Self-Supervised Reinforcement Learning for Recommender Systems

Xin Xin*
University of Glasgow
x.xin.1@research.gla.ac.uk

Alexandros Karatzoglou
Google, London
alexandros.karatzoglou@gmail.com

Ioannis Arapakis
Telefonica Research, Barcelona
arapakis.ioannis@gmail.com

Joemon M. Jose
University of Glasgow
Joemon.Jose@glasgow.ac.uk

ABSTRACT

In session-based or sequential recommendation, it is important to consider a number of factors like long-term user engagement, multiple types of user-item interactions such as clicks, purchases etc. The current state-of-the-art supervised approaches fail to model them appropriately. Casting sequential recommendation task as a reinforcement learning (RL) problem is a promising direction. A major component of RL approaches is to train the agent through interactions with the environment. However, it is often problematic to train a recommender in an on-line fashion due to the requirement to expose users to irrelevant recommendations. As a result, learning the policy from logged implicit feedback is of vital importance, which is challenging due to the *pure off-policy setting* and *lack of negative rewards (feedback)*.

In this paper, we propose *self-supervised reinforcement learning* for sequential recommendation tasks. Our approach augments standard recommendation models with two output layers: one for self-supervised learning and the other for RL. The RL part acts as a regularizer to drive the supervised layer focusing on specific rewards (e.g., recommending items which may lead to purchases rather than clicks) while the self-supervised layer with cross-entropy loss provides strong gradient signals for parameter updates. Based on such an approach, we propose two frameworks namely *Self-Supervised Q-learning* (SQN) and *Self-Supervised Actor-Critic* (SAC). We integrate the proposed frameworks with four state-of-the-art recommendation models. Experimental results on two real-world datasets demonstrate the effectiveness of our approach.

CCS CONCEPTS

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

KEYWORDS

Session-based Recommendation; Sequential Recommendation; Reinforcement Learning; Self-supervised Learning; Q-learning

ACM Reference Format:
Xin Xin, Alexandros Karatzoglou, Ioannis Arapakis, and Joemon M. Jose. 2020. Self-Supervised Reinforcement Learning for Recommender Systems. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '20), July 25–30, 2020, Virtual Event, China.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3397271.3401147

1 INTRODUCTION

Generating next item recommendation from sequential user-item interactions in a session (e.g., views, clicks or purchases) is one of the most common use cases in domains of recommender systems, such as e-commerce, video [18] and music recommendation [42]. Session-based and sequential recommendation models have often been trained with *self-supervised learning*, in which the model is trained to predict the data itself instead of some "external" labels. For instance, in language modeling the task is often formulated as predicting the next word given the previous m words. The same procedure can be utilized to predict the next item the user may be interested given past interactions, see e.g., [14, 21, 42]. However, this kind of approaches can lead to sub-optimal recommendations since the model is purely learned by a loss function defined on the discrepancy between model predictions and the self-supervision signal. Such a loss may not match the expectations from the perspective of recommendation systems (e.g., long-term engagement). Moreover, there can be multiple types of user signals in one session, such as clicks, purchases etc. How to leverage multiple types of user-item interactions to improve recommendation objectives (e.g., providing users recommendations that lead to real purchases) is also an important research question.

Reinforcement Learning (RL) has achieved impressive advances in game control [27, 37] and related fields. A RL agent is trained to take actions given the state of the environment it operates in with the objective of getting the maximum long-term cumulative rewards. A recommender system aims (or should aim) to provide recommendations (actions) to users (environment) with the objective of maximising the long-term user satisfaction (reward) with the system. Since in principle the reward schema can be designed at will, RL allows to create models that can serve multiple objectives such as diversity and novelty. As a result, exploiting RL for recommendation has become a promising research direction. There are two

*Part of this work is done when taking an internship in Telefonica Research, Barcelona.

SIGIR '20, July 25–30, 2020, Virtual Event, China

ACM ISBN 978-1-4503-8016-4/20/07…$15.00
https://doi.org/10.1145/3397271.3401147

classes of RL methods: model-free RL algorithms and model-based RL algorithms.

Model-free RL algorithms need to interact with the environment to observe the feedback and then optimize the policy. Doing this in an on-line fashion is typically unfeasible in commercial recommender systems since interactions with an under-trained policy would affect the user experience. A user may quickly abandon the service if the recommendations don't match her interests. A typical solution is learning off-policy from the logged implicit feedback. However, this poses the following challenges for applying RL-based methods:

(1) Pure off-policy settings. The policy should be learned from fixed logged data without interactions with the environment (users). Hence the data from which the RL agent is trained (i.e., logged data) will not match its policy. [3] proposed to use propensity scores to perform off-policy correction but this kind of methods can suffer from unbounded high variances [28].
(2) Lack of data and negative rewards. RL algorithms are data hungry, traditional techniques overcome this by either simulating the environments or by running RL iterations in controlled environments (e.g. games, robots). This is challenging in the case of recommendations especially considering the large number of potential actions (available items). Moreover, in most cases learning happens from implicit feedback. The agent only knows which items the user has interacted with but has no knowledge about what the user dislikes. In other words, simply regressing to the Bellman equation [1] (i.e., Q-learning) wouldn't lead to good ranking performance when there is no negative feedback since the model will be biased towards positive relevance values.

An alternative to off-policy training for recommender systems is the use of model-based RL algorithms. In model-based RL, one first builds a model to simulate the environment. Then the agent can learn by interactions with the simulated environment [4, 36]. These two-stage methods heavily depend on the constructed simulator. Although related methods like generative adversarial networks (GANs) [9] achieve good performance when generating images, simulating users' responses is a much more complex task.

In this paper, we propose *self-supervised reinforcement learning* for recommender systems. The proposed approach serves as a learning mechanism and can be easily integrated with existing recommendation models. More precisely, given a sequential or session-based recommendation model, the (final) hidden state of this model can be seen as it's output as this hidden state is multiplied with the last layer to generate recommendations [14, 21, 38, 42]. We augment these models with two final output layers (heads). One is the conventional self-supervised head[1] trained with cross-entropy loss to perform ranking while the second is trained with RL based on the defined rewards such as long-term user engagement, purchases, recommendation diversity and so on. For the training of the RL head, we adopt double Q-learning which is more stable and robust in the off-policy setting [10]. The two heads complement each other during the learning process. The RL head serves as a regularizer to introduce desired properties to the recommendations while the ranking-based supervised head can provide negative signals for parameter updates. We propose two frameworks based on such an approach: (1) *Self-Supervised Q-learning* (SQN) co-trains the two layers with a reply buffer generated from the logged implicit feedback; (2) *Self-Supervised Actor-Critic* (SAC) treats the RL head as a critic measuring the value of actions in a given state while the supervised head as an actor to perform the final ranking among candidate items. As a result, the model focuses on the pre-defined rewards while maintaining high ranking performance. We verify the effectiveness of our approach by integrating SQN and SAC on four state-of-the-art sequential or session-based recommendation models.

