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

Xiaocong Chen
xiaocong.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

SIGIR '22, July 11–15, 2022, Madrid, Spain.

ACM ISBN 978-1-4503-8732-3/22/07…$15.00
https://doi.org/10.1145/3477495.3532015

ACM Reference Format:
Xiaocong Chen, Lina Yao, Julian McAuley, Weili Guan, Xiaojun Chang, and Xianzhi Wang. 2022. Locality-Sensitive State-Guided Experience Replay Optimization for Sparse Rewards in Online Recommendation. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '22), July 11–15, 2022, Madrid, Spain.* ACM, New York, NY, USA, 10 pages. 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* [8]—the "maximize" operation becomes infeasible 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 a guideline to mitigate the stuck agent problem [3]. However, they are prone to converge to sub-optimality [26]. While both 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 network [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

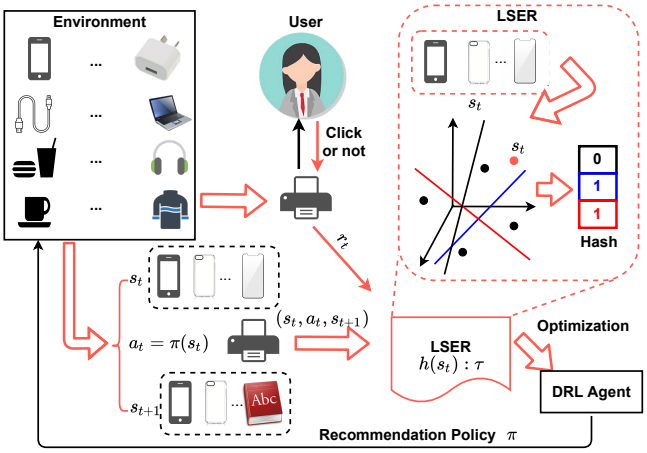


Figure 1: The proposed Locality-Sensitive stete-guided Experience Replay (LSER). The environment provides the current state s_t, with the policy π learned by the agent; the action a_t can be obtained by $a_t = \pi(s_t)$. r_t will be provided by the user (e.g. click or not). LSER takes s_t as the input and encodes it on the projective space. Given the encoded states, LSER will return the most similar experience for the DRL agent to update the parameters. After that, this transition $h(s_t) : (s_t, a_t, s_{t+1}, r_t)$ will be stored.

systems [5, 21]. Hence, we use the actor-critic network as the main framework in this study.

Most studies assuming the reward function is associated with explicit user feedback such as purchase or rating may be hindered by the sparse rewards prevalent in the context of recommender systems. As an example, before a user signals explicitly by making a purchase or rating on the item of interest, a large amount of interaction data has been generated. Such an absence of rewards makes it difficult for the agent to learn an appropriate recommendation policy. Moreover, existing DRL-based recommendation methods, except for policy-gradient-based ones, rely heavily on experience replay to learn from previous experience to avoid re-traversal of the state-action space and stabilize the training on large, sparse state and action spaces [6, 36]. The replay buffer may get stuck with a large number of interactions and thus degrade or even damage the policy learning. Furthermore, 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. [29] 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. [33] propose attentive experience replay (AER), which introduces similarity measurements into PER to boost the efficiency of finding similar states' experiences, but attention mechanisms cause inefficiency on large state and action spaces [18].

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 ϵ-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 $\mathcal{U} = \{u_0, u_1, \cdots u_n\}$, a set of items $\mathcal{I} = \{i_0, i_1, \cdots i_m\}$ and user attributes $\mathcal{D} = \{d_0, d_1, \cdots, d_n\}$, MDP can be represented as a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$, where $\mathcal{S}$ denotes the state space (i.e., the combination of the subsets of $\mathcal{I}$ and its corresponding user information) $\mathcal{A}$ denotes the *action space*, which represents the agent's selection during recommendation based on the *state space* $\mathcal{S}$, $\mathcal{P}$ denotes the set of transition probabilities for state transfer based on the action received, $\mathcal{R}$ 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 γ 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 the item set $\mathcal{I}$ and the user's profile information d_0, a typical recommendation iteration for the user proceeds as follows: first, the agent takes action a_0 based on the recommended policy π_0 under the observed state s_0 and receives the corresponding reward r_0—the reward r_0 is the numerical representation of user's behavior such as click-through or not; then, the agent generates a new policy π_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:

