Ren, Z., Huang, N., Wang, Y., Ren, P., Ma, J., Lei, J., Shi, X., Luo, H., Jose, J. M. and Xin, X. (2023) Contrastive State Augmentations for Reinforcement Learning-Based Recommender Systems. In: 46th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR23), Taipei, Taiwan, 23-27 July 2023, pp. 922-931. ISBN 9781450394086 (doi: 10.1145/3539618.3591656)

There may be differences between this version and the published version. You are advised to consult the publisher's version if you wish to cite from it.

https://eprints.gla.ac.uk/299230/

Deposited on: 23 May 2023

# Contrastive State Augmentations for Reinforcement Learning-Based Recommender Systems

Zhaochun Ren*
Shandong University
Qingdao, China
zhaochun.ren@sdu.edu.cn

Na Huang*
Shandong University
Qingdao, China
hn.z@mail.sdu.edu.cn

Yidan Wang
Shandong University
Qingdao, China
yidanwang@mail.sdu.edu.cn

Pengjie Ren
Shandong University
Qingdao, China
jay.ren@outlook.com

Jun Ma
Shandong University
Qingdao, China
chenzhumin@sdu.edu.cn

Jiahuan Lei
Meituan
Beijing, China
leijiahuan@meituan.com

Xinlei Shi
Meituan
Beijing, China
shixinlei@meituan.com

Hengliang Luo
Meituan
Beijing, China
luohengliang@meituan.com

Joemon M Jose
University of Glasgow
London, UK
Joemon.Jose@glasgow.ac.uk

Xin Xin†
Shandong University
Qingdao, China
xinxin@sdu.edu.cn

## ABSTRACT

Learning reinforcement learning (RL)-based recommenders from historical user-item interaction sequences is vital to generate high-reward recommendations and improve long-term cumulative benefits. However, existing RL recommendation methods encounter difficulties (i) to estimate the value functions for states which are not contained in the offline training data, and (ii) to learn effective state representations from user implicit feedback due to the lack of contrastive signals.

In this work, we propose **c**ontrastive **s**tate **a**ugmentations (CSA) for the training of RL-based recommender systems. To tackle the first issue, we propose four state augmentation strategies to enlarge the state space of the offline data. The proposed method improves the generalization capability of the recommender by making the RL agent visit the local state regions and ensuring the learned value functions are similar between the original and augmented states. For the second issue, we propose introducing contrastive signals between augmented states and the state randomly sampled from other sessions to improve the state representation learning further.

To verify the effectiveness of the proposed CSA, we conduct extensive experiments on two publicly accessible datasets and one dataset collected from a real-life e-commerce platform. We also conduct experiments on a simulated environment as the online evaluation setting. Experimental results demonstrate that CSA can effectively improve recommendation performance.

## CCS CONCEPTS

• **Information systems** → **Recommender systems**; **Retrieval models and ranking**; **Novelty in information retrieval**.

## KEYWORDS

Recommender system, Reinforcement learning, Contrastive learning, Data augmentation, Sequential recommendation

## 1 INTRODUCTION

Sequential recommendation shows promising improvement in predicting users' dynamic interests. It has been successfully deployed to provide personalized services in various application scenarios, such as e-commerce platforms, social networks, and lifestyle apps [18, 31, 39, 46]. Recent advances in deep neural networks inspire the recommendation community to adopt various kinds of models for modelling user-item interaction sequences, e.g., Markov

chains [29, 30], recurrent neural networks [14, 15], convolutional neural networks [37, 50], and attention-based methods [19, 34]. These methods are used to characterize the correlation among item transitions and learn representations of user preference. Although these methods have shown promising performance, they are usually trained with pre-defined supervision signals, such as next-item or random masked item predictions. Such supervised training of recommenders can result in sub-optimal performance since the model is purely learned by a loss function based on the discrepancy between model prediction and the supervision signal. The supervised loss may not match the expectation from the perspective of service providers, e.g., improving long-term benefits or promoting high-reward recommendations.

To maximize cumulative gains with more flexible reward settings, reinforcement learning (RL) has been applied to the sequential recommendation task. An obstacle in applying existing RL methods to recommendation is that conventional RL algorithms belong to a fundamentally online learning paradigm. Such a learning process of online RL involves iteratively collecting experiences by interacting with the user. However, this iterative online approach would be costly and risky for real-world recommender systems. An appealing alternative is utilizing offline RL methods, which target on learning policies from logged data without online explorations affecting the user experience [7, 22]. Although there exists some research focusing on offline RL [8, 47, 52], how to design appropriate offline RL solutions for the sequential recommendation task remains an open research challenge due to the following limitations:

- Potentially huge user state space limits the generalization capability of the offline RL algorithms. Since offline RL algorithms aim to learn policies without online explorations, these algorithms can only investigate the state-action pairs that occurred in the logged training data. During the model inference, there could be new and out-of-distribution user states. Besides, the state transition probability could also be different from the offline data. Consequently, the RL recommendation agent could suffer from severe distribution shift problems, resulting in inaccurate estimating of the value functions for the data that is not in the same distribution as the offline training set.
- The lack of contrastive signals makes the RL agent fail to learn effective state representations. Modern recommender systems are usually trained based on implicit feedback data, which only contains positive feedback (e.g., clicks and purchases). The lack of negative feedback could lead to a situation in which the RL agent cannot know which state is bad or which action should be avoided for a given state. Given the sparse user-item implicit interactions, how to improve the data efficiency to learn effective state representations still needs to be investigated to improve the performance of RL-based recommender systems further.

To address the above issues, we propose a simple yet effective training framework, which explores contrastive state augmentations (CSA) for RL-based recommender systems. More precisely, given an input item sequence, we use a sequential recommendation model to map the sequence into a hidden state, upon which we stack two final output layers. One is trained with the conventional supervised cross-entropy loss, while the second is trained through double Q-learning [11]. To tackle the first limitation, we propose

four state augmentation strategies for the RL output layer to enlarge the state space of the offline training data. Such an approach smooths the state space by making the RL agent explicitly visit the local state regions and ensuring the learned value functions are similar among the original sequence state and the augmented states since small perturbations in the given observation should not lead to drastically different value functions. For the second limitation, we use contrastive learning on the RL output layer to "pull" the representations of different augmentations of the same sequence state towards each other while "pushing" the state representations from different sequences apart.

Finally, we co-train the supervised loss, the RL loss over the original and augmented states, and the contrastive loss from the logged implicit feedback data. To verify the effectiveness of our method, we implement CSA on three state-of-the-art sequential recommendation models and conduct experiments on two publicly accessible benchmark datasets and one dataset collected from a real-life e-commerce platform. We also conduct experiments on a simulated environment to verify whether CSA can help to improve cumulative gains on multi-round recommendation scenarios. Experimental results demonstrate the superior performance of the proposed CSA.

To summarize, our main contributions are as follows:

- We propose a simple yet effective contrastive state augmentation method to train RL-based recommender systems. The proposed approach can be seen as a general framework, which can be incorporated with several off-the-shelf sequential models.
- We propose four state augmentation strategies to improve the generalization capability of the RL recommendation agent. Besides, we propose a contrastive loss to improve the state representation learning.
- Extensive experiments conducted on three real-world datasets and one simulated online environment demonstrate the effectiveness of our proposed approach.

## 2 RELATED WORK

This section provides a literature review regarding sequential recommendation, reinforcement learning, and contrastive learning.

### 2.1 Sequential Recommendation

Early sequential recommendation methods mainly rely on the Markov Chain (MC). MC-based methods estimate an item-item transition probability matrix and utilize it to predict the next item given a user's last interactions. Rendle et al. [30] combined matrix factorization and the first-order MC to capture both general and short-term user interests. Methods with high-order MCs that consider longer interaction sequences were also developed in [12, 13]. Many deep learning-based approaches have recently been proposed to model the user-item interaction sequences more effectively. Hidasi et al. [14] firstly introduced Gated Recurrent Units (GRU) into the session-based recommendation task, and a surge of following variants modified this model by incorporating pair-wise loss function [15], copy mechanism [28], memory networks [16, 17] and hierarchical structures [27], etc. However, RNN-based methods assume that the adjacent items in a session have sequential dependencies, which may fail to capture the skip signals. Tang and

Wang [37] and Yuan et al. [51] proposed to utilize convolutional neural networks (CNN) to model sequential patterns from local features of the embeddings of previous items. Kang and McAuley [19] proposed exploiting the well-known Transformer [38] in the sequential recommendation field.