To summarize, this work makes the following contributions:

- We propose self-supervised reinforcement learning for sequential recommendation. Our approach extends existing recommendation models with a RL layer which aims to introduce reward driven properties to the recommendation.
- We propose two frameworks SQN and SAC to co-train the supervised head and the RL head. We integrate four state-of-the-art recommendation models into the proposed frameworks.
- We conduct experiments on two real world e-commerce datasets with both clicks and purchases interactions to validate our proposal. Experimental results demonstrate the proposed methods are effective to improve hit ratios, especially when measured against recommending items that eventually got purchased.

2 PRELIMINARIES

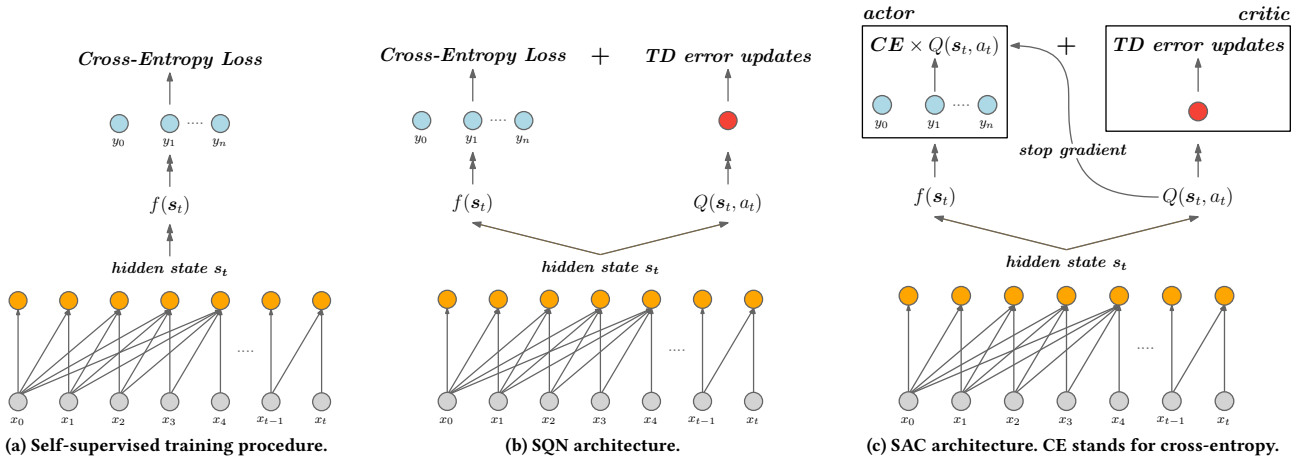
In this section, we first describe the basic problem of generating next item recommendations from sequential or session-based user data. We introduce reinforcement learning and analyse its limitations for recommendation. Lastly, we provide a literature review on related work.

2.1 Next Item Recommendation

Let $\mathcal{I}$ denote the whole item set, then a user-item interaction sequence can be represented as $x_{1:t} = \{x_1, x_2, ..., x_{t-1}, x_t\}$, where $x_i \in \mathcal{I} (0 < i \leq t)$ is the index of the interacted item at timestamp i. Note that in a real world scenario there may be different kinds of interactions. For instance, in e-commerce use cases, the interactions can be clicks, purchases, add to basket and so on. In video recommendation systems, the interactions can be characterized by the watching time of a video. The goal of next item recommendation is recommending the most relevant item x_{t+1} to the user given the sequence of previous interactions $x_{1:t}$.

We can cast this task as a (self-supervised) multi-class classification problem and build a sequential model that generates the classification logits $\mathbf{y}_{t+1} = [y_1, y_2, ...y_n] \in \mathbb{R}^n$, where n is the number of candidate items. We can then choose the top-k items from $\mathbf{y}_{t+1}$ as our recommendation list for timestamp $t + 1$. A common procedure to train this type of recommender is shown in Figure 1a. Typically one can use a generative model G to map the input sequence into a hidden state $\mathbf{s}_t$ as $\mathbf{s}_t = G(x_{1:t})$. This serves as a general encoder function. Plenty of models have been proposed for this task and we will discuss prominent ones in section 2.3.

[1]For simplicity, we use "self-supervised" and "supervised" interchangeable in this paper. Besides, "head" and "layer" are also interchangeable.



(a) Self-supervised training procedure. (b) SQN architecture. (c) SAC architecture. CE stands for cross-entropy.

Figure 1: The self-supervised learning procedure for recommendation and the proposed frameworks.

Based on the obtained hidden state, one can utilize a decoder to map the hidden state to the classification logits as $\mathbf{y}_{t+1} = f(\mathbf{s}_t)$. It is usually defined as a simple fully connected layer or the inner product with candidate item embeddings [14, 21, 38, 42]. Finally, we can train our recommendation model (agents) by optimizing a loss function based on the logits $\mathbf{y}_{t+1}$, such as the cross-entropy loss or the pair-wise ranking loss [31].

2.2 Reinforcement Learning

In terms of RL, we can formulate the next item recommendation problem as a Markov Decision Process (MDP) [35], in which the recommendation agent interacts with the environments $\mathcal{E}$ (users) by sequentially recommending items to maximize the long-term cumulative rewards. More precisely, the MDP can be defined by tuples consisting of $(\mathcal{S}, \mathcal{A}, \mathbf{P}, R, \rho_0, \gamma)$ where

- $\mathcal{S}$: a continuous state space to describe the user states. This is modeled based on the user (sequential) interactions with the items. The state of the user can be in fact represented by the hidden state of the sequential model discussed in section 2.1. Hence the state of a user at timestamp t can be represented as $\mathbf{s}_t = G(x_{1:t}) \in \mathcal{S}$ $(t > 0)$.
- $\mathcal{A}$: a discrete action space which contains candidate items. The action a of the agent is the selected item to be recommended. In off-line data, we can get the action at timestamp t from the user-item interaction (i.e., $a_t = x_{t+1}$). There are also works that focus on generating slate (set)-based recommendations and we will discuss them in section 2.3.
- $\mathbf{P}$: $\mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the state transition probability.
- R: $\mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, where $r(\mathbf{s}, a)$ denotes the immediate reward by taking action a at user state $\mathbf{s}$. The flexible reward scheme is crucial in the utility of RL for recommender systems as it allows for optimizing the recommendation models towards goals that are not captured by conventional loss functions. For example, in the e-commerce scenario, we can give a larger reward to purchase interactions compared with clicks to build a model that assists the user in his purchase rather than the browsing task. We can also set the reward according to item novelty [2] to promote recommendation diversity. For video recommendation, we can set the rewards according to the watching time [3].
- ρ_0 is the initial state distribution with $\mathbf{s}_0 \sim \rho_0$.
- γ is the discount factor for future rewards.

RL seeks a target policy $\pi_\theta(a|\mathbf{s})$ which translates the user state $\mathbf{s} \in \mathcal{S}$ into a distribution over actions $a \in \mathcal{A}$, so as to maximize the expected cumulative reward:

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

where $\theta \in \mathbb{R}^d$ denotes policy parameters. Note that the expectation is taken over trajectories $\tau = (\mathbf{s}_0, a_0, \mathbf{s}_1, \ldots)$, which are obtained by performing actions according to the target policy: $\mathbf{s}_0 \sim \rho_0$, $a_t \sim \pi_\theta(\cdot|\mathbf{s}_t)$, $s_{t+1} \sim \mathbf{P}(\cdot|\mathbf{s}_t, a_t)$.