$$r_c = \sum_{k=0} \gamma^k r_k$$

DRL-based recommender systems use a *replay buffer* to store and replay old experiences for training. Given the large state and action space in a recommender system, not every experience is worth replaying [6]—replaying an 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 and replay important experiences 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, 29]. 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 recommendations, 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 measurement problem, where LSER stores similar states in the same buckets and samples similar experiences based on state similarities. A popular way of searching for 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 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 a p-stable distribution [24] 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 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 $\boldsymbol{u}$ and $\boldsymbol{v}$ is represented by:*

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

where $c = \|\boldsymbol{u} - \boldsymbol{v}\|_p$ 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 $\mathcal{P}^n$ 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:

$$P = Pr[h_{p\in\mathcal{P}^n}(\mathbf{u}) = h_{p\in\mathcal{P}^n}(\mathbf{v})] = 1 - \frac{Ang(\mathbf{u}, \mathbf{v})}{\pi}$$
$$\text{where } Ang(\mathbf{u}, \mathbf{v}) = \arccos \frac{|\mathbf{u} \cap \mathbf{v}|}{\sqrt{|\mathbf{u}| \cdot |\mathbf{v}|}}. \quad (3)$$

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 $h(\mathbf{v})$. 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 $\boldsymbol{u}$ and $\boldsymbol{v}$ is bounded at both ends.*

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

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

and the lower bound:

$$\int_0^{w/(c+e)} f_p(q)\left(1-\frac{(c+e)q}{w}\right)dq \text{ with } q=\frac{t}{c+e}$$
$$=\int_0^{w/(c+e)}\left(f_p(q)\left(1-\frac{qc}{w}\right)-\frac{qef_p(q)}{w}\right)dq$$
$$=P-\frac{e}{w}\int_0^{w/(c+e)} qf_p(q)\,dq-\int_{w/(c+e)}^{w/c} f_p(q)\left(1-\frac{qc}{w}\right)dq$$
$$\geq P-\frac{e}{c+e}-\left(1-\frac{c}{c+e}\right)=P-\frac{2e}{c+e}$$

We compute the upper bound based on Hölder's inequality in L^1 space:

$$\int_0^{w/(c-e)} qf_p(q)\,dq\leq\left(\sup_{q\in[0,w/(c-e)]} q\right)\|f_p\|_1\leq\frac{w}{c-e}$$

Considering the L^∞ space, we have:

$$\int_0^{w/(c-e)} qf_p(q)\,dq\leq\|f_p\|_\infty\int_0^{w/(c-e)} q\,dq=\frac{w^2\|f_p\|_\infty}{2(c-e)^2}$$

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

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

and in L^∞:

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

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

$$\left[P-\min\left(\frac{2e}{c+e},\frac{e^2w\|f_p\|_\infty}{2(c+e)^2}\right),P+\min\left(\frac{e}{c-e},\frac{w^2\|f_p\|_\infty}{2(c-e)^2}\right)\right]$$

□

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 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 $\mathcal{S}$, and a query q_s, LSER can quickly find a state $s\in\mathcal{S}$ within distance r_2 or determine that $\mathcal{S}$ has no states within distance r_1. 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 $\mathcal{H}$ such that $p_1\geq Pr_{h\sim\mathcal{H}}[h_{ab}(\mathbf{u})=h_{ab}(\mathbf{v})]$ when $\mathbf{u}$ and $\mathbf{v}$ are similar and $p_2\leq Pr_{h\sim\mathcal{H}}[h_{ab}(\mathbf{u})=h_{ab}(\mathbf{v})]$ when $\mathbf{u}$ and $\mathbf{v}$ are dissimilar.

