5.18        | 4.94       | 4.12    | 145.33    |
| HER              | 5.33        | 5.11       | 4.20    | 120.33    |
| LSER             | 5.23        | 5.04       | 4.15    | **85.12** |

(indicated by a wider error bar), but outperformed LSER at ∼30,000 episodes, due to occurrence of sub-optimal policies.

3.6 Impact of Number of Hyperplanes

In our method, the number of hyperplanes is critical to determine the length of the result hash-bits of a given state. Longer hash-bits can provide more accurate similarity measurement result but low efficiency, while shorter hash-bits can increase efficiency but decrease the accuracy. It’s a trade-off which needs a middle-point to balance between efficiency and accuracy. We want to answer the following question: “Does increase the hyperplanes always boost the recommendation performance?” and find out the optimal number.

We report the experimental results in VirtualTB, where we evaluate the effect by varying number hyperplanes in LSER (shown in Fig 4). The results on the other two platforms show the similar pattern. The performance gradually increases with more hyperplanes, but it levels off or even drops when number of hyperplanes reaches 20.

3.7 Discussion and Future Extensions

Fig 3a shows LSER suffers instability after reaching the first peak at episode ∼50,000. Different from the other methods, LSER can quickly reach the optimal policy but suffers fluctuation. That indicates ε-greedy tends to lead the agent towards learning from low-priority experience after the optimal policy is reached. We alleviate the issue by adjusting the value of ε. Here, we tried ε = {0, 0.9, 0.99, 1} to determine the best choice of the ε on VirtualTB. The results are shown in Fig 3e, where ε = 1 corresponds to greedy sampling while ε = 0 refers to randomly sampling. Besides, we provide an intervention strategy to stabilize the training process—the agent will stop exploration once the test reward is higher than a reward threshold Tr. This strategy allows the agent to find an near-optimal policy at an early stage. We examined the performance under Tr=0.95, which delivers a better training process.

4 RELATED WORK

Zhao et al. [38] first introduced DRL to recommender systems Zhao et al. [38]. They use DQN to embed user and item information for news recommendation, where Vanilla ER is used to help the agent learn from past experience. And until present, most methods use only vanilla ER, which uniformly samples experiences from the replay buffer. Among them, Zhao et al. [39] 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 that 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, 36].

Intuitively, some instances are more important than others; so a better experience replay strategy is to sampling experiences according to how much 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. Replay strategies reply on optimization objectives. 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, they 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 artificial successful trajectories [1].

For complex control tasks, PER [28] 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 [24] 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 [32] is an experience replay method that combines the advantages from PER and Ref-ER. It uses attention score to indicate state similarity and replays those experiences awarded high similarity with high priority. All aforementioned work focuses on optimizing the sampling strategy, aiming to select the salient and relevant agent’s experiences in replay buffer effectively. Selective experience replay (SER) [16], in contrast, aims to optimize the storing process to ensure only valuable experience will be stored. The main idea is to use
an Long-short term memory (LSTM) network to store only useful experience.

5 CONCLUSION

In this paper, we propose 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 location-sensitivity 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 [29]. Moreover, more advance 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

[1] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. 2017. Hindsight experience replay. In Advances in neural information processing systems. 5048–5058.

[2] Yuexing Bai, Jian Guo, and Hongyang Wang. 2019. A Model-Based Reinforcement Learning with Adversarial Training for Online Recommendation. In Advances in Neural Information Processing Systems. H. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alch´e-Buc, E. Fox, and R. Garnett (Eds.), Vol. 32. Curran Associates, Inc., 10735–10746. https://proceedings.neurips.cc/paper/2019/file/4e6b6523da9ec1347bf146ea35e53d-Paper.pdf.

[3] Haokun Chen, Xinyi Dai, Han Cai, Weinan Zhang, Xuejian Wang, Ruiming Tang, Junchi Zhao, and Yunchao Sun. 2018. A deep reinforcement learning recommendation system. In Proceedings of the 2018 International Joint Conference on Neural Networks (IJCNN). IEEE, 1–8.

[4] Xiaocong Chen, Chaoran Huang, Lina Yao, Xianzhi Wang, Wenjie Zhang, et al. 2019. Recogym: A reinforcement learning environment for the recommendation. In Proceedings of the 2019 ACM SIGIR Conference on Research & Development in Information Retrieval. New York.

