Supervised Advantage Actor-Critic for Recommender Systems

Xin Xin
Shandong University
xinxin@sdu.edu.cn

Alexandros Karatzoglou
Google Research, London
alexkz@google.com

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

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

ABSTRACT

Casting session-based or sequential recommendation as reinforcement learning (RL) through reward signals is a promising research direction towards recommender systems (RS) that maximize cumulative profits. However, the direct use of RL algorithms in the RS setting is impractical due to challenges like off-policy training, huge action spaces and lack of sufficient reward signals. Recent RL approaches for RS attempt to tackle these challenges by combining RL and (self-)supervised sequential learning, but still suffer from certain limitations. For example, the estimation of Q-values tends to be biased toward positive values due to the lack of negative reward signals. Moreover, the Q-values also depend heavily on the specific timestamp of a sequence.

To address the above problems, we propose negative sampling strategy for training the RL component and combine it with supervised sequential learning. We call this method Supervised Negative Q-learning (SNQN). Based on sampled (negative) actions (items), we can calculate the "advantage" of a positive action over the average case, which can be further utilized as a normalized weight for learning the supervised sequential part. This leads to another learning framework: Supervised Advantage Actor-Critic (SA2C). We instantiate SNQN and SA2C with four state-of-the-art sequential recommendation models and conduct experiments on two real-world datasets. Experimental results show that the proposed approaches achieve significantly better performance than state-of-the-art supervised methods and existing self-supervised RL methods.

CCS CONCEPTS

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

KEYWORDS

Recommendation; Reinforcement Learning; Actor-Critic; Q-learning; Advantage Actor-Critic; Negative Sampling

ACM Reference Format:
Xin Xin, Alexandros Karatzoglou, Ioannis Arapakis, and Joemon M. Jose. 2022. Supervised Advantage Actor-Critic for Recommender Systems. In *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining (WSDM '22), February 21–25, 2022, Tempe, AZ, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3488560.3498494

1 INTRODUCTION

The online world offers a wealth of choices in both entertainment, purchasing and social networking. Over the last 20 years, users have been navigating these choices with the help of recommender systems (RS). Examples of applied RS include, but are not limited to, e-commerce [13], video platforms such as Youtube, TikTok, and music apps like Spotify [45]. Most of these use cases involve session-based or next-item recommendation, whereby users use a service to interact with items in a sequence and the recommendations are generated to recommend the most interesting items for next timestamp given the current user state, by using historical interaction sequences as the training input.

Session-based recommendation models can usually be trained in a (self-)supervised learning fashion, in which a sequential model (e.g., a transformer [17, 41] or a RNN [9]) is trained to predict the next item in the sequence itself, rather than some "external" labels [9, 17, 45]. This training approach is also widely adopted in language modeling tasks, to predict the next word given the previous word sequence [23]. However, supervised learning can also result into sub-optimal recommendations, since the loss function used in supervised learning is purely defined on the discrepancy between model predictions and the actual interactions in the sequence. Recommmendations from a model trained on such a loss function may not match the desired properties of a RS from the perspective of both users and service providers. For example, service providers may want to promote recommendations that can lead to real purchases not just clicks. The service provider may also wish to have long-term user engagement instead of focusing on immediate rewards and clicks. Other desirable properties of RS could also be taken into consideration, like diversity and novelty of the recommended item lists, which may lead to a multi-objective optimization problem [21, 31]. Recommendation models trained with simple supervised learning may encounter difficulties in tackling the above recommendation expectations and objectives.

Reinforcement learning (RL) has achieved success in game control [4, 24, 36, 37], robotics [19] and related fields [2, 22]. A RL agent is trained to take actions given the observation (state) of the

WSDM '22, February 21–25, 2022, Tempe, AZ, USA

ACM ISBN 978-1-4503-9132-0/22/02…$15.00
https://doi.org/10.1145/3488560.3498494

environment with the objective of getting the maximum discounted cumulative rewards. This long-term optimization perspective of RL is a good match with the nature of RS (i.e. getting maximum cumulative rewards in one interaction session). The flexibility that accompanies reward-based learning allows us to create models that can serve multiple recommendation objectives (e.g., promoting purchases, increasing diversity and so on). As a result, the use of RL for recommendation has become a promising research direction.