Regarding the learning task, most existing sequential recommendation methods utilize the next-item prediction task [5, 14, 19, 51]. Besides, self-supervised learning has demonstrated its effectiveness in representation learning by constructing training signals from raw data other than external labels. Sun et al. [34] proposed to use the task of predicting random masked items to train sequential recommenders. Zhou et al. [55] proposed four auxiliary self-supervised tasks to maximize the mutual information among attributes, items, and sequences. Xia et al. [44] proposed a self-supervision task to maximize the mutual information between sequence representations learned from hypergraphs.

Despite the advances of the above methods, they are trained to minimize the discrepancy between model predictions and predefined (self-)supervision signals. Such learning signals may not match the recommendation expectation, e.g., improving cumulative gains in one interaction session.

## 2.2 Reinforcement Learning

RL has shown promising improvement to increase cumulative gains in the long-run. Conventional RL methods can be categorized into on-policy methods (e.g., policy gradient [36]) and off-policy methods (e.g., Q-learning [24, 41], actor-critic [10, 20]). On-policy methods target on learning policies through real-time interactions with the environment. Off-policy methods exploit a replay buffer to store past experiences and thus improve the data efficiency of the RL algorithm. Both on-policy and off-policy methods need to perform online explorations to collect the training data[1]. On the contrary, offline RL [23] aims to train the agent from fixed data without new explorations. Recent research has started to investigate the offline RL problem [7, 8, 21], e.g., addressing the over-estimation of value functions [7, 21] or learning from expert demonstrations [8].

RL has recently been introduced into recommender systems because it considers users' long-term engagement [54, 58]. Zhao et al. [53] proposed to generate list-wise and page-wise recommendations using RL. To address the problem of distribution shift under the off-policy settings, Chen et al. [2] proposed utilizing propensity scores to perform the off-policy correction. However, such methods suffer from high variances in the estimated propensity score. Model-based RL approaches [4, 57] firstly build a model to simulate the environment. The policy is then trained through interactions with the constructed simulator. Xin et al. [49] proposed to infuse self-supervision signal to improve the training of the RL agent. Besides, contextual information has been considered to enhance the RL process for recommendations. For example, Xian et al. [45] proposed a policy gradient approach to extract paths from knowledge graphs and regard these paths as the interpretation of the recommendation process. Wang et al. [40] fused historical and future knowledge to guide RL-based sequential recommendation.

---

[1]We follow the definition of [23] that both on-policy methods (e.g., policy gradient) and off-policy methods (e.g., Q-learning and actor-critic) are online RL methods.

Given the large user state space, how to perform effective offline learning for RL-based sequential recommendation to maximize long-term cumulative benefits remains an open research challenge.

## 2.3 Contrastive Learning

Recently, contrastive learning achieves remarkable successes in various fields, such as speech processing [25], computer vision [3], as well as natural language processing [9, 43]. By maximizing mutual information among the positive transformations of the data itself while improving discrimination ability to the negatives, it discovers the semantic information shared by different views and gets high-quality representations. As a result, various recommendation methods [42, 48, 55] employ contrastive learning to optimize the representation learning. Specifically, Zhou et al. [55] proposed to adopt a pre-train and fine-tuning strategy and utilize contrastive learning during pre-training to incorporate correlations among item meta information. Wu et al. [42] proposed a multi-task framework with contrastive learning to improve the graph-based collaborative filtering methods. Xie et al. [48] proposed to utilize a contrastive objective to enhance user representations. They used item crop, item mask, and item reorder as data augmentation approaches to construct contrastive signals. Qiu et al. [26] performed contrastive self-supervised learning based on dropout.

Different from existing works which focus on utilizing contrastive learning under the (self-)supervised paradigm, our research explores contrastive signals to improve the representation learning for RL-based recommenders.

## 3 METHOD

In this section, we first formulate the recommendation task of our research (Sec. 3.1). Then we introduce the primary settings of the RL setup (Sec. 3.2). The state augmentation is detailed in section 3.3. Finally, the contrastive learning for RL-based recommender is described in section 3.4. Figure 1 shows the overview of CSA.

### 3.1 Task Formulation

Assume that we have a set of items, denoted by $I$ where $i \in I$ denotes an item. Then a user-item interaction sequence can be represented as $x_{1:t} = \{i_1, ..., i_j, ..., i_t\}$, where $t$ is the length of interactions and $i_j$ is the item that the user interacted with at the $j$-th time step. The goal of conventional sequential recommendation is to predict the next item that the user is likely to interact with (i.e., $i_{t+1}$) given the sequence of previous interactions $x_{1:t}$.

Generally speaking, one can use a sequential model $G(\cdot)$ (e.g., GRU) to map the input sequence $x_{1:t}$ into a hidden representation $\mathbf{s}_t = G(x_{1:t})$, which can be seen as a general encoding process. Based on the hidden representation, one can utilize a decoder to map the representation into a ranking score for the next time step as $\mathbf{y}_{t+1} = f(\mathbf{s}_t) = [y_1, y_2, ..., y_{|I|}] \in \mathbb{R}^{|I|}$. $f$ is usually defined as a simple, fully connected layer or the inner product with candidate item embeddings. During the model inference, top-$k$ items with the highest ranking scores are selected as the recommended items. Note that in this research, we expect the recommended items can lead to high-reward user feedback (e.g., purchases in the e-commerce scenario) or better cumulative benefits in the long-run.
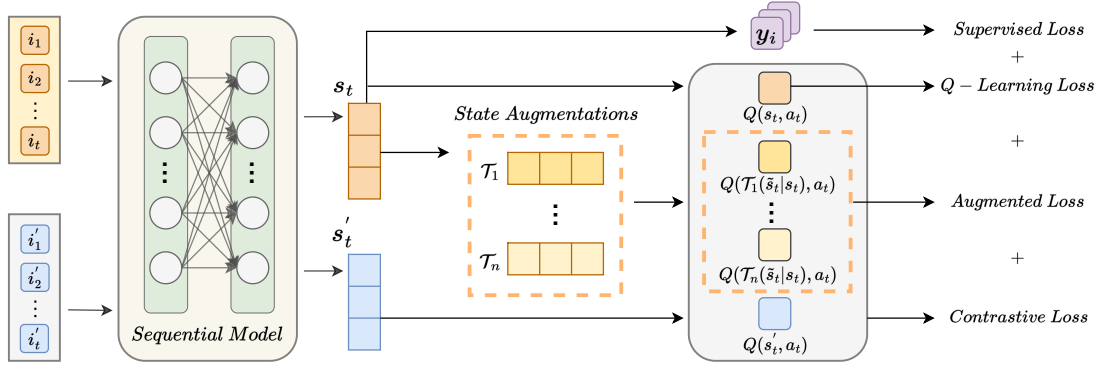
**Figure 1: The overview of CSA.** $\{i_1, i_2, ..., i_t\}$ denotes the original input sequence. $\{\mathcal{T}_1, ..., \mathcal{T}_n\}$ denotes state augmentations. **The base sequential model is trained jointly through the supervised loss, the RL loss over the original state (i.e, Q-learning loss) and the augmented states (i.e., augmented loss), and the contrastive loss over the comparison between** $\{i'_1, i'_2, ..., i'_t\}$ **which is a sampled sequence from another session.**

## 3.2 Reinforcement Learning Setup

From the perspective of RL, a sequential recommendation can be framed as a Markov Decision Process (MDP), in which the recommendation agent interacts with the environments (users) by sequentially recommending items to maximize the cumulative rewards. The MDP can be represented by a tuple of $(\mathcal{S}, \mathcal{A}, \rho, r, \gamma)$:

- $\mathcal{S}$ is the state space of the environment. It is modeled by the user's historical interactions with the items. Specifically, the state of the user can be denoted as the hidden representation from the sequential model: $\mathbf{s}_t = G(x_{1:t}) \in \mathcal{S}$.
- $\mathcal{A}$ is the action space. At each time step $t$, the agent selects an item from the candidate pool and recommends it to the user. In offline training data, we get the action at timestamp $t$ from the logged user-item interaction (i.e. $a_t = i_{t+1} \in \mathcal{I}$)[2].
- $\rho(\mathbf{s}_{t+1}|\mathbf{s}_t, a_t)$ is the transition function given the current state and the taken action, describing the distribution of the next state.
- $r(\mathbf{s}_t, a_t)$ is the immediate reward if action $a_t$ is taken under $\mathbf{s}_t$.
- $\gamma$ is the discount factor for the future reward with $\gamma \in [0, 1]$.