Theorem 2. *Let $\mathcal{H}$ 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, $\rho=\frac{\log(1/p_1)}{\log(1/p_2)}$, and $k=\frac{\log(n)}{\log(1/p_2)}$ where k is the number of hash functions, and LSH initializes L tables. Based on the definition in [16], we have:

$$kL=k\left\lceil p_1^{-k}\right\rceil\leq k(e^{\log(1/p_1)\cdot k}+1)\leq k(n^\rho/p_1+1)=O(n^\rho/p_1\log n)\tag{4}$$

The space complexity is straightforward and can be calculated as $O(Lnd_s)$ where d_s is the dimension of state s. It can be written as $O(n^{1+\rho}/p_1d_s)$ (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 $L=\lfloor n\rfloor+\lceil m\rceil$ and its corresponding $\lfloor k\rfloor,\lceil k\rceil$. Given any state $s\in\mathcal{S}$ 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

$$(1-p_1^{\lfloor k\rfloor})^{\lfloor n\rfloor}(1-p_1^{\lceil k\rceil})^{\lceil m\rceil}\leq(1-p_1^{\lfloor k\rfloor})^{n-1}(1-p_1^{\lceil k\rceil})^m\tag{5}$$
$$\leq e^{-np_1^{\lfloor k\rfloor}-mp_1^{\lceil k\rceil}}(1-p_1^{\lfloor k\rfloor})^{-1}\tag{6}$$
$$=e^{-(np_1^{-1+\alpha}+mp_1^\alpha)n^{-\rho}}(1-p_1^{\lfloor k\rfloor})^{-1}\tag{7}$$
$$=e^{-1}(1-p_1^{\lfloor k\rfloor})^{-1}\tag{8}$$

where $\alpha=\lceil k\rceil-k$. We have applied the definitions $p_1^k=p_2^{\rho k}=n^{-\rho}$ for step 6 to step 7 and $np_1^{-1+\alpha}+mp_1^\alpha=n^\rho$ for step (7) to (8). Finally, we get the probability of LSER's getting the similar states as follows:

$$P\geq 1-e^{-1}(1-p_1^{\lfloor k\rfloor})^{-1}>0$$

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 $\tau_t:(s_t,a_t,s_{t+1},r_t)$ will be stored in the replay buffer based on the value $h_{p\in\mathcal{P}^n}(\tau_t.s_t)$. If the replay buffer is full, the transition with the same value of $h_{p\in\mathcal{P}^n}(\tau_t.s_t)$ 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.

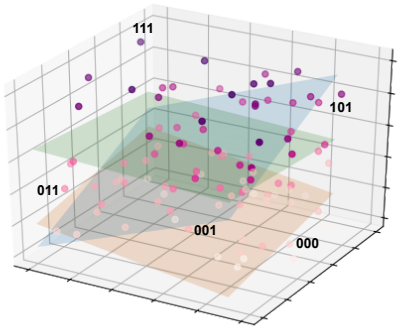


Figure 2: Given a high dimensional space, three random hype-planes are initialized based on normal distribution. Each hyper-plane splits the space into two hash areas 0 and 1. The space is split into six hash areas. We can find that, states are encoded into a binary string e.g.,{111, 101, 011, 001, 000}

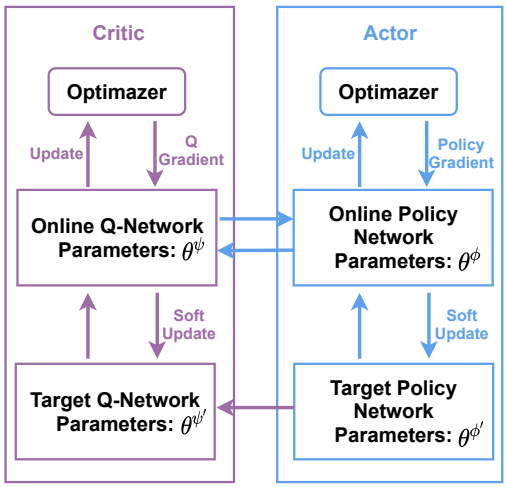


Figure 3: The structure of the DDPG which will be used as the main algorithm for our DRL agent in this work.