However, unlike game control and robotics, directly utilizing RL for RS comes with sets of unique difficulties and challenges. Model-free RL algorithms train the agent through an "error-and-correction" manner, in which the RL agent needs to interact with the environment and collect experience. The training procedure forces the agent to imitate good actions and avoid bad ones. Doing this under the setting of RS is often problematic, since interactions with an under-trained policy would negatively affect the user experience. A user may quickly abandon the service if he is continuously recommended irrelevant items. A typical solution is to perform off-policy learning from the logged implicit feedback data [1, 43]. This entails trying to infer a target policy from the data generated by a different behavior policy, which is still an open research problem due to its high variance [25]. Moreover, learning from implicit feedback also introduces the challenge of insufficient negative signal [29, 43]. Another alternative is to use model-based RL algorithms, in which a model is firstly constructed to simulate the environment (users). Then the agent can learn from the interactions with the simulated environment [2, 35]. However, these two-stage methods depend heavily on the accuracy of the constructed simulator.

Recently, self-supervised reinforcement learning [43] has been proposed for RS, achieving promising results on off-line evaluation metrics. Two learning frameworks namely Self-Supervised Q-learning (SQN) and Self-Supervised Actor-Critic (SAC) are proposed. The key insight of self-supervised RL is to utilize the RL component as a form of a regularizer to fine-tune the recommendation model towards the defined rewards, for instance in the e-commerce domain provide recommendations that lead to more purchases rather than just clicks [43]. Although SQN and SAC achieve good performance, they still suffer from some limitations. For example, the RL head[1] in SQN and SAC is only defined on positive (interacted) actions (items), so the negative comparison signals only come from the cross-entropy loss of the supervised part. As a result, the RL head contributes to reward-based learning but cannot be used to generate recommendations, as it lacks negative feedback to remove the bias introduced by the existence of only positive reward signals. Moreover, SAC uses the output Q-values[2] as the critic to re-weight the actor (supervised part). However, the Q-values depend heavily on the specific timestamp of a sequence, which further introduces bias to the learning procedure.

To address the above issues, we first propose to introduce a negative sampling strategy for training RL in a RS setting and then combine it with supervised sequential learning. We call this Supervised Negative Q-learning (SNQN). Another interpretation of negative sampling in RL is imitation learning under sparse reward settings [27]. Different from SQN, which only performs RL on positive actions (clicks, views, etc.), the RL output head of SNQN is learned on both positive actions and a set of sampled negative actions. This design allows the RL part of the SNQN to not only act as a regularizer but also as a good ranking model, which can also be used to generate recommendations. Based on the sampled negative actions and the estimate of the Q-values, we can moreover calculate the "advantage" of a positive action over the other actions. We then propose the Supervised Advantage Actor-Critic (SA2C), that uses this advantage instead of the raw Q-values to re-weight the supervised output layer. The advantage values can be seen as normalized Q-values that help us alleviate the bias from sequence timestamp on the estimation of Q-values.

To summarize, this work makes the following contributions:

- We propose SNQN to introduce negative sampling into the RL training of the RS model and then combine it with supervised sequential learning. As a result, both the supervised head and the RL head can be used to generate recommendations. We show that joint training of the two heads with a shared base model helps to achieve better performance than separate learning.
- We propose SA2C to calculate the *advantage* of a positive action. This advantage can be seen as a normalized Q-value and is further utilized to re-weight the supervised component.
- We integrate the proposed SNQN and SA2C with four state-of-the-art recommendation models and conduct experiments on two real-world e-commerce datasets. Experimental results demonstrate the proposed methods are effective in improving the performance of RS compared to existing methods.

2 RELATED WORK

Early next-item recommendation models were based on the utilization of Markov Chains [3, 8, 30] and factorization methods [10, 28], which are limited in model expressiveness for complex sequential signals [39, 45]. Recently, plenty of deep learning-based approaches have been proposed. Both recurrent neural networks (RNN) and convolutional neural networks (CNN) have shown promising results to model the sequence [9, 39, 45]. Transformer architectures have been proven to be highly successful [41] for language modeling tasks, and the use of self-attention for recommendations has also received a lot of attention [17].