The RL algorithm seeks a target policy $\pi_\theta(a|\mathbf{s})$, where $\theta$ denotes the parameters, to translate the user state $\mathbf{s} \in \mathcal{S}$ into a distribution over actions $a \in \mathcal{A}$. The policy is trained to maximize the expected cumulative discount reward in the MDP as

$$\max_{\pi_\theta} \mathop{\mathbb{E}}_{\tau \sim \pi_\theta} [\mathcal{R}(\tau)], \text{where } \mathcal{R}(\tau) = \sum_{t=0}^{|\tau|} \gamma^t r(\mathbf{s}_t, a_t). \quad (1)$$

$\tau$ denotes a trajectory of $(\mathbf{s}_t, a_t, \mathbf{s}_{t+1})$ which is sampled according to the target policy $\pi_\theta$. To train the target policy, we adopt the value-based Q-learning algorithm [33] in this work. We don't utilize the policy gradient method since it is based on the on-policy update from the Monte-Carlo sampling and needs plenty of online explorations, which may affect the user experience. Although Chen et al. [2] proposed to utilize inverse propensity scores to correct the offline data distribution, such kinds of methods could suffer from severe high variance.

Q-learning utilizes policy evaluation to calculate the Q-value, which is defined as the estimation of the cumulative reward for an action $a_t$ given the state $\mathbf{s}_t$.

The Q-value is trained by minimizing the one-step time difference (TD) error:

$$Q_{i+1} \leftarrow \mathop{\arg\min}_{Q} \mathbb{E}_{\pi_i} [r_t + \gamma Q_i(\mathbf{s}_{t+1}, a_{t+1}) - Q_i(\mathbf{s}_t, a_t)], \quad (2)$$

where $Q_i$ is the Q-value in the $i$-th step of policy evaluation. $\pi_i$ is the target policy in the $i$-th update step. When learning from logged offline data, the policy evaluation can be reformulated as

$$Q_{i+1} \leftarrow \mathop{\arg\min}_{Q} \mathbb{E}_{(\mathbf{s}_t, a_t) \sim \mathcal{D}} [r_t + \gamma Q_i(\mathbf{s}_{t+1}, a_{t+1}) - Q_i(\mathbf{s}_t, a_t)], \quad (3)$$

where $\mathcal{D}$ is the collected historical data. The policy is then improved to maximize the state-action Q-values of performing an action $a_t$ given $\mathbf{s}_t$, also known as policy improvement:

$$\pi_{i+1} \leftarrow \mathop{\arg\max}_{\pi} \mathbb{E}[Q(\mathbf{s}_t, \pi_i(a_t|\mathbf{s}_t))]. \quad (4)$$

In deep Q-learning, the Q-values are parameterized using neural networks (with parameter $\theta$) and trained using gradient descent.

In this work we utilize double Q-learning to further enhance the learning stability and follow [49] to combine supervised learning and RL through a shared base sequential model. More specifically, given the base model $G(\cdot)$ and the input item sequence $x_{1:t}$, the hidden representation can be formulated as $\mathbf{s}_t = G(x_{1:t})$, which is then fed into a fully connected layer. The layer outputs a $|\mathcal{I}|$-dimensional vector $\mathbf{y}_t$, denoting the prediction ranking scores over the item set:

$$\mathbf{y}_t = \phi(\mathbf{W}_s \mathbf{s}_t + \mathbf{b}_s), \quad (5)$$

where $\phi$ denotes the activation function. $\mathbf{W}_s$ and $\mathbf{b}_s$ are trainable parameters. Then the supervised training loss can be defined as the cross-entropy over the ranking distribution:

$$\mathcal{L}_s = -\sum_{i=1}^{|\mathcal{I}|} Y_i \log(p_i), \text{where } p_i = \frac{e^{y_i}}{\sum_{i'=1}^{|\mathcal{I}|} e^{y_{i'}}}, \quad (6)$$

where $Y_i$ indicates the ground truth and $Y_i = 1$ denotes the user interacted with the $i$-th item, otherwise $Y_i = 0$.

---

[2]In this paper, we follow the top-$k$ recommendation paradigm. We leave the recommendation of a set of items as one of our future work.

Regarding the Q-learning network, we stack the other fully connected layer upon the hidden representation $\mathbf{s}_t$. Then we calculate the Q-values as

$$Q(\mathbf{s}_t, a_t) = \phi(\mathbf{W}_r \mathbf{s}_t + \mathbf{b}_r), \tag{7}$$

where $\mathbf{W}_r$ and $\mathbf{b}_r$ are trainable parameters of the RL layer. The Q-learning loss can be defined as

$$\mathcal{L}_q = (r(\mathbf{s}_t, a_t) + \gamma \max_{a'} Q(\mathbf{s}_{t+1}, a') - Q(\mathbf{s}_t, a_t))^2. \tag{8}$$

## 3.3 State Augmentations

Due to the fact that the recommender is trained on the historical data without online explorations, the agent can only investigate the state-action pairs that occurred in the offline training data. As a result, the Q-network may fail to estimate the value functions for unseen states. To this end, we propose to use state augmentations to learn better value functions during the training stage. Based on the described RL setup, we then discuss the design of the transformations for a given state, which combines local perturbations to the states with similar value function estimation to enhance the generalization capability of the RL agent. Augmentations such as rotations, translations, colour jitters, etc., are commonly used to increase the variety of data points in the field of computer vision. Such transformations can preserve the semantics of the image after the transformation. However, in the recommendation scenario, a state augmentation strategy that is too aggressive may end up hurting the RL agent since the reward for the original state may not coincide with the reward for the augmented state. To avoid explicit modeling of reward functions, the key assumption held when performing state augmentations is that the immediate reward for the augmented state $\tilde{\mathbf{s}}_t$ should keep similar to the reward for the original state. Therefore, the choice of state augmentation needs to be a local transformation to perturb the original state. Through performing state augmentations, we are able to artificially let the agent visit states $\tilde{\mathbf{s}}_t$ that do not occur in the offline training data (i.e., $\tilde{\mathbf{s}}_t \in \mathcal{S}$ and $\tilde{\mathbf{s}}_t \notin \mathcal{D}$), thus increasing the robustness of the trained RL-agent.

Precisely, we introduce the following augmentation strategies:

- **Zero-mean Gaussian noise.** Specifically, we add a zero-mean Gaussian noise to the original state as

$$\tilde{\mathbf{s}}_t = \mathbf{s}_t + \epsilon_n, \text{where } \epsilon_n \in \mathcal{N}\left(0, \sigma^2 \mathbf{I}\right). \tag{9}$$

$\sigma$ is a hyperparameter to control the variance of Gaussian noise. $\mathbf{I}$ denotes an identity matrix.

- **Uniform noise.** Similar to the Gaussian noise, we add a uniform noise to the original state as

$$\tilde{\mathbf{s}}_t = \mathbf{s}_t + \epsilon_u, \text{where } \epsilon_u \in \mathbf{U}\left(\alpha, \beta\right). \tag{10}$$

$\alpha$ and $\beta$ are hyperparameters to control the noise variance. $\alpha = 0.001$ in our experimental settings.

- **Item mask.** "Word dropout" is a common data augmentation technique to avoid overfitting in various natural language processing tasks, such as sentence generation [1], sentiment analysis [6], and question answering [32]. It usually masks a random input word with zeros. Inspired by "Word dropout", we propose to apply a random item mask as the third augmentation method.

For each user historical sequence $x_{1:t}$ with $t > T$, we randomly mask one item of the sequence[3]. If the item in the sequence is masked, it is replaced by a special token [mask].