Solutions to find the optimal θ can be categorized into policy gradient-based approaches (e.g., REINFORCE [41]) and value-based approaches (e.g., Q-learning [37]).

Policy-gradient based approaches aim at directly learning the mapping function π_θ. Using the "log-trick" [41], gradients of the expected cumulative rewards with respect to policy parameters θ can be derived as:

$$\mathbb{E}_{\tau \sim \pi_\theta}[R(\tau) \nabla_\theta \log \pi_\theta(\tau)]. \tag{2}$$

In on-line RL environments, it's easy to estimate the expectation. However, under the recommendation settings, to avoid recommending irrelevant items to users, the agent must be trained using the historical logged data. Even if the RL agent can interact with live users, the actions (recommended items) may be controlled by other recommenders with different policies since many recommendation models might be deployed in a real live recommender system. As a result, what we can estimate from the batched (logged) data is

$$\mathbb{E}_{\tau \sim \beta}[R(\tau) \nabla_\theta \log \pi_\theta(\tau)], \tag{3}$$

where β denotes the behavior policy that we follow when generating the training data. Obviously, there is distribution discrepancy

$$x_{1:t} \{\overset{click}{x_1}, \overset{purchase}{x_2}, \ldots, \overset{click}{x_{t-1}}, \overset{click}{x_t}\}$$

$Q(s_0, x_1) =$ reward of click $+ \max_a Q(s_1, a)$

$Q(s_1, x_2) =$ reward of purchase $+ \max_a Q(s_2, a)$

$Q(s_0, x_1^-) = ?$ $Q(s_1, x_2^-) = ?$ ** *no learning constraints* **

$argmax Q(s, a) = ?$ ** *fails to perform ranking* **

Figure 2: Q-learning fails to learn a proper preference ranking because of data sparsity and the lack of negative feedback. x_1^- and x_2^- are unseen (negative) items for the corresponding timestamp.

between the target policy π_θ and the behavior policy β. Applying policy-gradient methods for recommendation using this data is thus infeasible.

Value-based approaches first calculate the Q-values (i.e., $Q(\mathbf{s}, a)$, the value of an action a at a given state $\mathbf{s}$) according to the Bellman equation while the action is taken by $a = \text{argmax}\, Q(\mathbf{s}, a)$. The one-step temporal difference (TD) update rule formulates the target $Q(\mathbf{s}_t, a_t)$ as

$$Q(\mathbf{s}_t, a_t) = r(\mathbf{s}_t, a_t) + \gamma \max_{a'} Q(\mathbf{s}_{t+1}, a'). \tag{4}$$

One of the major limitation of implicit feedback data is the lack of negative feedback [31, 43], which means we only know which items the user has interacted with but have no knowledge about the missing transactions. Thus there are limited state-action pairs to learn from and Q-values learned purely on this data would be sub-optimal as shown in Figure 2. As a result, taking actions using these Q-values by $a = \text{argmax}\, Q(\mathbf{s}, a)$ would result in poor performance. Even though the estimation of $Q(\mathbf{s}, a)$ is unbiased due to the greedy selection of Q-learning[2], the distribution of $\mathbf{s}$ in the logged data is biased. So the distribution discrepancy problem of policy gradient-based methods still exists in Q-learning even though the Q-learning algorithm is "off-policy" [7].

2.3 Related Work

Early work focusing on sequential recommendation mainly rely on Markov Chain (MC) models [5, 12, 32] and factorization-based methods [15, 30]. Rendle et. al [32] introduced to use first-order MC to capture short-term user preferences and combined the MC with matrix factorization (MF) [24] to model long-term preferences. Methods with high-order MCs that consider more longer interaction sequences were also proposed in [11, 12]. Factorization-based methods such as factorization machines (FM) [30] can utilize the previous items a user has interacted with as context features. The general factorization framework (GFF) [15] models a session as the average of the items that the user interacted within that session.

MC-based methods face challenges in modeling complex sequential signals such as skip behaviors in the user-item sequences [38, 42] while factorization-based methods do not model the order of user-item interactions. As a result, plenty of deep learning-based approaches have been proposed to model the interaction sequences more effectively. [14] proposed to utilize gated recurrent units (GRU) [6] to model the session. [38] and [42] utilized convolutional neural networks (CNN) to capture sequential signals. [21] exploited the well-known Transformer [40] in the field of sequential recommendation with promising results. Generally speaking, all of these models can serve as the model G described in section 2.1 whose input is a sequence of user-item interactions while the output is a latent representation $\mathbf{s}$ that describes the corresponding user state.

Attempts to utilize RL for recommendation have also been made. To address the problem of distribution discrepancy under the off-policy settings, [3] proposed to utilize propensity scores to perform off-policy correction. However, the estimation of propensity scores has high variances and there is a trade-off between bias and variance, which introduces additional training difficulties. [44] proposed to utilize negative sampling along with Q-learning. But their method doesn't address the off-policy problem. Model-based RL approaches [4, 34, 45] firstly build a model to simulate the environment in order to avoid any issues with off-policy training. However, these two-stage approaches heavily depend on the accuracy of the simulator. Moreover, recent work has also been done on providing slate-based recommendations [3, 4, 8, 19] in which actions are considered to be sets (slates) of items to be recommended. This assumption creates an even larger action space as a slate of items is regarded as one single action. To keep in line with existing self-supervised recommendation models, in this paper we set the action to be recommending the top-k items that are scored by the supervised head. We leave research in this area of set-based recommendation as one of our future work directions.

Bandit algorithms which share the same reward schema and long-term expectation with RL have also been investigated for recommendation [25, 26]. Bandit algorithms assume that taking actions does not affect the state [25] while in full RL the assumption is that the state is affected by the actions. Generally speaking, recommendations actually have an effect on user behavior [33] and hence RL is more suitable for modeling the recommendation task. Another related field is imitation learning where the policy is learned from expert demonstrations [16, 17, 39]. Our work can be also considered as a form of imitation learning as part of the model is trained to imitate user actions.

3 METHOD

As discussed in section 2.2, directly applying standard RL algorithms to recommender systems data is essentially unfeasible. In this section, we propose to co-train a RL output layer along with the self-supervised head. The reward can be designed according to the specific demands of the recommendation setting. We first describe the proposed SQN algorithm and then extend it to SAC. Both algorithms can be easily integrated with existing recommendation models.

3.1 Self-Supervised Q-learning

Given an input item sequence $x_{1:t}$ and an existing recommendation model G, the self-supervised training loss can be defined as the cross-entropy over the classification distribution:

$$L_s = -\sum_{i=1}^{n} Y_i log(p_i), \text{where } p_i = \frac{e^{y_i}}{\sum_{i'=1}^{n} e^{y_{i'}}}. \tag{5}$$

[2]We don't consider the bias introduced by the steps of TD learning. This is not related to our work.