RL has been previously applied in RS. Chen et al. [1] proposed to calculate a propensity score to perform off-policy correction for off-policy learning. However, the estimation of propensity scores has high variance and there are tricks like smoothing or clipping to train the model (we discuss this in section 4). Model-based RL approaches [2, 33, 47] attempt to eliminate the off-policy issue by building a model to simulate the environment. The policy can then be trained through interactions with the simulator. However, two-stage approaches depend heavily on the accuracy of the simulator. Although related methods, such as generative adversarial networks (GANs) [6], achieve good performance when generating content like images and speeches, simulating users' responses is a much more complex and difficult task [2].

Recently, Xin et al. [43] proposed self-supervised reinforcement learning for RS. Two learning frameworks SQN and SAC are suggested. SQN augments the recommendation model with two heads.

[1] For simplicity, we make "head" and "output layer" interchangeable in this paper.
[2] The Q-value for a state and action is an estimate of the expected cumulative reward under this state-action pair.

One is defined on the supervised mode and the other RL head is based on the Q-learning for positive reward actions. SQN co-trains the supervised loss and RL loss to conduct transfer learning between each other [43]. In this way long term rewards e.g. a purchase at the end of a session can be incorporated into the learning process, while the model is still trained efficiently on logged data. As the computed Q-values are an estimation of the goodness of the actions, SAC further utilizes these Q-values to re-weight the supervised part. SQN and SAC can be seen as attempts to utilize a Q-learning based RL estimator to "reinforce" existing sequential (or session-based) supervised recommendation models [43] and achieve promising results on off-line evaluation metrics.

Moreover, research on slate-based recommendation has also been conducted in [1, 2, 5, 15], where actions are considered to be sets (slates) of items. This setting leads to an exponentially increased action space. Finally, bandit algorithms are also reward-driven and have long-term optimization perspective. However, bandit algorithms assume that taking actions does not affect the state [20], while actually recommendations do have an effect on user behavior [32]; hence RL is a more suitable choice for the RS task. Another related field is imitation learning, where the policy is learned from expert demonstrations [11, 12, 27, 40].

3 METHOD

3.1 Next-Item Recommendation Formulation

Let $\mathcal{I}$ denote the item set, then a user-item interaction sequence[3] can be represented as $x_{1:t} = \{x_1, x_2, ..., x_{t-1}, x_t\}$, where $x_i \in \mathcal{I} (0 < i \leq t)$ denotes the interacted item at timestamp i. The task of next-item recommendation is to recommend the most relevant item x_{t+1} to the user, given the sequence of $x_{1:t}$.

From the conventional supervised learning perspective, this task can be regarded as a multi-class classification problem. A common solution is to build a recommendation model whose output is 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. Each candidate item corresponds to a class. Then the recommendation list for timestamp $t+1$ can be generated by choosing top-k items according to $\mathbf{y}_{t+1}$. Typically one can use a generative sequential model $G(\cdot)$ to encode the input sequence into a hidden state $\mathbf{s}_t$ as $\mathbf{s}_t = G(x_{1:t})$. Generally speaking, plenty of deep-learning based models [9, 17, 39, 45] can serve as the generative model $G(\cdot)$. After that, a decoder can be utilized 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 [9, 17, 39, 45]. Finally, the whole model (agents) can be trained by optimizing the supervised cross-entropy loss based on the classification logits.

3.2 Reinforcement Learning Setup

From the perspective of RL, the next item recommendation task can be formulated as a Markov Decision Process (MDP) [34], in which the recommendation agent interacts with the environments $\mathcal{E}$ (users) by sequentially recommending items to maximize the discounted cumulative rewards. The MDP can be defined by tuples of $(\mathcal{S}, \mathcal{A}, \mathbf{P}, R, \rho_0, \gamma)$ [1, 2, 43] where

- $\mathcal{S}$: a continuous state space to describe the user state. If we reuse the hidden state of the supervised sequential model discussed in section 3.1, 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 to recommend the selected item. In off-line training data, we can get the positive action at timestamp t from the input sequence (i.e., $a_t^+ = x_{t+1} (t \geq 0)$).
- $\mathbf{P}$: $\mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$ is the state transition probability. When learning from off-line data, we can make an assumption that only positive actions can affect the user state. In other words, taking a negative (unobserved) action doesn't update the user state [15, 46].
- R: $\mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is the reward function, where $r(\mathbf{s}, a)$ denotes the immediate reward by taking action a at state $\mathbf{s}$. The flexible reward scheme allows the agent to optimize the recommendation models towards expectations that are not captured by simple supervised loss functions.
- ρ_0 is the initial state distribution with $\mathbf{s}_0 \sim \rho_0$.
- γ is the discount factor for future rewards.