- **Dimension dropout.** Similar to the item mask, we replace one random dimension with zero in the state as $\tilde{\mathbf{s}}_t = \mathbf{s}_t \cdot \mathbf{1}$, where $\mathbf{1}$ is a vector of 1s with one 0 randomly sampled from $Bernoulli(p)$. $p$ is a hyperparameter. This transformation preserves all-but-one intrinsic dimension of the original state.

We denote a data augmentation transformation as $\mathcal{T}(\tilde{\mathbf{s}}_t|\mathbf{s}_t)$ with $\mathbf{s}_t \in \mathcal{D}$. We perform multiple (i.e., $n$ times) augmentations for a given state and then the $i$-th step of policy evaluation over the augmented states can be formulated as

$$Q_{i+1} \leftarrow \arg\min_Q \mathbb{E}_\mathcal{D} \sum_{j=1}^{n} [r_t + \gamma Q_i(\mathcal{T}_j(\tilde{\mathbf{s}}_{t+1}|\mathbf{s}_{t+1}), a_{t+1}) \tag{11}$$
$$- Q_i(\mathcal{T}_j(\tilde{\mathbf{s}}_t|\mathbf{s}_t), a_t)].$$

And the corresponding augmented loss can be formulated as

$$\mathcal{L}_a = - \sum_{j=1}^{n} (r(\mathbf{s}_t, a_t) + \gamma \max_{a'} Q(\mathcal{T}_j(\tilde{\mathbf{s}}_{t+1}|\mathbf{s}_{t+1}), a') - Q(\mathcal{T}_j(\tilde{\mathbf{s}}_t|\mathbf{s}_t), a_t))^2. \tag{12}$$

The main difference between our proposed objective and the standard Q-learning is that we augment the Bellman error in Eq. (8) to the sum error over $n$ different augmentations of the same state. Intuitively, this will help improve the consistency of the Q-value within some perturbation field of the current state $\mathbf{s}_t$, since the Bellman backups are taken over $n$ different views of the same state. By combining such local perturbations with the states we can distill the benefits of the augmentations to learn a more robust policy that can generalize better to unseen states when deployed in an online recommendation scenario.

## 3.4 Contrastive Training for RL Recommenders

We address the issue of large state space and lack of negative signals in implicit feedback. Inspired by the SimCLR framework [3] for learning visual representation, we further propose a reinforcement learning-based contrastive loss to obtain more effective state representations. In the supervised settings, the contrastive loss can be minimized by defining the same instance as positive pairs while remaining others as negative pairs. While in the RL-based contrastive loss, we define different augmentations of the same state as positive pairs. As for negatives, we simply sample another state from another sequence randomly. Hence, the RL-based contrastive loss is to pull the representations of different augmentations of the same sequence state towards each other while pushing the state representations from different sequences apart. To achieve this, the contrastive loss learns to minimize the Q-value difference between augmented views of the same state and maximize the Q-value difference between the states derived from other sequences. Specifically, for each state $\mathbf{s}_t$ in a mini-batch $B$, we firstly sample another state $\mathbf{s}_t'$ from another sequence randomly. Then we apply $n$ same augmentation strategy $\mathcal{T}$ to obtain $n$ different views of $\mathbf{s}_t$ (i.e. $\sum_{j=1}^{n} \mathcal{T}_j(\tilde{\mathbf{s}}_t|\mathbf{s}_t)$) and average them. Thus the contrastive loss has

---

[3]We don't apply item mask to sequences whose length is shorter than $T$.

the form:

$$\mathcal{L}_c = -\log \delta((Q(\mathbf{s}'_t, a_t) - \frac{1}{n}\sum_{j=1}^{n} Q(\mathcal{T}_j(\tilde{\mathbf{s}}_t | \mathbf{s}_t), a_t))^2$$
$$- (Q(\mathbf{s}_t, a_t) - \frac{1}{n}\sum_{j=1}^{n}(Q(\mathcal{T}_j(\tilde{\mathbf{s}}_t | \mathbf{s}_t), a_t))^2), \tag{13}$$

where $\delta$ is the sigmoid function. Similar contrastive operations can also be conducted for actions.

Finally, we jointly train the weighted sum of the supervised loss, the Q-learning loss, the augmented loss, and the contrastive loss on the offline implicit feedback data:

$$\mathcal{L} = w_s\mathcal{L}_s + w_q\mathcal{L}_q + w_a\mathcal{L}_a + w_c\mathcal{L}_c, \tag{14}$$

where $w_s, w_q, w_a, w_c$ are weights of the corresponding loss respectively. Without the lose of generality and to avoid trivial hyperparameter tuning, in our experiments we set all the loss weights to 1 as our default settings.

## 3.5 Discussion

The proposed CSA can be used as a learning framework and integrated with existing recommendation models, as long as the models can map the input sequence into a hidden state. This keeps inline with most deep learning-based recommendation models introduced over recent years. To verify the performance of CSA, we choose a SOTA recommendation model, i.e., SQN [49], which combines supervised loss and deep Q-learning. While the proposed CSA can also work for pure RL-based recommendations models. e.g., DQN [35] and CQL [22]. CSA can be seen as an attempt to explore state augmentations and contrastive learning to improve the RL agent trained on the biased state-action data in the offline setting. The proposed methods are for the generic recommendation, i.e., more flexible reward settings such as novelty and dwell time can also be used as the reward function.

## 4 EXPERIMENTAL SETUP

In this section, we detail experimental setups to verify the effectiveness of the proposed CSA. We aim to answer the following research questions: **RQ**1 How does CSA perform in the offline evaluation setting when integrated with existing sequential models? **RQ**2 How does CSA perform when generating multi-round recommendations in the simulated online environment? **RQ**3 How does the state augmentation affect CSA performance? **RQ**4 How does contrastive learning affect performance?

## 4.1 Datasets

We evaluate our method on two publicly accessible datasets: RC15, RetailRocket; and one dataset collected from a real-life serving e-commerce platform:meituan. The RC15 dataset is based on the RecSys Challenge 2015. This dataset is session-based, and each session contains a sequence of clicks and purchases. Following [49], we remove the sessions whose length is smaller than three and then sort the user-item interactions in one session according to the timestamp. RetailRocket is collected from a real-world e-commerce website. It contains sequential events of viewing and adding to the

**Table 1: Statistics of datasets.**

| Dataset | RC15 | RetailRocket | meituan |
|---|---|---|---|
| #sequences | 200,000 | 195,523 | 108,650 |
| #items | 26,702 | 70,852 | 87,987 |
| #clicks | 1,110,965 | 1,176,680 | 1,966,922 |
| #purchases | 43,946 | 57,269 | 427,701 |

cart. For consistency, we treat views as clicks and adding to the cart as purchases.

The meituan dataset consists of one week (from April 1st, 2022 to April 7th, 2022) of transaction records on the e-commerce platform. We choose users with both purchases and clicks and then sort the user-item interactions in one session according to the timestamp. We remove items that interacted less than five times and the sequences whose lengths are smaller than 3. Table 1 summarizes the detailed statistics of datasets.

## 4.2 Evaluation Metrics

We employ top-$k$ Hit Ratio (HR@$k$), top-$k$ Normalized Discount Cumulative Gain (NDCG@$k$) to evaluate the performance, which is widely used in related works [30, 49]. HR@$k$ is a recall-based metric, measuring whether the ground-truth item is in the top-$k$ positions of the recommendation list. NDCG is a rank-sensitive metric which assigns higher scores to top positions in the recommendation list. We report results of HR@{5,10,20}, NDCG@{5,10,20}. For the two publicly accessible datasets, we use the sample data splits with [49]. For the anonymous dataset, the training, validation, and test set ratio is 8:1:1. The ranking is performed among the whole item set. Note that we use both clicks and purchases for model training and report the HR and NDCG on purchase predictions since this work focuses on generating recommendations that can lead to high-reward user feedback.

## 4.3 Baselines

We integrate the proposed CSA with three state-of-the-art sequential recommendation models: (1) **GRU4Rec** [14] is a classical RNN-based method for a session-based recommendation. (2) **SASRec** [19] is a self-attention-based sequential recommendation model, which uses the multi-head attention mechanism to recommend the next item. (3) **FMLPRec** [56] stacks multiple multi-layer perceptrons (MLP) blocks to produce the representation of sequential user preference for the recommendation.