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 ϵ-greedy to achieve a trade-off between exploitation and exploration. LSER first draws a random probability $p \in [0, 1]$ then uses reward-driven sampling if the probability is less than a threshold ϵ_{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 ϵ-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 $h_{p \in \mathcal{P}^n}$. 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 $h_{p \in \mathcal{P}^n}$) 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 overestimation 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:

$$l(\theta_{\psi}) = \frac{1}{N} \sum_{j=1}^{N} ((r + \gamma \xi) - \psi_{\theta_{\psi}}(s_t, a_t))^2$$
$$\text{where } \xi = \psi_{\theta'_{\psi}}(s_{t+1}, \phi_{\theta'_{\phi}}(s_{t+1}))$$

where θ_{ψ} and θ_{ϕ} are the critic and actor parameters, N is the size of the mini-batch from the replay buffer, $\psi_{\theta'_{\psi}}$ and $\phi_{\theta'_{\phi}}$ 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 τ.

3 EXPERIMENTS

Algorithm 1: LSH memory by using dictionary

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

Initialize n_h hyperplanes on projection space $\mathcal{P}_{d_s}$;
Initialize empty dictionary $\mathcal{M}$;
Function *encode(s)* **is**
  for p *in* $\mathcal{P}_{d_s}$ **do**
    Calculate hash bits by using Eq.2;
  end
  return *"".join(hash bits)*
end
Function *Push(τ)* **is**
  v_h = encode($\tau.s_t$) `// get the hash code`
  if *T.size < T.capacity* **then**
    if v_h *in* $\mathcal{M}$ **then**
      $\mathcal{M}[v_h]$.append(τ);
    else
      $\mathcal{M}[v_h] = \tau$;
    end
  else
    if v_h *in* $\mathcal{M}$ *and* $\tau.r_t > \mathcal{M}[v_h][0].r$ **then**
      `// replace tuple has the minimal reward`
      $\mathcal{M}[v_h][0] = \tau$;
    end
  end
  Sort($\mathcal{M}[v_h]$) based on reward in ascending order;
end
Function *Sample(s,b)* **is**
  v_h = encode(s);
  p = 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**
      result = $\mathcal{M}[v_h][-b:]$;
    else
      result = random.sample($\mathcal{M}[v_h], b$);
    end
  else
    if $p < \epsilon_{max}$ **then**
      result = $\mathcal{M}[v_1][-b:] + \mathcal{M}[v_2][-b:]$;
    else
      result = random.sample($\mathcal{M}[v_1], b$) + random.sample($\mathcal{M}[v_2], b$);
    end
  end
  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, VirualTB 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. τ, γ, and c are set to 0.001, 0.99 and $1e^6$, 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.