Y_i is an indicator function and $Y_i = 1$ if the user interacted with the i-th item in the next timestamp. Otherwise, $Y_i = 0$. Due to the fact that the recommendation model G has already encoded the input sequence into a latent representation $\mathbf{s}_t$, we can directly utilize $\mathbf{s}_t$ as the state for the RL part without introducing another model. What we need is an additional final layer to calculate the Q-values. A concise solution is stacking a fully-connected layer on the top of G:

$$Q(\mathbf{s}_t, a_t) = \delta(\mathbf{s}_t \mathbf{h}_t^T + b) = \delta(G(x_{1:t})\mathbf{h}_t^T + b), \tag{6}$$

where δ denotes the activation function, $\mathbf{h}_t$ and b are trainable parameters of the RL head.

After that, we can define the loss for the RL part based on the one-step TD error:

$$L_q = (r(\mathbf{s}_t, a_t) + \gamma \max_{a'} Q(\mathbf{s}_{t+1}, a') - Q(\mathbf{s}_t, a_t))^2 \tag{7}$$

We jointly train the self-supervised loss and the RL loss on the replay buffer generated from the implicit feedback data:

$$L_{SQN} = L_s + L_q. \tag{8}$$

Figure 1b demonstrates the architecture of SQN.

When generating recommendations, we still return the top-k items from the supervised head. The RL head acts as a regularizer to fine-tune the recommendation model G according to our reward settings. As discussed in section 2.2, the state distribution in the logged data is biased, so generating recommendations using the Q-values is problematic. However, due to the greedy selection of $Q(\mathbf{s}_{t+1}, \cdot)$, the estimation of $Q(\mathbf{s}_t, a_t)$ itself is unbiased. As a result, by utilizing Q-learning as a regularizer and keeping the self-supervised layer as the source of recommendations we avoid any off-policy correction issues. The lack of negative rewards in Q-learning does also not affect the methods since the RL output layer is trained on positive actions and the supervised cross-entropy loss provides the negative gradient signals which come from the denominator of Eq.(5).

To enhance the learning stability, we utilize double Q-learning [10] to alternatively train two copies of learnable paramaters. Algorithm 1 describes the training procedure of SQN.

3.2 Self-Supervised Actor-Critic

In the previous subsection, we proposed SQN in which the Q-learning head acts as a regularizer to fine-tune the model in line with the reward schema. The learned Q-values are unbiased and learned from positive user-item interactions (feedback). As a result, these values can be regarded as an unbiased measurement of how the recommended item satisfies our defined rewards. Hence the actions with high Q-values should get increased weight on the self-supervised loss, and vice versa.

We can thus treat the self-supervised head which is used for generating recommendations as a type of "actor" and the Q-learning head as the "critic". Based on this observation, we can use the Q-values as weights for the self-supervised loss:

$$L_A = L_s \cdot Q(\mathbf{s}_t, a_t). \tag{9}$$

This is similar to what is used in the well-known Actor-Critic (AC) methods [23]. However, the actor in AC is based on policy gradient which is on-policy while the "actor" in our methods is essentially self-supervised. Moreover, there is only one base model G in SAC while AC has two separate networks for the actor and the critic. To enhance stability, we stop the gradient flow and fix the Q-values when they are used in that case. We then train the actor and critic jointly. Figure 1c illustrates the architecture of SAC. In complex sequential recommendation models (e.g., using the Transformer encoder as G [21]), the learning of Q-values can be unstable [29], particularly in the early stages of training. To mitigate these issues, we set a threshold T. When the number of update steps is smaller than T, we perform normal SQN updates. After that, the Q-values become more stable and we start to use the critic values in the self-supervised layer and perform updates according to the architecture of Figure 1c. The training procedure of SAC is demonstrated in Algorithm 2.

3.3 Discussion

The proposed training frameworks can be integrated with existing recommendation models, as long as they follow the procedure of Figure 1a to generate recommendations. This is the case for most session-based or sequential recommendation models introduced over the last years. In this paper we utilize the cross-entropy loss as the self-supervised loss, while the proposed methods also work for other losses [13, 31]. The proposed methods are for general purpose recommendation. You can design the reward schema according to your own demands and recommendation objectives.

Due to the biased state-action distribution in the off-line setting and the lack of sufficient data, directly generating recommendations from RL is difficult. The proposed SQN and SAC frameworks can be seen as attempts to exploit an unbiased RL estimator[3] to "reinforce" existing self-supervised recommendation models. Another way of looking at the proposed approach is as a form of transfer learning

[3] In our case, the estimation of $Q(\mathbf{s}, a)$ is unbiased.

Algorithm 1 Training procedure of SQN

Input: user-item interaction sequence set $\mathcal{X}$, recommendation model G, reinforcement head Q, supervised head
Output: all parameters in the learning space Θ

Initialize all trainable parameters
Create G' and Q' as copies of G and Q, respectively
repeat
  Draw a mini-batch of $(x_{1:t}, a_t)$ from $\mathcal{X}$, set rewards r
  $\mathbf{s}_t = G(x_{1:t})$, $\mathbf{s}'_t = G'(x_{1:t})$
  $\mathbf{s}_{t+1} = G(x_{1:t+1})$, $\mathbf{s}'_{t+1} = G'(x_{1:t+1})$
  Generate random variable $z \in (0, 1)$ uniformly
  if $z \leq 0.5$ **then**
    $a^* = \text{argmax}_a\, Q(\mathbf{s}_{t+1}, a)$
    $L_q = (r + \gamma Q'(\mathbf{s}'_{t+1}, a^*) - Q(\mathbf{s}_t, a_t))^2$
    Calculate L_s and L_{SQN} according to Eq.(5) and Eq.(8)
    Perform updates by $\nabla_\Theta L_{SQN}$
  else
    $a^* = \text{argmax}_a\, Q'(\mathbf{s}'_{t+1}, a)$
    $L_q = (r + \gamma Q(\mathbf{s}_{t+1}, a^*) - Q'(\mathbf{s}'_t, a_t))^2$
    Calculate L_s and L_{SQN} according to Eq.(5) and Eq.(8)
    Perform updates by $\nabla_\Theta L_{SQN}$
  end if
until converge
return all parameters in Θ

Algorithm 2 Training procedure of SAC

Input: the interaction sequence set $\mathcal{X}$, recommendation model G, reinforcement head Q, supervised head, threshold T
Output: all parameters in the learning space Θ