Each model is trained using the following methods: (1) **Normal** denotes training the model with the normal supervised cross-entropy loss. (2) **SQN** [49] combines the supervised loss and deep Q-learning. (3) **CSA-N** denotes CSA with the Gaussian noise augmentation. (4) **CSA-U** is CSA with the uniform noise augmentation. (5) **CSA-M** is CSA with the item mask augmentation. (6) **CSA-D** denotes CSA with the dimension dropout augmentation.

## 4.4 Implementation details

For all datasets, the input sequences are composed of the last ten items before the target timestamp. We complement the sequence with a padding item if the sequence length is less than 10. We use a mini-batch Adam optimizer to optimize the model. The batch size is

set as 256. For a fair comparison, the embedding size is set as 64 for all models. The learning rate is tuned to the best for a given model and dataset. Specifically, for GRU4Rec and SASRec, the learning rate is set as 0.01 for RC15, 0.005 for RetailRocket and 0.001 for the anonymous dataset. The number of heads in self-attention of SASRec is set as 1 according to its original paper [19]. For FMLPRec, the learning rate is set as 0.001 for all datasets. For CSA, the times of state augmentations are set as $n = 2$[4].

## 5 EXPERIMENTAL RESULTS

In this section, we present the experimental results to answer the research questions described in section 4.

### 5.1 Offline Performance Comparison (RQ1)

Table 2 and Table 3 show the offline performance comparison on the two publicly accessible datasets and the anonymous dataset, respectively.

We have the following observations: (i) Compared with the standard supervised training, it is evident that RL-based training methods perform better on all datasets, demonstrating that incorporating RL can help improve the recommendation performance. (ii) Our proposed CSA further improves the performance over SQN with almost all state augmentation strategies. On RC15 and RetailRocket, four data augmentation strategies help consistently to improve the performance, except for CSA-U with the SASRec model on RC15. On the anonymous dataset, CSA-N, CSA-U and CSA-D consistently outperform the SQN method, while CSA-M performs comparably with SQN. For cases in which CSA cannot outperform the SQN method, the reason could be that the augmented noises are too aggressive so the value functions could be changed for the augmented states. To conclude, the proposed CSA is effective to improve the RL-based recommendation performance in the offline evaluation setting.

### 5.2 Online Performance Comparison (RQ2)

*5.2.1 Kuaishou Environment.* KuaiRec is a real-world dataset collected from the recommendation logs of a video-sharing mobile app. There are 1,411 users and 3,327 videos that compose a matrix where each entry represents a user's feedback on a video. The density of this matrix is almost 100% (i.e., 99.6%), which means that we can always get a user's feedback for every item. Such a dataset can be regarded as a simulated online environment since we can get a reward for every possible action.

*5.2.2 Implementation and Evaluation.* In this environment, we conduct 10 rounds of recommendation to each user and average the obtained rewards of all users to get the immediate reward in each round. The reward is defined as the watching ratio of a video. Since the objective of RL is to maximize the cumulative gains in the long-run, we report the discounted accumulative reward of the 10-round recommendation process. Experiments are conducted three times, and the average is reported.

*5.2.3 Performance Analysis.* The performance comparison of multi-round recommendation is illustrated in Figure 2. We choose
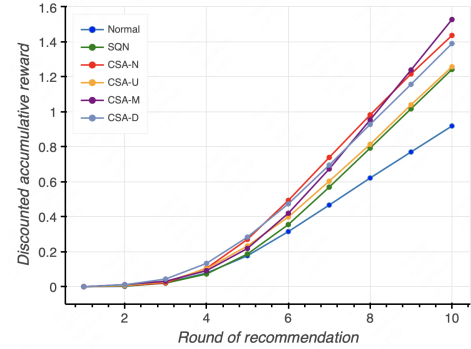
**Figure 2: Comparison of multi-round recommendation performance. GRU4Rec is the base sequential model.**

GRU4Rec as the base sequential model. It is evident that our proposed CSA significantly outperforms standard training and SQN. Besides, we can observe more insightful facts: (i) in comparison of the accumulated reward between normal supervised training and RL-based training, we find that with the number of recommendation rounds increasing, the RL-based methods can achieve more significant gains than the supervised method, indicating the intrinsic advantage of RL to maximize long-term benefits; (ii) compared with SQN, the proposed methods achieve higher accumulative rewards in each round of recommendation, demonstrating the effectiveness of the state augmentations; (iii) when comparing the performance of four state augmentation methods, we find that CSA-N achieves the highest reward in the first nine rounds. In contrast, CSA-M achieves the highest reward in the tenth round, implying that CSA-M may perform better for long sequences.

### 5.3 Effect of State Augmentations (RQ3)

In this section, we first examine how the number of state augmentations (i.e., $n$) affects the method performance. Then we investigate how the hyperparameters of state augmentation methods influence their performance. We report the experimental results on the RC15 dataset with GRU4Rec as the sequential model. Results on other datasets and sequential models show similar trends.

*5.3.1 Effect of the number of state augmentations.* The results of HR@5 and NDCG@5 are shown in Figure 3(a) and Figure 3(b) respectively. We can see that when the number of state augmentations is greater than 0, the performance is significantly improved, which demonstrates the effectiveness of the state augmentation strategies.

Considering the performance and efficiency, the number of state augmentation is set to 2 in our experiments. In addition, we can observe that method performance starts to decrease with a further increase in the augmentation number. The reason could be that too many augmentations would make more states have similar Q-values, thus increasing the risk of model collapse.

*5.3.2 Effect of state augmentation hyperparameters.* We investigate the performance sensitivity regarding to $\sigma$ in CSA-N, $\beta$ in CSA-U, $T$ in CSA-M, and $p$ in CSA-D. The results are illustrated in Figure 4(a), Figure 4(b), Figure 4(c) and Figure 4(d) respectively. We can see that the performance of CSA-N and CSA-U first improves with the increase of the $\sigma$ and $\beta$, which are hyperparameters used to control the magnitude of the noise. The best performance is achieved when