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

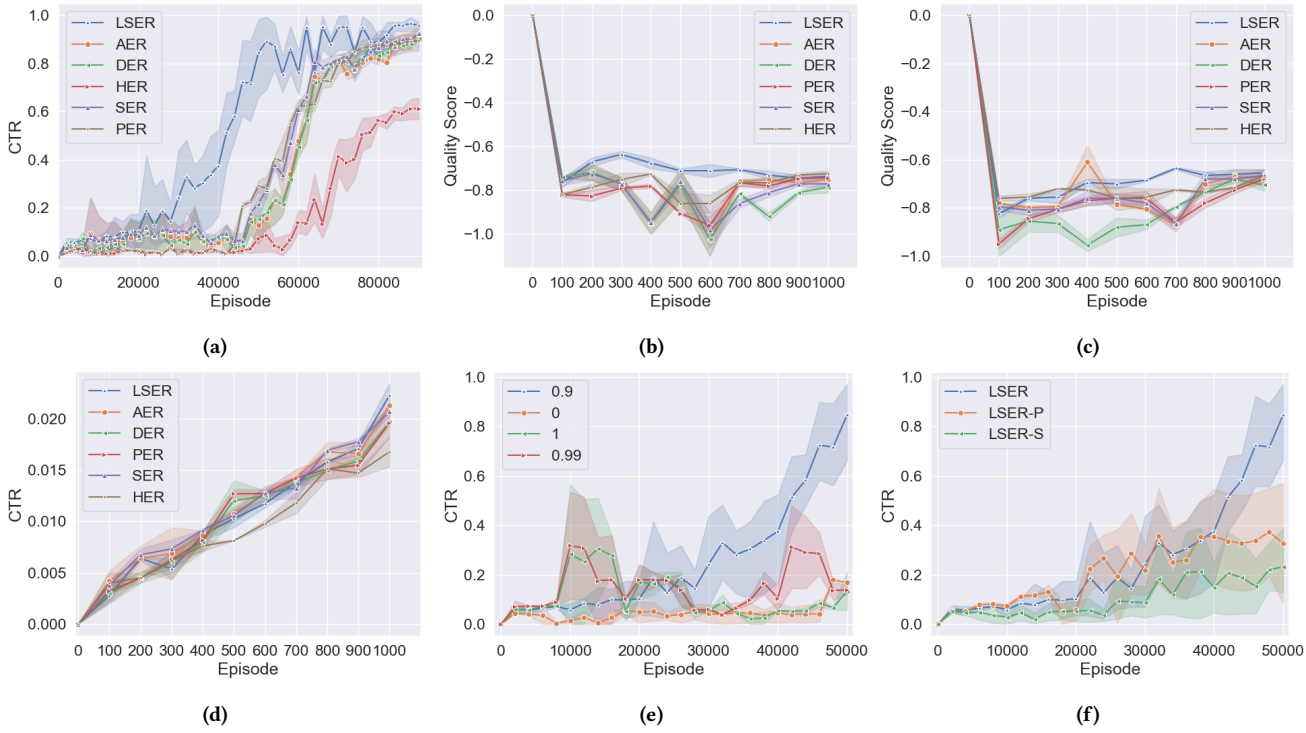


Figure 4: Result comparison with four baseline methods on VirtualTB, RecSim and RecoGym. The experiments are repeated five times, and mean values are reported. 95% confidence intervals are shown. (a) is the result for VirtualTB; (b) is the result for long-term satisfaction in RecSim; (c) is the result for interest evolution in RecSim; (d) is the result for RecoGym; (e) is the result for ϵ 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 similarly 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

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 force agents to explore more when users' interests shift.

All methods gained similar results on RecSim and RecoGym because all methods can iterate all possible combinations of states and actions. Fig. 4b, 4c and 4d show that LSER is slightly better and more stable than 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 Computational Efficiency

We show the computational efficiency of our method, LSER, by comparing the running time of the selected experience replay methods in Table 1.

While performing poorly on RecSim and RecoGym, LSER is faster than most baselines. In comparison, LSER introduces extra running time in small environments (e.g., RecSim and RecoGym) than large environments. For VirutalTB, AER takes a much longer time than all other methods, due to attention calculation [18].

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**

	Running Time (10^3 s)			
	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	<u>90.05</u>
AER	**5.18**	**4.94**	**4.12**	145.33
HER	5.33	5.11	4.20	120.33
LSER	<u>5.23</u>	<u>5.04</u>	<u>4.15</u>	**85.12**

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. 4f, where LSER-P denotes LSER with the replaced store strategy and LSER-S denotes LSER with the replaced sampling strategy. We found the sampling strategy played the most critical role in achieving good performance, as LSER-S underperformed LSER significantly. The store strategy also contributed to better performance. LSER-P was less stable (indicated by a wider error bar). but outperformed LSER at $\sim 30,000$ episodes due to the 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 a given state's hash-bits. Longer hash-bits can provide more accurate similarity measurement results but low efficiency, while shorter hash-bits can increase efficiency but decrease accuracy. This is a trade-off that balances efficiency and accuracy.