Initialize all trainable parameters
Create G' and Q' as copies of G and Q, $t = 0$
repeat
  Draw a mini-batch of $(x_{1:t}, a_t)$ from $\mathcal{X}$, set rewards r
  $\mathbf{s}_t = G(x_{1:t})$, $\mathbf{s}'_t = G'(x_{1:t})$
  $\mathbf{s}_{t+1} = G(x_{1:t+1})$, $\mathbf{s}'_{t+1} = G'(x_{1:t+1})$
  Generate random variable $z \in (0, 1)$ uniformly
  if $z \leq 0.5$ **then**
    $a^* = \text{argmax}_a\ Q(\mathbf{s}_{t+1}, a)$
    $L_q = (r + \gamma Q'(\mathbf{s}'_{t+1}, a^*) - Q(\mathbf{s}_t, a_t))^2$
    if $t \leq T$ **then**
      Perform updates by $\nabla_\Theta L_{SQN}$
    else
      $L_A = L_s \times Q(\mathbf{s}_t, a_t)$, $L_{SAC} = L_A + L_s$
      Perform updates by $\nabla_\Theta L_{SAC}$
    end if
  else
    $a^* = \text{argmax}_a\ Q'(\mathbf{s}'_{t+1}, a)$
    $L_q = (r + \gamma Q(\mathbf{s}_{t+1}, a^*) - Q'(\mathbf{s}'_t, a_t))^2$
    if $t \leq T$ **then**
      Perform updates by $\nabla_\Theta L_{SQN}$
    else
      $L_A = L_s \times Q'(\mathbf{s}'_t, a_t)$, $L_{SAC} = L_A + L_s$
      Perform updates by $\nabla_\Theta L_{SAC}$
    end if
  end if
  $t = t + 1$
until converge
return all parameters in Θ

whereby the self-supervised model is used to "pretrain" parts of the Q-learning model and vice versa.

4 EXPERIMENTS

In this section, we conduct experiments[4] on two real-world sequential recommendation datasets to evaluate the proposed SQN and SAC in the e-commerce scenario. We aim to answer the following research questions:

RQ1: How do the proposed methods perform when integrated with existing recommendation models?

RQ2: How does the RL component affect performance, including the reward setting and the discount factor.

RQ3: What is the performance if we only use Q-leaning for recommendation?

In the following parts, we will describe the experimental settings and answer the above research questions.

[4]The implementation can be found at https://drive.google.com/open?id=1nLL3_knhj_RbaP_IepBLkwaT6zNIeD5z

Table 1: Dataset statistics.

Dataset	RC15	RetailRocket
#sequences	200,000	195,523
#items	26,702	70,852
#clicks	1,110,965	1,176,680
#purchase	43,946	57,269

4.1 Experimental Settings

4.1.1 Datasets. We conduct experiments with two public accessible e-commerce datasets: RC15[5] and RetailRocket[6].

RC15. This is based on the dataset of RecSys Challange 2015. The dataset is session-based and each session contains a sequence of clicks and purchases. We remove the sessions whose length is smaller than 3 and then sample 200k sessions, resulting into a dataset which contains 1,110,965 clicks and 43,946 purchases over 26702 items. We sort the user-item interactions in one session according to the timestamp.

RetailRocket. This dataset is collected from a real-world e-commerce website. It contains sequential events of viewing and adding to cart. To keep in line with the RC15 dataset, we treat views as clicks and adding to cart as purchases. We remove the items which are interacted less than 3 times and the sequences whose length is smaller than 3. The final dataset contains 1,176,680 clicks and 57,269 purchases over 70852 items.

Table 1 summarizes the statistics of the two datasets.

4.1.2 Evaluation protocols. We adopt cross-validation to evaluate the performance of the proposed methods. The ratio of training, validation, and test set is 8:1:1. We randomly sample 80% of the sequences as training set. For validation and test sets, the evaluation is done by providing the events of a sequence one-by-one and checking the rank of the item of the next event. The ranking is performed among the whole item set. Each experiment is repeated 5 times, and the average performance is reported.

The recommendation quality is measured with two metrics: hit ration (HR) and normalized discounted cumulative gain (NDCG). HR@k is a recall-based metric, measuring whether the ground-truth item is in the top-k positions of the recommendation list. We can define HR for clicks as:

$$\text{HR(click)} = \frac{\text{\#hits among clicks}}{\text{\#clicks}}. \tag{10}$$

HR(purchase) is defined similarly with HR(click) by replacing the clicks with purchases. NDCG is a rank sensitive metric which assign higher scores to top positions in the recommendation list [20].

4.1.3 Baselines. We integrated the proposed SQN and SAC with four state-of-the-art (generative) sequential recommendation models:

- GRU [14]: This method utilizes a GRU to model the input sequences. The final hidden state of the GRU is treated as the latent representation for the input sequence.

[5]https://recsys.acm.org/recsys15/challenge/
[6]https://www.kaggle.com/retailrocket/ecommerce-dataset

Table 2: Top-k recommendation performance comparison of different models ($k = 5, 10, 20$) on RC15 dataset. NG is short for NDCG. Boldface denotes the highest score. $*$ denotes the significance p-value < 0.01 compared with the corresponding baseline.

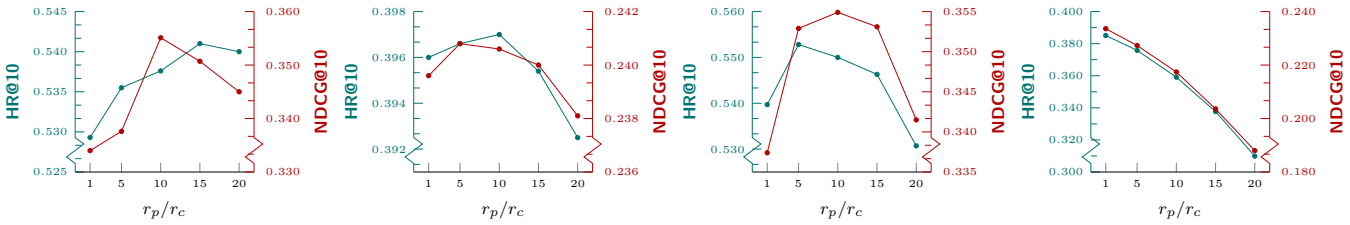
Models	purchase						click					
	HR@5	NG@5	HR@10	NG@10	HR@20	NG@20	HR@5	NG@5	HR@10	NG@10	HR@20	NG@20
GRU	0.3994	0.2824	0.5183	0.3204	0.6067	0.3429	0.2876	0.1982	0.3793	0.2279	0.4581	0.2478
GRU-SQN	0.4228*	0.3016*	0.5333*	0.3376*	0.6233*	0.3605*	**0.3020***	**0.2093***	**0.3946***	**0.2394***	**0.4741***	**0.2587***
GRU-SAC	**0.4394***	**0.3154***	**0.5525***	**0.3521***	**0.6378***	**0.3739***	0.2863	0.1985	0.3764	0.2277	0.4541	0.2474
Caser	0.4475	0.3211	0.5559	0.3565	0.6393	0.3775	0.2728	0.1896	0.3593	0.2177	0.4371	0.2372
Caser-SQN	0.4553*	0.3302*	0.5637*	0.3653*	0.6417*	0.3862*	**0.2742**	**0.1909**	**0.3613**	**0.2192**	**0.4381**	**0.2386**
Caser-SAC	**0.4866***	**0.3527***	**0.5914***	**0.3868***	**0.6689***	**0.4065***	0.2726	0.1894	0.3580	0.2171	0.4340	0.2362
NItNet	0.3632	0.2547	0.4716	0.2900	0.5558	0.3114	0.2950	0.2030	0.3885	0.2332	0.4684	0.2535
NItNet-SQN	0.3845*	0.2736*	0.4945*	0.3094*	**0.5766***	0.3302*	**0.3091***	**0.2137***	**0.4037***	**0.2442***	**0.4835***	**0.2645***
NItNet-SAC	**0.3914***	**0.2813***	**0.4964***	**0.3155***	0.5763*	**0.3357***	0.2977*	0.2055*	0.3906	0.2357*	0.4693	0.2557*
SASRec	0.4228	0.2938	0.5418	0.3326	0.6329	0.3558	0.3187	0.2200	0.4164	0.2515	0.4974	0.2720
SASRec-SQN	0.4336	0.3067*	0.5505	0.3435*	0.6442*	0.3674*	**0.3272***	**0.2263***	**0.4255***	**0.2580***	**0.5066***	**0.2786***
SASRec-SAC	**0.4540***	**0.3246***	**0.5701***	**0.3623***	**0.6576***	**0.3846***	0.3130	0.2161	0.4114	0.2480	0.4945	0.2691