**Table 2: Offline performance comparison of different methods on RC15 and RetailRocket. Boldface denotes the highest score. HR and NG are short for Hit Ratio and NDCG, respectively.**
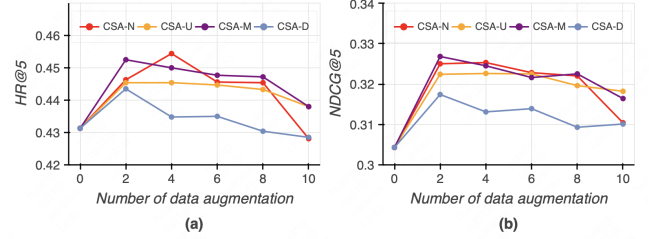
| method | | RC15 | | | | | | RetailRocket | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | HR@5 | NG@5 | HR@10 | NG@10 | HR@20 | NG@20 | HR@5 | NG@5 | HR@10 | NG@10 | HR@20 | NG@20 |
| GRU4Rec | normal | 0.4057 | 0.2802 | 0.5191 | 0.3170 | 0.6093 | 0.3398 | 0.4559 | 0.3775 | 0.5092 | 0.3948 | 0.5551 | 0.4064 |
| | SQN | 0.4313 | 0.3043 | 0.5433 | 0.3405 | 0.6332 | 0.3634 | 0.5009 | 0.4135 | 0.5538 | 0.4308 | 0.5993 | 0.4423 |
| | CSA-N | 0.4463 | 0.3250 | 0.5578 | 0.3613 | 0.6436 | 0.3830 | 0.5111 | 0.4298 | 0.5608 | 0.4459 | **0.6020** | 0.4563 |
| | CSA-U | 0.4454 | 0.3224 | 0.5558 | 0.3582 | 0.6427 | 0.3803 | 0.5075 | 0.4264 | 0.5594 | 0.4434 | 0.5989 | 0.4534 |
| | CSA-M | **0.4525** | **0.3268** | **0.5629** | **0.3627** | **0.6531** | **0.3856** | **0.5126** | **0.4358** | **0.5638** | **0.4524** | 0.6018 | **0.4621** |
| | CSA-D | 0.4435 | 0.3174 | 0.5604 | 0.3555 | 0.6466 | 0.3774 | 0.5103 | 0.4256 | 0.5583 | 0.4412 | 0.6010 | 0.4520 |
| SASRec | normal | 0.4308 | 0.3006 | 0.5512 | 0.3396 | 0.6385 | 0.3619 | 0.5335 | 0.4322 | 0.5878 | 0.4497 | 0.6330 | 0.4611 |
| | SQN | 0.4433 | 0.3098 | 0.5509 | 0.3448 | 0.6427 | 0.3681 | 0.5649 | 0.4748 | 0.6203 | 0.4929 | 0.6545 | 0.5016 |
| | CSA-N | 0.4537 | 0.3208 | **0.5705** | 0.3587 | **0.6598** | 0.3812 | **0.5874** | **0.4934** | 0.6316 | 0.5078 | 0.6719 | 0.5180 |
| | CSA-U | 0.4364 | 0.3122 | 0.5470 | 0.3482 | 0.6346 | 0.3705 | 0.5855 | 0.4914 | **0.6390** | **0.5090** | **0.6768** | **0.5185** |
| | CSA-M | **0.4550** | **0.3243** | 0.5701 | **0.3617** | 0.6535 | **0.3828** | 0.5683 | 0.4745 | 0.6269 | 0.4937 | 0.6693 | 0.5044 |
| | CSA-D | 0.4509 | 0.3184 | 0.5669 | 0.3561 | 0.6574 | 0.3792 | 0.5738 | 0.4773 | 0.6280 | 0.4950 | 0.6630 | 0.5038 |
| FMLPRec | normal | 0.4087 | 0.2929 | 0.5145 | 0.3272 | 0.6010 | 0.3493 | 0.5218 | 0.4307 | 0.5712 | 0.4468 | 0.6118 | 0.4571 |
| | SQN | 0.4205 | 0.2994 | 0.5304 | 0.3353 | 0.6199 | 0.3581 | 0.5534 | 0.4631 | 0.5991 | 0.4781 | 0.6366 | 0.4876 |
| | CSA-N | **0.4394** | **0.3209** | 0.5422 | **0.3543** | 0.6251 | **0.3752** | **0.5876** | 0.5041 | 0.6313 | 0.5183 | **0.6696** | 0.5280 |
| | CSA-U | 0.4359 | 0.3125 | 0.5420 | 0.3472 | **0.6277** | 0.3690 | 0.5870 | **0.5078** | **0.6320** | **0.5223** | 0.6675 | **0.5312** |
| | CSA-M | 0.4359 | 0.3140 | **0.5436** | 0.3489 | 0.6277 | 0.3703 | 0.5868 | 0.5004 | 0.6297 | 0.5144 | 0.6679 | 0.5242 |
| | CSA-D | 0.4391 | 0.3173 | 0.5422 | 0.3508 | 0.6268 | 0.3722 | 0.5816 | 0.4940 | 0.6254 | 0.5083 | 0.6611 | 0.5147 |

**Table 3: Offline performance comparison of methods on meituan dataset. Boldface denotes the highest score. HR and NG are short for Hit Ratio and NDCG, respectively.**

| method | | meituan | | | | | |
|---|---|---|---|---|---|---|---|
| | | HR@5 | NG@5 | HR@10 | NG@10 | HR@20 | NG@20 |
| GRU4Rec | normal | 0.5139 | 0.4352 | 0.5676 | 0.4526 | 0.6110 | 0.4636 |
| | SQN | 0.5238 | 0.4484 | 0.5783 | 0.4661 | 0.6224 | 0.4773 |
| | CSA-N | 0.5256 | 0.4528 | **0.5799** | 0.4706 | **0.6242** | 0.4818 |
| | CSA-U | **0.5270** | **0.4565** | 0.5795 | **0.4735** | 0.6220 | **0.4843** |
| | CSA-M | 0.5095 | 0.4385 | 0.5618 | 0.4554 | 0.6066 | 0.4668 |
| | CSA-D | 0.5245 | 0.4502 | 0.5774 | 0.4674 | 0.6203 | 0.4783 |
| SASRec | normal | 0.5230 | 0.4299 | 0.5816 | 0.4489 | 0.6272 | 0.4605 |
| | SQN | 0.5352 | 0.4416 | 0.5952 | 0.4612 | 0.6398 | 0.4725 |
| | CSA-N | **0.5455** | **0.4595** | 0.6016 | **0.4778** | **0.6482** | **0.4896** |
| | CSA-U | 0.5453 | 0.4560 | **0.6040** | 0.4751 | 0.6472 | 0.4861 |
| | CSA-M | 0.5331 | 0.4412 | 0.5908 | 0.4600 | 0.6368 | 0.4717 |
| | CSA-D | 0.5366 | 0.4432 | 0.5946 | 0.4620 | 0.6408 | 0.4737 |
| FMLPRec | normal | 0.5132 | 0.4310 | 0.5721 | 0.4501 | 0.6166 | 0.4614 |
| | SQN | 0.5316 | 0.4492 | 0.5877 | 0.4675 | 0.6313 | 0.4786 |
| | CSA-N | 0.5408 | **0.4663** | 0.5935 | **0.4833** | 0.6359 | 0.4941 |
| | CSA-U | **0.5415** | 0.4659 | 0.5926 | 0.4825 | **0.6395** | **0.4944** |
| | CSA-M | 0.5308 | 0.4517 | 0.5862 | 0.4697 | 0.6306 | 0.4809 |
| | CSA-D | 0.5339 | 0.4627 | 0.5888 | 0.4806 | 0.6313 | 0.4914 |



**Figure 3: Effect of the number of state augmentations.**

$T > 3$ also decreases the performance since a larger $T$ indicates a smaller portion of states will be augmented.

For CSA-D, we can find that the best performance is achieved when $p = 0.1$.

## 5.4 Effect of contrastive learning (RQ4)

*5.4.1 Ablation study of the contrastive loss.* This section examines how the contrastive learning term (i.e., $\mathcal{L}_c$) contributes to the method performance. The results of GRU4Rec and SASRec on the RC15 dataset are shown in Table 4. We can see that when the contrastive loss is removed, the method performance drops in almost all cases. Such results demonstrate the proposed contrastive term's effectiveness in improving the recommendation performance. The contrastive loss helps to learn more effective state representations by making the Q-values from different augmentations over one sequence state similar while pushing away the Q-values over the state coming from another sequence.

*5.4.2 Comparison of contrastive states and contrastive actions.* In this section, we explore the comparison between contrastive states and contrastive actions. The results of GRU4Rec and SASRec on the RC15 dataset are shown in Table 5. Combining the results of

$\sigma = 0.003$ and $\beta = 0.005$, indicating that the noises introduced at such values effectively perturb the state while maintaining the original semantic information. Then as the introduced noises become larger, the performance starts to decrease, demonstrating that too aggressive noises would also hurt the method's performance.

For CSA-M, the best performance is achieved when $T = 3$. On the one hand, masking an item could lead to a dramatic state change with poor model performance when $T < 3$. On the other hand,
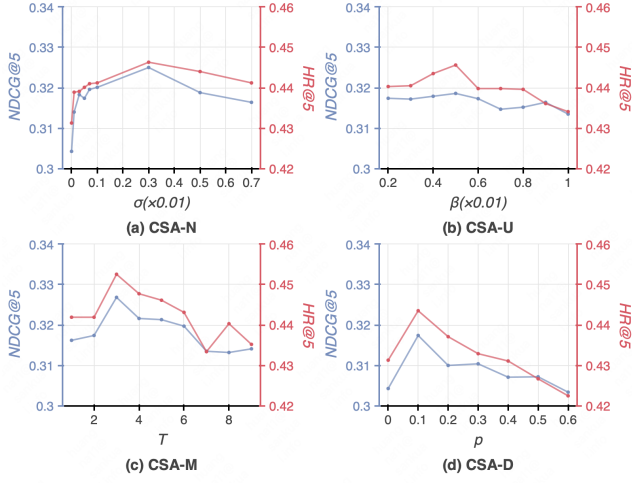
**Figure 4: Effect of state augmentation hyperparameters.**