[1]The default setting of RecoGym only contains 100 users and ten products; RecSim contains 100 users and 20 products.

We report the experimental results in VirtualTB, where we evaluate the effect by varying the number hyperplanes in LSER (shown in Fig 5a). The results on the other two platforms show a similar pattern.

The performance gradually increases with more hyperplanes, but it levels off or even drops when the number of hyperplanes reaches 20.

3.7 Study of Different Backbone Algorithms

LSER is a kind of experience replay method which could be used in any DRL-based method. Hence, in this section, we would like to investigate the performance of LSER with existing DRL-based recommendation methods. To investigate this, we conduct the experiments on the following DRL RS benchmarks:

- TPGR [3] is a tree-structured reinforcement learning-based method for large-scale recommendations.
- KGRL [5] is a reinforcement learning-based method that utilizes a knowledge graph for recommendations.
- PGPR [35] is a knowledge graph reasoning-based method for the explainable recommendation.

All of those three baselines use the normal experience replay method. We will replace it with LSER to see the performance. Moreover, those three methods are evaluated in different experimental settings, which makes it hard to conduct a unified comparison. Hence, we conduct such experiments in VirtualTB. The results can be found in Figure 5b.

As we can find, all of those methods have a considerable improvement when LSER replaces the experience replay method. PGPR shows a significant improvement after the LSER is incorporated. The possible reason is that PGPR is a knowledge graph reasoning-based method that relies on path reasoning, which LSER's similarity measurement mechanism can benefit from as similar knowledge paths could lead to the same result.

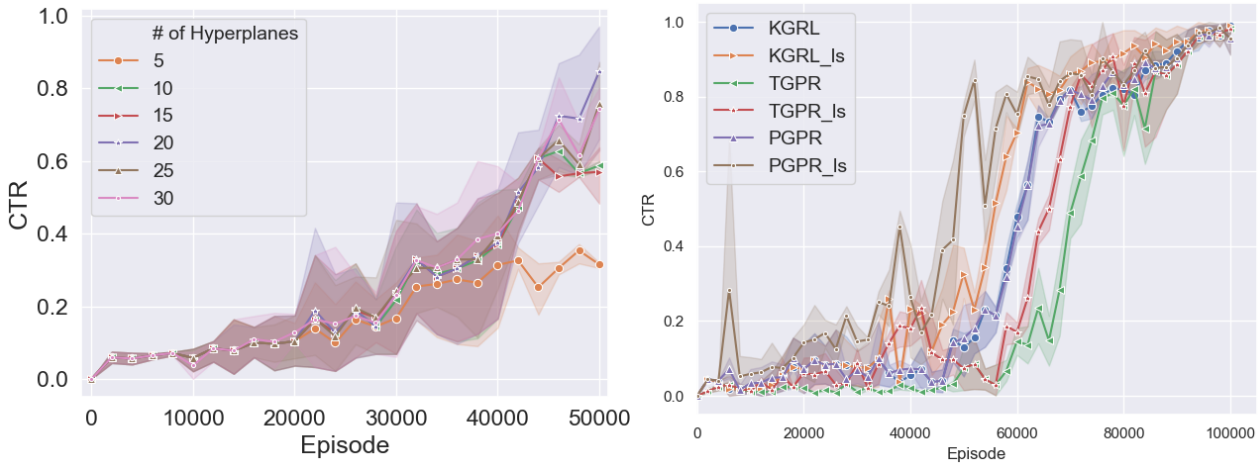
3.8 Discussion

Figure 4a shows LSER suffers instability after reaching the first peak at episode $\sim 50,000$. Different from the other methods, LSER can quickly reach the optimal policy but suffers from fluctuation. That indicates ϵ-greedy tends to lead the agent towards learning from low-priority experiences after the optimal policy is reached. We alleviate the issue by adjusting the value of ϵ.