Table 3: Top-k recommendation performance comparison of different models ($k = 5, 10, 20$) on RetailRocket. NG is short for NDCG. Boldface denotes the highest score. $*$ denotes the significance p-value < 0.01 compared with the corresponding baseline.

Models	purchase						click					
	HR@5	NG@5	HR@10	NG@10	HR@20	NG@20	HR@5	NG@5	HR@10	NG@10	HR@20	NG@20
GRU	0.4608	0.3834	0.5107	0.3995	0.5564	0.4111	0.2233	0.1735	0.2673	0.1878	0.3082	0.1981
GRU-SQN	**0.5069***	0.4130*	**0.5589***	0.4289*	**0.5946***	0.4392*	**0.2487***	**0.1939***	**0.2967***	**0.2094***	**0.3406***	**0.2205***
GRU-SAC	0.4942*	**0.4179***	0.5464*	**0.4341***	0.5870*	**0.4428***	0.2451*	0.1924*	0.2930*	0.2074*	0.3371*	0.2186*
Caser	0.3491	0.2935	0.3857	0.3053	0.4198	0.3141	0.1966	0.1566	0.2302	0.1675	0.2628	0.1758
Caser-SQN	0.3674*	0.3089*	0.4050*	0.3210*	0.4409*	0.3301*	0.2089*	0.1661*	0.2454*	0.1778*	0.2803*	0.1867*
Caser-SAC	**0.3871***	**0.3234***	**0.4336***	**0.3386***	**0.4763***	**0.3494***	**0.2206***	**0.1732***	**0.2617***	**0.1865***	**0.2999***	**0.1961***
NItNet	0.5630	0.4630	0.6127	0.4792	0.6477	0.4881	0.2495	0.1906	0.2990	0.2067	0.3419	0.2175
NItNet-SQN	**0.5895***	0.4860*	**0.6403***	0.5026*	**0.6766***	0.5118*	**0.2610***	**0.1982***	**0.3129***	**0.2150***	**0.3586***	**0.2266***
NItNet-SAC	**0.5895***	**0.4985***	0.6358*	**0.5162***	0.6657*	**0.5243***	0.2529*	0.1964*	0.3010*	0.2119*	0.3458*	0.2233*
SASRec	0.5267	0.4298	0.5916	0.4510	0.6341	0.4618	0.2541	0.1931	0.3085	0.2107	0.3570	0.2230
SASRec-SQN	**0.5681***	0.4617*	**0.6203***	0.4806*	**0.6619***	0.4914*	**0.2761***	**0.2104***	**0.3302***	**0.2279***	**0.3803***	**0.2406***
SASRec-SAC	0.5623*	**0.4679***	0.6127*	**0.4844***	0.6505*	**0.4940***	0.2670*	0.2056*	0.3208*	0.2230*	0.3701*	0.2355*

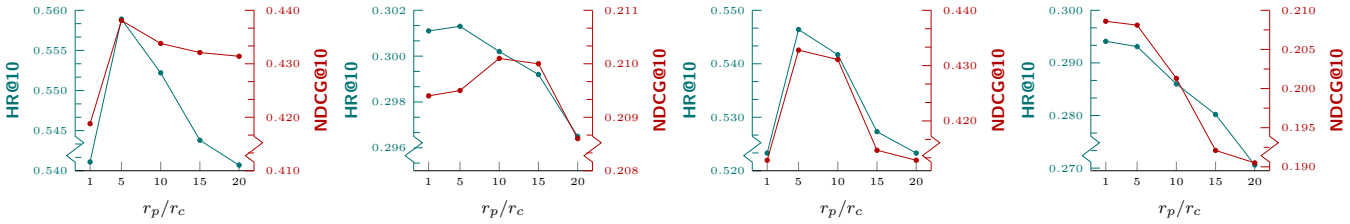
- Caser [38]: This is a recently proposed CNN-based method which captures sequential signals by applying convolution operations on the embedding matrix of previous interacted items.
- NItNet [42]: This method utilizes dilated CNN to enlarge the receptive field and residual connection to increase the network depth, achieving good performance with high efficiency.
- SASRec [21]: This baseline is motivated from self-attention and uses the Transformer [40] architecture. The output of the Transformer encoder is treated as the latent representation for the input sequence.

4.1.4 Parameter settings. For both of the datasets, the input sequences are composed of the last 10 items before the target timestamp. If the sequence length is less than 10, we complement the sequence with a padding item. We train all models with the Adam optimizer [22]. The mini-batch size is set as 256. The learning rate is set as 0.01 for RC15 and 0.005 for RetailRocket. We evaluate on the validation set every 2000 batches of updates. For a fair comparison, the item embedding size is set as 64 for all models. For GRU4Rec, the size of the hidden state is set as 64. For Caser, we use 1 vertical convolution filter and 16 horizontal filters whose heights are set from {2,3,4}. The drop-out ratio is set as 0.1. For NextItNet, we utilize the code published by its authors and keep the settings unchanged. For SASRec, the number of heads in self-attention is set as 1 according to its original paper [21]. For the proposed SQN and SAC, if not mentioned otherwise, the discount factor γ is set as



(a) SQN for purchase (b) SQN for click (c) SAC for purchase (d) SAC for click

Figure 3: Effect of reward settings on RC15



(a) SQN for purchase (b) SQN for click (c) SAC for purchase (d) SAC for click

Figure 4: Effect of reward settings on RetailRocket

0.5 while the ratio between the click reward (r_c) and the purchase reward (r_p) is set as $r_p/r_c = 5$.

4.2 Performance Comparison (RQ1)

Table 2 and Table 3 show the performance of top-k recommendation on RC15 and RetailRocket, respectively.

We observe that on the RC15 dataset, the proposed SQN method achieves consistently better performance than the corresponding baseline when predicting both clicks and purchases. SQN introduces a Q-learning head which aims to model the long-term cumulative reward. The additional learning signal from this head improves both clicks and purchase recommendation performance because the models are now trained to select actions which are optimized not only for the current state but also for future interactions. We can see that on this dataset, the best performance when predicting purchase interactions is achieved by SAC. Since the learned Q-values are used as weights for the supervised loss function, the model is "reinforced" to focus more on purchases. As a result, the SAC method achieves significant better results when recommending purchases. We can assume that the strong but sparse signal that comes with a purchase is better utilized by SAC.