**Table 4: Effect of $\mathcal{L}_c$. Boldface denotes the higher score. HR and NG are short for Hit Ratio and NDCG, respectively. GRU and SAS are short for GRU4Rec and SASRec. N,U,M,D denote four augmentation strategies. -c denotes the removal of $\mathcal{L}_c$.**

| method | | HR@5 | NG@5 | HR@10 | NG@10 | HR@20 | NG@20 |
|---|---|---|---|---|---|---|---|
| | | | | RC15 | | | |
| GRU | N | **0.4463** | **0.3250** | **0.5578** | **0.3613** | **0.6436** | **0.3830** |
| | N-c | 0.4463 | 0.3201 | 0.5526 | 0.3546 | 0.6335 | 0.3751 |
| | U | 0.4454 | **0.3224** | **0.5558** | **0.3582** | **0.6427** | **0.3803** |
| | U-c | **0.4458** | 0.3173 | 0.5535 | 0.3521 | 0.6427 | 0.3748 |
| | M | **0.4525** | **0.3268** | **0.5629** | **0.3627** | **0.6531** | **0.3856** |
| | M-c | 0.4477 | 0.3216 | 0.5546 | 0.3563 | 0.6429 | 0.3787 |
| | D | **0.4435** | **0.3174** | **0.5604** | **0.3555** | **0.6466** | **0.3774** |
| | D-c | 0.4431 | 0.3141 | 0.5516 | 0.3494 | 0.6383 | 0.3713 |
| SAS | N | **0.4537** | **0.3208** | 0.5705 | 0.3587 | **0.6598** | **0.3812** |
| | N-c | 0.4532 | 0.3207 | **0.5735** | **0.3599** | 0.6577 | 0.3812 |
| | U | **0.4364** | **0.3122** | **0.5470** | **0.3482** | **0.6346** | **0.3705** |
| | U-c | 0.4085 | 0.2887 | 0.5053 | 0.3200 | 0.5876 | 0.3409 |
| | M | **0.4550** | **0.3243** | **0.5701** | **0.3617** | **0.6535** | **0.3828** |
| | M-c | 0.4424 | 0.3149 | 0.5581 | 0.3523 | 0.6455 | 0.3745 |
| | D | **0.4509** | **0.3184** | **0.5669** | **0.3561** | **0.6574** | **0.3792** |
| | D-c | 0.4449 | 0.3119 | 0.5535 | 0.3473 | 0.6475 | 0.3713 |

Table 5 and Table 2, we can see that both the contrastive states and contrastive actions can help to improve the recommendation performance, obtaining better results than SQN. This observation demonstrates that introducing contrastive signals can help to improve RL-based recommender systems. Besides, we also find that in most cases, contrastive operations conducted on actions perform worse than that conducted on states. The reason is that different models have different encoding capabilities to map the input sequences into hidden states. As a result, the contrastive operations conducted on states tend to help the recommender to learn more stable state representations, leading to better results.

**Table 5: Effect of contrastive operations conducted on states and actions. Boldfae denotes the higher score. HR and NG are short for Hit Ratio and NDCG, respectively. GRU and SAS are short for GRU4Rec and SASRec. N,U,M,D denote four augmentation strategies. (s) denotes contrastive operations are conducted on states. (a) denotes contrastive opertations are conducted on actions.**

| method | | HR@5 | NG@5 | HR@10 | NG@10 | HR@20 | NG@20 |
|---|---|---|---|---|---|---|---|
| | | | | RC15 | | | |
| GRU | N(s) | **0.4463** | **0.3250** | **0.5578** | **0.3613** | **0.6436** | **0.3830** |
| | N(a) | 0.4375 | 0.3152 | 0.5470 | 0.3508 | 0.6319 | 0.3724 |
| | U(s) | **0.4454** | **0.3224** | **0.5558** | **0.3582** | **0.6427** | **0.3803** |
| | U(a) | 0.4391 | 0.3151 | 0.5503 | 0.3513 | 0.6381 | 0.3736 |
| | M(s) | **0.4525** | **0.3268** | **0.5629** | **0.3627** | **0.6531** | **0.3856** |
| | M(a) | 0.4419 | 0.3162 | 0.5565 | 0.3535 | 0.6408 | 0.3749 |
| | D(s) | **0.4435** | **0.3174** | **0.5604** | **0.3555** | **0.6466** | **0.3774** |
| | D(a) | 0.4320 | 0.3053 | 0.5392 | 0.3399 | 0.6289 | 0.3626 |
| SAS | N(s) | **0.4537** | 0.3208 | **0.5705** | 0.3587 | 0.6598 | 0.3812 |
| | N(a) | 0.4532 | **0.3238** | 0.5701 | **0.3620** | **0.6600** | **0.3849** |
| | U(s) | **0.4364** | **0.3122** | **0.5470** | **0.3482** | **0.6346** | **0.3705** |
| | U(a) | 0.4101 | 0.2934 | 0.5129 | 0.3268 | 0.5959 | 0.3478 |
| | M(s) | **0.4550** | **0.3243** | **0.5701** | **0.3617** | **0.6535** | **0.3828** |
| | M(a) | 0.4507 | 0.3190 | 0.5611 | 0.3549 | 0.6500 | 0.3775 |
| | D(s) | **0.4509** | **0.3184** | **0.5669** | **0.3561** | **0.6574** | **0.3792** |
| | D(a) | 0.4461 | 0.3167 | 0.5558 | 0.3525 | 0.6438 | 0.3749 |

## 6 CONCLUSION AND FUTURE WORK

This paper proposed contrastive state augmentation (CSA) for training RL-based recommender systems to address the issues of inaccurate value estimation for unseen states and insufficient state representation learning from implicit user feedback. We devised four state augmentation strategies to improve the generalization capability of the RL recommendation agent. In addition, we introduced contrastive signals to facilitate state representation learning. To verify the effectiveness of CSA, we implemented it with three state-of-the-art sequential recommendation models. We conducted extensive experiments on three real-world datasets and one simulated online environment. Experimental results demonstrate that the proposed CSA training framework can effectively improve recommendation performance. In the future, we intend to investigate more state augmentation strategies and design more effective contrastive loss to improve the performance further. We believe that combining self-supervised representation learning and RL could be a promising direction for future recommender systems.

# REFERENCES

[1] Samuel R Bowman, Luke Vilnis, Oriol Vinyals, Andrew M Dai, Rafal Jozefowicz, and Samy Bengio. 2015. Generating sentences from a continuous space. *arXiv preprint arXiv:1511.06349* (2015).

[2] Minmin Chen, Alex Beutel, Paul Covington, Sagar Jain, Francois Belletti, and Ed H Chi. 2019. Top-k off-policy correction for a REINFORCE recommender system. In *WSDM*. 456–464.

[3] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. 2020. A simple framework for contrastive learning of visual representations. In *ICML*. 1597–1607.

[4] Xinshi Chen, Shuang Li, Hui Li, Shaohua Jiang, Yuan Qi, and Le Song. 2019. Generative adversarial user model for reinforcement learning based recommendation system. In *ICML*. 1052–1061.

[5] Zhiyong Cheng, Sai Han, Fan Liu, Lei Zhu, Zan Gao, and Yuxin Peng. 2023. Multi-Behavior Recommendation with Cascading Graph Convolution Networks. In *WWW*. 1–10.

[6] Andrew M Dai and Quoc V Le. 2015. Semi-supervised sequence learning. *NIPS* (2015).

[7] Scott Fujimoto, David Meger, and Doina Precup. 2019. Off-policy deep reinforcement learning without exploration. In *ICML*. 2052–2062.

[8] Chengqian Gao, Ke Xu, Kuangqi Zhou, Lanqing Li, Xueqian Wang, Bo Yuan, and Peilin Zhao. 2022. Value penalized q-learning for recommender systems. In *SIGIR*. 2008–2012.

[9] Tianyu Gao, Xingcheng Yao, and Danqi Chen. 2021. Simcse: Simple contrastive learning of sentence embeddings. *arXiv preprint arXiv:2104.08821* (2021).

[10] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. 2018. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905* (2018).

[11] Hado Hasselt. 2010. Double Q-learning. *NIPS* (2010).

[12] Ruining He, Chen Fang, Zhaowen Wang, and Julian McAuley. 2016. Vista: A visually, socially, and temporally-aware model for artistic recommendation. In *RecSys*. 309–316.

[13] Ruining He and Julian McAuley. 2016. Fusing similarity models with markov chains for sparse sequential recommendation. In *ICDM*. 191–200.

[14] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. 2015. Session-based recommendations with recurrent neural networks. *arXiv preprint arXiv:1511.06939* (2015).