Here, we tried $\epsilon = \{0, 0.9, 0.99, 1\}$ to determine the best choice of the ϵ on VirtualTB. The results are shown in Fig. 4e, where $\epsilon = 1$ corresponds to greedy sampling while $\epsilon = 0$ 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 T_r. This strategy allows the agent to find a near-optimal policy at an early stage. We examined the performance under T_r=0.95, which delivers a better training process.

4 RELATED WORK

Zheng et al. [42] first introduced DRL to recommender systems, and Zhao et al. [40] further exploited by under page-wise recommendation scenario. Both of these two works use DQN to embed user and item information for news recommendations, whereas



(a) Performance Comparison of different number of hyperplanes

(b) Results for different DRL RS with LSER

Figure 5: Results for ablations and different DRL agents.

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

[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 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (Eds.). 5048–5058.

[2] Xueying Bai, Jian Guan, and Hongning 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'Alché-Buc, E. Fox, and R. Garnett (Eds.), Vol. 32. Curran Associates, Inc., 10735–10746. https://proceedings.neurips.cc/paper/2019/file/e49eb6523da9e1c347bc148ea8ac55d3-Paper.pdf

[3] Haokun Chen, Xinyi Dai, Han Cai, Weinan Zhang, Xuejian Wang, Ruiming Tang, Yuzhou Zhang, and Yong Yu. 2019. Large-scale interactive recommendation with tree-structured policy gradient. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. AAAI Press, 3312–3320.

[4] Shi-Yong Chen, Yang Yu, Qing Da, Jun Tan, Hai-Kuan Huang, and Hai-Hong Tang. 2018. Stabilizing reinforcement learning in dynamic environment with application to online recommendation. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 1187–1196.

[5] Xiaocong Chen, Chaoran Huang, Lina Yao, Xianzhi Wang, Wenjie Zhang, et al. 2020. Knowledge-guided deep reinforcement learning for interactive recommendation. In *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 1–8.

[6] Xiaocong Chen, Lina Yao, Julian McAuley, Guangling Zhou, and Xianzhi Wang. 2021. A Survey of Deep Reinforcement Learning in Recommender Systems: A Systematic Review and Future Directions. *arXiv preprint arXiv:2109.03540* (2021).

[7] Tim De Bruin, Jens Kober, Karl Tuyls, and Robert Babuška. 2015. The importance of experience replay database composition in deep reinforcement learning. In *Deep reinforcement learning workshop, NIPS*.

[8] Gabriel Dulac-Arnold, Richard Evans, Hado van Hasselt, Peter Sunehag, Timothy Lillicrap, Jonathan Hunt, Timothy Mann, Theophane Weber, Thomas Degris, and Ben Coppin. 2015. Deep reinforcement learning in large discrete action spaces. *arXiv preprint arXiv:1512.07679* (2015).

[9] Scott Fujimoto, Herke Hoof, and David Meger. 2018. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*. PMLR, 1587–1596.

[10] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. 2018. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*. PMLR, 1861–1870.

[11] Dan Horgan, John Quan, David Budden, Gabriel Barth-Maron, Matteo Hessel, Hado van Hasselt, and David Silver. 2018. Distributed Prioritized Experience Replay. In *International Conference on Learning Representations*.

[12] Yuenan Hou, Lifeng Liu, Qing Wei, Xudong Xu, and Chunlin Chen. 2017. A novel ddpg method with prioritized experience replay. In *2017 IEEE international conference on systems, man, and cybernetics (SMC)*. IEEE, 316–321.

[13] 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.