On the RetailRocket dataset, we can see that both SQN and SAC achieve consistent better performance than the corresponding baseline in terms of predicting both clicks and purchases. This further verifies the effectiveness of the proposed methods. Besides, we can also see that even though SQN sometimes achieves the highest HR(purchase), SAC always achieves the best performance with respect to the NDCG of purchase. This demonstrates that the proposed SAC is actually more likely to push the items which may lead to a purchase to the top positions of the recommendation list.

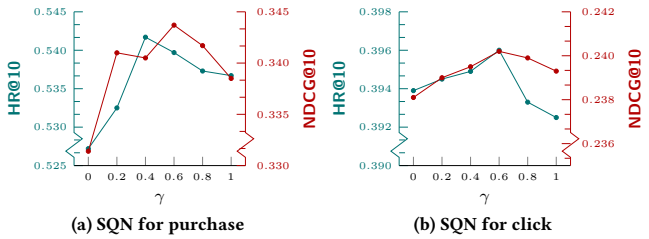
To conclude, it's obvious that the proposed SQN and SAC achieve consistent improvement with respect to the selected baselines. This demonstrates the effectiveness and the generalization ability of our methods.

4.3 RL Investigation(RQ2)

4.3.1 Effect of reward settings. In this part, we conduct experiments to investigate how the reward setting of RL affects the model performance. Figure 3 and Figure 4 show the results of HR@10 and NDCG@10 when changing the ratio between r_p and r_c (i.e., r_p/r_c) on RC15 and RetailRocket, respectively. We show the performance when choosing GRU as the base model. Results when utilizing other baseline models show similar trends and are omitted due to space limitation.

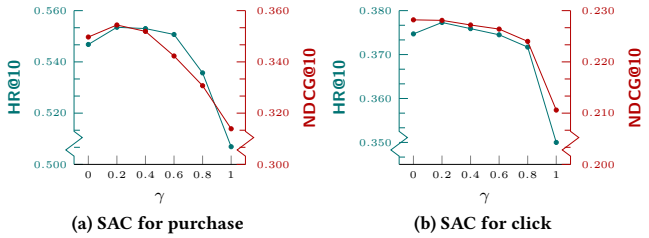
We can see from Figure 3a and Figure 4a that the performance of SQN when predicting purchase interactions start to improve when r_p/r_c increases from 1. It shows that when we assign a higher reward to purchases, the introduced RL head successfully drives the model to focus on the desired rewards. Performance start to decrease after reaching higher ratios. The reason may be that high reward differences might cause instability in the TD updates and thus affects the model performance. Figure 3c and Figure 4c shows that the performance of SAC when predicting purchase behaviours also improves at the beginning and then drops with the increase of r_p/r_c. It's similar with SQN.

For click recommendation, we can see from Figure 3b and Figure 4b that the performance of SQN is actually stable at the beginning (even increases a little) and then starts to decrease. There are two factors for this performance drop. The first is the instability of RL as discussed before. The second is that too much reward discrepancy might diminish the relative importance of clicks which constitute



(a) SQN for purchase (b) SQN for click

Figure 5: SQN with different discount factors



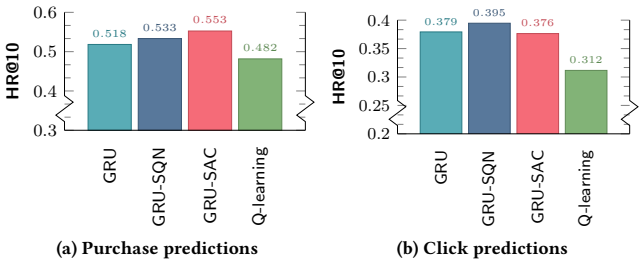
(a) SAC for purchase (b) SAC for click

Figure 6: SAC with different discount factors

the vast majority of the datapoints. This also helps to explain the performance drop of SAC as shown in Figure 3d and Figure 4d.

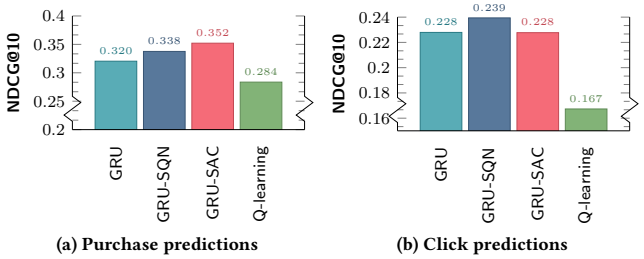
In addition, observing the performance of SQN and SAC when $r_p/r_c = 1$, we can find that they still perform better than the basic GRU. For example, when predicting purchases on the RC15 dataset, the HR@10 of SAC is around 0.54 according to Figure 3c while the basic GRU method only achieves 0.5183 according to Table 2. This means that even if we don't distinguish between clicks and purchases, the proposed SQN and SAC still works better than the basic model. The reason is that the introduced RL head successfully adds additional learning signals for long-term rewards.

4.3.2 Effect of the discount factor. In this part, we show how the discount factor affects the recommendation performance. Figure 5 and Figure 6 illustrates the HR@10 and NDCG@10 of SQN and SAC with different discount factors on the RC15 dataset. We choose GRU as the base recommendation model. The results on RetailRocket show similar trends and are omitted here. We can see that the performance of SQN and SAC improves when the discount factor γ increases from 0. $\gamma = 0$ means that the model doesn't consider long-term reward and only focuses on immediate feedback. This observation leads to the conclusion that taking long-term rewards into account does improve the overall HR and NDCG on both click and purchase recommendations. However, we can also see the performance decreases when the discount factor is too large. Compared with the game control domain in which there maybe thousands of steps in one MDP, the user interaction sequence is much shorter. The average sequence length of the two datasets is only 6. As a result, although $\gamma = 0.95$ or 0.99 is a common setting for game control, a smaller discount factor should be applied under the recommendation settings.



(a) Purchase predictions (b) Click predictions

Figure 7: Comparison of HR when only using Q-learning for recommendations.



(a) Purchase predictions (b) Click predictions

Figure 8: Comparison of NDCG when only using Q-learning for recommendations.

4.4 Q-learning for Recommendation (RQ3)

We also conduct experiments to examine the performance if we generate recommendations only using Q-learning. To make Q-learning more effective when perform ranking, we explicitly introduce uniformly sampled unseen items to provide negative rewards [31, 44]. Figure 7 and Figure 8 show the results in terms of HR@10 and NDCG@10 on the RC15 dataset, respectively. The base model is GRU. We can see that the performance of Q-learning is even worse than the basic GRU method. As discussed in section 2.2, directly utilizing Q-learning for recommendation is problematic and off-policy correction needs to be considered in that situation. However, the estimation of Q-values based on the given state is unbiased and exploiting Q-learning as a regularizer or critic doesn't suffer from the above problem. Hence the proposed SQN and SAC achieve better performance.