[5] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. In Advances in neural information processing systems. 8026–8037.

[6] David Ruhde, Stephen Bonner, Travis Dunlop, Flavian Vasilie, and Alexandros Karatzoglou. 2018. Recognify: A reinforcement learning environment for the problem of product recommendation in online advertising. arXiv preprint arXiv:1808.00720 (2018).

[7] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. 2016. Prioritized experience replay. In International Conference on Learning Representations.

[8] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. 2015. Trust region policy optimization. In International conference on machine learning. PMLR, 1889–1897.

[9] Liujing Hu, Qing Da, Anxiang Zeng, Yang Yu, and Yinghui Xu. 2018. Reinforcement learning to rank in e-commerce search engine: Formalization, analysis, and application. In Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. 368–377.

[10] Eugene Ie, Chih wei Hsu, Martin Madden, Vihan Jain, Samnit Narvekar, Jing Wang, Rui Wu, and Craig Bottou. 2019. RecSim: A Configurable Simulation Platform for Recommender Systems. (2019). arXiv:1908.04347 [cs.LG].

[11] Yu-Hsiang Lai, Kai-Wei Chang, and Anxin Tang. 2019. Reinforcement learning in online advertising with submodularity. In Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 33. 3396–3404.

[12] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2015. Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971 (2015).

[13] Fei-Fei Li, Jiang Zhang, Li-Fang Chen, Bo Han, and Yang Yu. 2015. Video policy gradient for contextualized diverse recommendations. In The World Wide Web Conference. 1421–1431.

[14] Florian Pachet, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. In Advances in neural information processing systems. 8026–8037.

[15] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. In Advances in neural information processing systems. 8026–8037.

[16] David Silver, Aja Huang, Charles Jiang, Matthew Opening, Li, Xiaolong, et al. 2016. Mastering the game of Go with deep neural networks and tree search. nature 529, 7558 (2016). 484–489.

[17] Zheqian Sun, Wengang Zhou, and Houqiang Li. 2020. Attentive experience replay. In Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 34. 5900–5907.

[18] Daoshen Zha, Jue-Her Hang, Lai, Kai, Xiaoshuang Zhou, and Xia Hu. 2019. Experience Replay Optimization. In Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19 International Joint Conferences on Artificial Intelligence Organization, 4243–4249. https://doi.org/10.24963/ijcai.2019/589.

[19] Rui Wang, Linan Ao, Aixin Sun, and Yi Tan. 2019. Deep learning based recommender system: A survey and new perspectives. ACM Computing Surveys (CSUR) 52, 1 (2019), 1–38.

[20] Xingyi Zhao, Xing Wang, Yuren Zhang, Li Zhao, Zheng Liu, Chunxia Kang, and Xing Xie. 2020. Leveraging Demonstrations for Reinforcement Recommendation: Reasoning over Knowledge Graphs. In Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval. 239–248.

[21] Yujing Hu, Qing Da, Anxiang Zeng, Yang Yu, and Yinghui Xu. 2018. Reinforcement learning to rank in e-commerce search engine: Formalization, analysis, and application. In Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. 368–377.

[22] Vincent Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. Mastering the game of Go with deep neural networks and tree search.

[23] Karatzoglou. 2018. Recogym: A reinforcement learning environment for the recommendation. In Proceedings of the 43rd International Conference on Knowledge Discovery & Data Mining. 239–248.
by Xiangyu Zhao, Long Xia, Jiliang Tang, and Dawei Yin with Martin Vesely as coordinator. ACM SIGWEB Newsletter Spring (2019), 1–15.

[38] Xiangyu Zhao, Long Xia, Liang Zhang, Zhuoye Ding, Dawei Yin, and Jiliang Tang. 2018. Deep reinforcement learning for page-wise recommendations. In Proceedings of the 12th ACM Conference on Recommender Systems. 95–103.

[39] Xiangyu Zhao, Xudong Zheng, Xiwang Yang, Xiaobing Liu, and Jiliang Tang. 2020. Jointly learning to recommend and advertise. In Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. 3319–3327.

[40] Guanjie Zheng, Fuzheng Zhang, Zihan Zheng, Yang Xiang, Nicholas Jing Yuan, Xing Xie, and Zhenhui Li. 2018. DRN: A deep reinforcement learning framework for news recommendation. In Proceedings of the 2018 World Wide Web Conference. 167–176.