[14] Eugene Ie, Vihan Jain, Jing Wang, Sanmit Narvekar, Ritesh Agarwal, Rui Wu, Heng-Tze Cheng, Tushar Chandra, and Craig Boutilier. 2019. SLATEQ: a tractable decomposition for reinforcement learning with recommendation sets. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*. 2592–2599.

[15] Eugene Ie, Chih wei Hsu, Martin Mladenov, Vihan Jain, Sanmit Narvekar, Jing Wang, Rui Wu, and Craig Boutilier. 2019. RecSim: A Configurable Simulation Platform for Recommender Systems. (2019). arXiv:1909.04847 [cs.LG]

[16] Piotr Indyk and Rajeev Motwani. 1998. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*. 604–613.

[17] David Isele and Akansel Cosgun. 2018. Selective experience replay for lifelong learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 32.

[18] Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. 2020. Reformer: The Efficient Transformer. In *International Conference on Learning Representations*. https://openreview.net/forum?id=rkgNKkHtvB

[19] Vijay R Konda and John N Tsitsiklis. 2000. Actor-critic algorithms. In *Advances in neural information processing systems*. 1008–1014.

[20] 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).

[21] Feng Liu, Huifeng Guo, Xutao Li, Ruiming Tang, Yunming Ye, and Xiuqiang He. 2020. End-to-End Deep Reinforcement Learning based Recommendation with Supervised Embedding. In *Proceedings of the 13th International Conference on Web Search and Data Mining*. 384–392.

[22] Jieliang Luo and Hui Li. 2020. Dynamic experience replay. In *Conference on Robot Learning*. PMLR, 1191–1200.

[23] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* (2013).

[24] John Nolan. 2003. *Stable distributions: models for heavy-tailed data*. Birkhauser New York.

[25] Guido Novati and Petros Koumoutsakos. 2019. Remember and forget for experience replay. In *International Conference on Machine Learning*. PMLR, 4851–4860.

[26] Feiyang Pan, Qingpeng Cai, Pingzhong Tang, Fuzhen Zhuang, and Qing He. 2019. Policy gradients for contextual recommendations. In *The World Wide Web Conference*. 1421–1431.

[27] 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.

[28] David Rohde, Stephen Bonner, Travis Dunlop, Flavian Vasile, and Alexandros Karatzoglou. 2018. Recogym: A reinforcement learning environment for the problem of product recommendation in online advertising. *arXiv preprint arXiv:1808.00720* (2018).

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

[30] 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.

[31] Jing-Cheng Shi, Yang Yu, Qing Da, Shi-Yong Chen, and An-Xiang Zeng. 2019. Virtual-taobao: Virtualizing real-world online retail environment for reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 4902–4909.

[32] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George 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. *nature* 529, 7587 (2016), 484–489.

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

[34] Lu Wang, Wei Zhang, Xiaofeng He, and Hongyuan Zha. 2018. Supervised reinforcement learning with recurrent neural network for dynamic treatment recommendation. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2447–2456.

[35] Yikun Xian, Zuohui Fu, S Muthukrishnan, Gerard de Melo, and Yongfeng Zhang. 2019. Reinforcement Knowledge Graph Reasoning for Explainable Recommendation. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, 285–294.

[36] Daochen Zha, Kwei-Herng Lai, Kaixiong 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

[37] Shuai Zhang, Lina Yao, Aixin Sun, and Yi Tay. 2019. Deep learning based recommender system: A survey and new perspectives. *ACM Computing Surveys (CSUR)* 52, 1 (2019), 1–38.

[38] Kangzhi Zhao, Xiting Wang, Yuren Zhang, Li Zhao, Zheng Liu, Chunxiao Xing, 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.

[39] Xiangyu Zhao, Long Xia, Jiliang Tang, and Dawei Yin. 2019. Deep reinforcement learning for search, recommendation, and online advertising: a survey. *SIGWEB Newsl.* 2019, Spring (2019), 4:1–4:15. https://doi.org/10.1145/3320496.3320500

[40] 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.

[41] 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.

[42] 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.