5 CONCLUSION AND FUTURE WORK

We propose self-supervised reinforcement learning for recommender systems. We first formalize the next item recommendation task and then analysis the difficulties when exploiting RL for this task. The first is the pure off-policy setting which means the recommender agent must be trained from logged data without interactions between the environment. The second is the lack of negative rewards. To address these problems, we propose to augment the existing recommendation model with another RL head. This head acts as a regularizer to introduce our specific desires into the recommendation. The motivation is to utilize the unbiased estimator of RL to fine-tune the recommendation model according to our own reward settings. Based on that, we propose SQN and SAC to perform joint

training of the supervised head and the RL head. To verify the effectiveness of our methods, we integrate them with four state-of-the-art recommendation models and conduct experiments on two real-world e-commerce datasets. Experimental results demonstrate that the proposed SQN and SAC are effective to improve the hit ratio, especially when predicting the real purchase interactions. Future work includes online tests and more experiments on other use cases, such as recommendation diversity promotion, improving watching time for video recommendation and so on. Besides, we are also trying to extend the framework for slate-based recommendation in which the action is recommending a set of items.

REFERENCES

[1] Richard Bellman. 1966. Dynamic programming. *Science* 153, 3731 (1966), 34–37.

[2] Keith Bradley and Barry Smyth. 2001. Improving recommendation diversity. In *Proceedings of the Twelfth Irish Conference on Artificial Intelligence and Cognitive Science, Maynooth, Ireland*. Citeseer, 85–94.

[3] 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 *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*. ACM, 456–464.

[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 *International Conference on Machine Learning*. 1052–1061.

[5] Chen Cheng, Haiqin Yang, Michael R Lyu, and Irwin King. 2013. Where you like to go next: Successive point-of-interest recommendation. In *Twenty-Third international joint conference on Artificial Intelligence*.

[6] Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259* (2014).

[7] Scott Fujimoto, David Meger, and Doina Precup. 2018. Off-policy deep reinforcement learning without exploration. *arXiv preprint arXiv:1812.02900* (2018).

[8] Yu Gong, Yu Zhu, Lu Duan, Qingwen Liu, Ziyu Guan, Fei Sun, Wenwu Ou, and Kenny Q Zhu. 2019. Exact-K Recommendation via Maximal Clique Optimization. *arXiv preprint arXiv:1905.07089* (2019).

[9] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In *Advances in neural information processing systems*. 2672–2680.

[10] Hado V Hasselt. 2010. Double Q-learning. In *Advances in Neural Information Processing Systems*. 2613–2621.

[11] Ruining He, Chen Fang, Zhaowen Wang, and Julian McAuley. 2016. Vista: A visually, socially, and temporally-aware model for artistic recommendation. In *Proceedings of the 10th ACM Conference on Recommender Systems*. ACM, 309–316.

[12] Ruining He and Julian McAuley. 2016. Fusing similarity models with markov chains for sparse sequential recommendation. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*. IEEE, 191–200.

[13] Balázs Hidasi and Alexandros Karatzoglou. 2018. Recurrent neural networks with top-k gains for session-based recommendations. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*. ACM, 843–852.

[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 and Domonkos Tikk. 2016. General factorization framework for context-aware recommendations. *Data Mining and Knowledge Discovery* 30, 2 (2016), 342–371.

[16] Jonathan Ho and Stefano Ermon. 2016. Generative adversarial imitation learning. In *Advances in neural information processing systems*. 4565–4573.

[17] Jonathan Ho, Jayesh Gupta, and Stefano Ermon. 2016. Model-free imitation learning with policy optimization. In *International Conference on Machine Learning*. 2760–2769.

[18] 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*. ACM, 368–377.

[19] 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. (2019).

[20] Kalervo Järvelin and Jaana Kekäläinen. 2002. Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems (TOIS)* 20, 4 (2002), 422–446.

[21] Wang-Cheng Kang and Julian McAuley. 2018. Self-attentive sequential recommendation. In *2018 IEEE International Conference on Data Mining (ICDM)*. IEEE, 197–206.

[22] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

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

[24] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. *Computer* 42, 8 (2009), 30–37.

[25] Lihong Li, Wei Chu, John Langford, and Robert E Schapire. 2010. A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th international conference on World wide web*. ACM, 661–670.

[26] Lihong Li, Wei Chu, John Langford, and Xuanhui Wang. 2011. Unbiased offline evaluation of contextual-bandit-based news article recommendation algorithms. In *Proceedings of the fourth ACM international conference on Web search and data mining*. ACM, 297–306.

[27] 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.

[28] Rémi Munos, Tom Stepleton, Anna Harutyunyan, and Marc Bellemare. 2016. Safe and efficient off-policy reinforcement learning. In *Advances in Neural Information Processing Systems*. 1054–1062.

[29] Emilio Parisotto, H Francis Song, Jack W Rae, Razvan Pascanu, Caglar Gulcehre, Siddhant M Jayakumar, Max Jaderberg, Raphael Lopez Kaufman, Aidan Clark, Seb Noury, et al. 2019. Stabilizing Transformers for Reinforcement Learning. *arXiv preprint arXiv:1910.06764* (2019).

[30] Steffen Rendle. 2010. Factorization machines. In *2010 IEEE International Conference on Data Mining*. IEEE, 995–1000.

[31] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian personalized ranking from implicit feedback. In *Proceedings of the twenty-fifth conference on uncertainty in artificial intelligence*. AUAI Press, 452–461.

[32] Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. 2010. Factorizing personalized markov chains for next-basket recommendation. In *Proceedings of the 19th international conference on World wide web*. ACM, 811–820.

[33] 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).

[34] Wenjie Shang, Yang Yu, Qingyang Li, Zhiwei Qin, Yiping Meng, and Jieping Ye. 2019. Environment Reconstruction with Hidden Confounders for Reinforcement Learning based Recommendation. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 566–576.

[35] Guy Shani, David Heckerman, and Ronen I Brafman. 2005. An MDP-based recommender system. *Journal of Machine Learning Research* 6, Sep (2005), 1265–1295.

[36] 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.

[37] 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.

[38] Jiaxi Tang and Ke Wang. 2018. Personalized top-n sequential recommendation via convolutional sequence embedding. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*. ACM, 565–573.

[39] Faraz Torabi, Garrett Warnell, and Peter Stone. 2018. Behavioral cloning from observation. *arXiv preprint arXiv:1805.01954* (2018).

[40] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*. 5998–6008.

[41] Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning* 8, 3-4 (1992), 229–256.

[42] Fajie Yuan, Alexandros Karatzoglou, Ioannis Arapakis, Joemon M Jose, and Xiangnan He. 2019. A Simple Convolutional Generative Network for Next Item Recommendation. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*. ACM, 582–590.

[43] Fajie Yuan, Xin Xin, Xiangnan He, Guibing Guo, Weinan Zhang, Chua Tat-Seng, and Joemon M Jose. 2018. fBGD: Learning embeddings from positive unlabeled data with BGD. (2018).

[44] Xiangyu Zhao, Liang Zhang, Zhuoye Ding, Long Xia, Jiliang Tang, and Dawei Yin. 2018. Recommendations with negative feedback via pairwise deep reinforcement learning. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 1040–1048.

[45] 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. *arXiv preprint arXiv:1902.05570* (2019).