[15] Balázs Hidasi, Massimo Quadrana, Alexandros Karatzoglou, and Domonkos Tikk. 2016. Parallel recurrent neural network architectures for feature-rich session-based recommendations. In *RecSys*. 241–248.

[16] Jin Huang, Zhaochun Ren, Wayne Xin Zhao, Gaole He, Ji-Rong Wen, and Daxiang Dong. 2019. Taxonomy-aware multi-hop reasoning networks for sequential recommendation. In *WSDM*. 573–581.

[17] Jin Huang, Wayne Xin Zhao, Hongjian Dou, Ji-Rong Wen, and Edward Y Chang. 2018. Improving sequential recommendation with knowledge-enhanced memory networks. In *SIGIR*. 505–514.

[18] Mohsen Jamali and Martin Ester. 2010. A matrix factorization technique with trust propagation for recommendation in social networks. In *RecSys*. 135–142.

[19] Wang-Cheng Kang and Julian McAuley. 2018. Self-attentive sequential recommendation. In *ICDM*. 197–206.

[20] Vijay Konda and John Tsitsiklis. 1999. Actor-critic algorithms. *NIPS* (1999).

[21] Aviral Kumar, Justin Fu, Matthew Soh, George Tucker, and Sergey Levine. 2019. Stabilizing off-policy q-learning via bootstrapping error reduction. *NIPS* (2019).

[22] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. 2020. Conservative q-learning for offline reinforcement learning. *NIPS* (2020), 1179–1191.

[23] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. 2020. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643* (2020).

[24] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *nature* 518, 7540 (2015), 529–533.

[25] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. 2018. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748* (2018).

[26] Ruihong Qiu, Zi Huang, Hongzhi Yin, and Zijian Wang. 2022. Contrastive learning for representation degeneration problem in sequential recommendation. In *WSDM*. 813–823.

[27] Massimo Quadrana, Alexandros Karatzoglou, Balázs Hidasi, and Paolo Cremonesi. 2017. Personalizing session-based recommendations with hierarchical recurrent neural networks. In *RecSys*. 130–137.

[28] Pengjie Ren, Zhumin Chen, Jing Li, Zhaochun Ren, Jun Ma, and Maarten De Rijke. 2019. Repeatnet: A repeat aware neural recommendation machine for session-based recommendation. In *AAAI*. 4806–4813.

[29] Steffen Rendle. 2010. Factorization machines. In *ICDM*. 995–1000.

[30] Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. 2010. Factorizing personalized markov chains for next-basket recommendation. In *WWW*.

[31] J Ben Schafer, Joseph A Konstan, and John Riedl. 2001. E-commerce recommendation applications. *Data mining and knowledge discovery* 5, 1 (2001), 115–153.

[32] G Shubhashri, N Unnamalai, and G Kamalika. 2018. LAWBO: a smart lawyer chatbot.. In *COMAD/CODS*. 348–351.

[33] 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.

[34] Fei Sun, Jun Liu, Jian Wu, Changhua Pei, Xiao Lin, Wenwu Ou, and Peng Jiang. 2019. BERT4Rec: Sequential recommendation with bidirectional encoder representations from transformer. In *CIKM*. 1441–1450.

[35] Richard S Sutton, Andrew G Barto, et al. 1998. *Introduction to reinforcement learning*. Vol. 135. MIT press Cambridge.

[36] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. 1999. Policy gradient methods for reinforcement learning with function approximation. *NIPS* (1999).

[37] Jiaxi Tang and Ke Wang. 2018. Personalized top-n sequential recommendation via convolutional sequence embedding. In *WSDM*. 565–573.

[38] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *NIPS* (2017).

[39] Jizhe Wang, Pipei Huang, Huan Zhao, Zhibo Zhang, Binqiang Zhao, and Dik Lun Lee. 2018. Billion-scale commodity embedding for e-commerce recommendation in alibaba. In *SIGKDD*. 839–848.

[40] Pengfei Wang, Yu Fan, Long Xia, Wayne Xin Zhao, ShaoZhang Niu, and Jimmy Huang. 2020. KERL: A knowledge-guided reinforcement learning model for sequential recommendation. In *SIGIR*. 209–218.

[41] Christopher John Cornish Hellaby Watkins. 1989. Learning from delayed rewards. (1989).

[42] Jiancan Wu, Xiang Wang, Fuli Feng, Xiangnan He, Liang Chen, Jianxun Lian, and Xing Xie. 2021. Self-supervised graph learning for recommendation. In *SIGIR*. 726–735.

[43] Zhuofeng Wu, Sinong Wang, Jiatao Gu, Madian Khabsa, Fei Sun, and Hao Ma. 2020. Clear: Contrastive learning for sentence representation. *arXiv preprint arXiv:2012.15466* (2020).

[44] Xin Xia, Hongzhi Yin, Junliang Yu, Qinyong Wang, Lizhen Cui, and Xiangliang Zhang. 2021. Self-supervised hypergraph convolutional networks for session-based recommendation. In *AAAI*. 4503–4511.

[45] Yikun Xian, Zuohui Fu, Shan Muthukrishnan, Gerard De Melo, and Yongfeng Zhang. 2019. Reinforcement knowledge graph reasoning for explainable recommendation. In *SIGIR*. 285–294.

[46] Bo Xiao and Izak Benbasat. 2007. E-commerce product recommendation agents: Use, characteristics, and impact. *MIS quarterly* (2007), 137–209.

[47] Teng Xiao and Donglin Wang. 2021. A general offline reinforcement learning framework for interactive recommendation. In *AAAI*. 4512–4520.

[48] Xu Xie, Fei Sun, Zhaoyang Liu, Shiwen Wu, Jinyang Gao, Jiandong Zhang, Bolin Ding, and Bin Cui. 2022. Contrastive learning for sequential recommendation. In *ICDE*. 1259–1273.

[49] Xin Xin, Alexandros Karatzoglou, Ioannis Arapakis, and Joemon M Jose. 2020. Self-supervised reinforcement learning for recommender systems. In *SIGIR*. 931–940.

[50] An Yan, Shuo Cheng, Wang-Cheng Kang, Mengting Wan, and Julian McAuley. 2019. CosRec: 2D convolutional neural networks for sequential recommendation. In *CIKM*. 2173–2176.

[51] Fajie Yuan, Alexandros Karatzoglou, Ioannis Arapakis, Joemon M Jose, and Xiangnan He. 2019. A simple convolutional generative network for next item recommendation. In *WSDM*. 582–590.

[52] Ruiyi Zhang, Tong Yu, Yilin Shen, and Hongxia Jin. 2022. Text-Based Interactive Recommendation via Offline Reinforcement Learning. In *AAAI*. 11694–11702.

[53] Xiangyu Zhao, Long Xia, Liang Zhang, Zhuoye Ding, Dawei Yin, and Jiliang Tang. 2018. Deep reinforcement learning for page-wise recommendations. In *RecSys*. 95–103.

[54] Xiangyu Zhao, Liang Zhang, Zhuoye Ding, Long Xia, Jiliang Tang, and Dawei Yin. 2018. Recommendations with negative feedback via pairwise deep reinforcement learning. In *SIGKDD*. 1040–1048.

[55] Kun Zhou, Hui Wang, Wayne Xin Zhao, Yutao Zhu, Sirui Wang, Fuzheng Zhang, Zhongyuan Wang, and Ji-Rong Wen. 2020. S3-rec: Self-supervised learning for sequential recommendation with mutual information maximization. In *CIKM*. 1893–1902.

[56] Kun Zhou, Hui Yu, Wayne Xin Zhao, and Ji-Rong Wen. 2022. Filter-enhanced MLP is all you need for sequential recommendation. In *WWW*. 2388–2399.

[57] Lixin Zou, Long Xia, Zhuoye Ding, Jiaxing Song, Weidong Liu, and Dawei Yin. 2019. Reinforcement learning to optimize long-term user engagement in recommender systems. In *SIGKDD*. 2810–2818.

[58] Lixin Zou, Long Xia, Pan Du, Zhuo Zhang, Ting Bai, Weidong Liu, Jian-Yun Nie, and Dawei Yin. 2020. Pseudo Dyna-Q: A reinforcement learning framework for interactive recommendation. In *WSDM*. 816–824.