The goal of RL is to seek a target policy $\pi_\theta(a|\mathbf{s})$ so that sampling trajectories according to $\pi_\theta(a|\mathbf{s})$, would lead to the maximum expected cumulative reward:

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

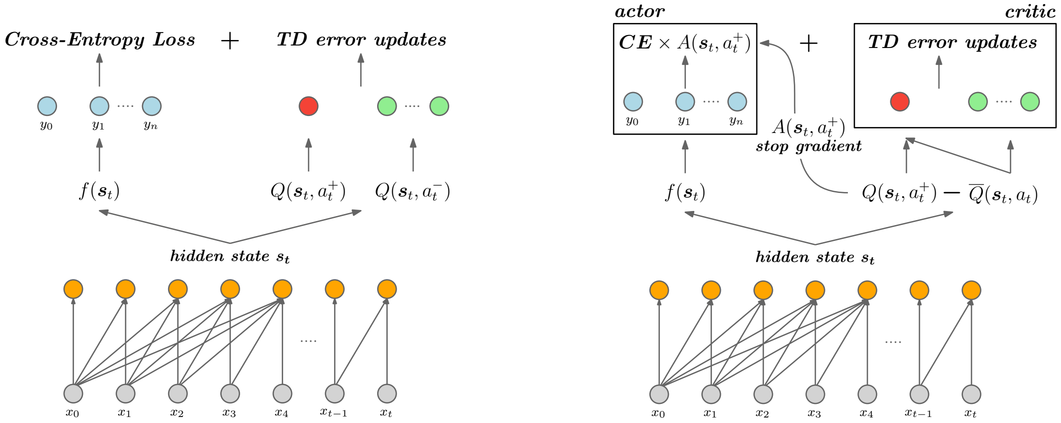
where $\theta \in \mathbb{R}^d$ denotes policy parameters. Note that the expectation is taken over trajectories $\tau = (\mathbf{s}_0, a_0, \mathbf{s}_1, ...)$, which are obtained by performing actions according to the target policy.

In on-line RL environments like game control, it's easy to sample the trajectories $\tau \sim \pi_\theta$ and the agent is trained through an "error-and-correction" approach. However, under the RS setting, we cannot afford to make "errors" (i.e. letting the user interact with under-trained policies) due to the negative impact on the user experience. Even if we can split a small portion of traffic to make the RL agent interact with live users, the final recommended items may still be controlled by other recommenders with different policies, since many recommendation models are deployed in a real-live RS. As a result, the sampled trajectories will come from another behavior policy $\tau \sim \beta$. Off-policy learning is still a difficult research problem due to high variance and distribution mismatch [1, 25]. Although some value-based RL algorithms like Q-learning [36] use a replay buffer to reuse past experience, they need the data distribution of the replay buffer to be similar with the target policy π_θ [4]. Moreover, value-based algorithms are actually regression-based methods, which don't perform well in ranking-oriented RS problems, as shown in [43]. In what follows, we will illustrate the detail of our proposition to use RL for RS by introducing negative sampling into RL, and then combine RL with supervised learning for more efficient off-line learning.

3.3 Supervised Negative Q-learning

Given an input user-item interaction sequence $x_{1:t}$ and an existing recommendation model $G(\cdot)$, the supervised training loss can be

[3] In the real-world scenario, there may be different kinds of interactions. For instance, in e-commerce, the interactions can be clicks, purchases, add to basket and so on. In video platforms, the interactions can be characterized by the watching time of a video.



(a) SNQN architecture.

(b) SA2C architecture. CE is short for cross-entropy.

Figure 1: The learning framework architectures of SNQN and SA2C.

defined as the cross-entropy over the classification distribution:

$$L_s = -\sum_{i=1}^{n} Y_i log(p_i), \text{where } p_i = \frac{e^{y_i}}{\sum_{i'=1}^{n} e^{y_{i'}}}. \tag{2}$$

Y_i is an indicator function which is defined as $Y_i = 1$ if the user interacted with the i-th item in the next timestamp. Otherwise, $Y_i = 0$. It's obvious that the cross-entropy loss will push the positive logits to high values. Meanwhile, the cross-entropy loss can also provide negative learning signals by pushing down the output values of items that the user has not interacted with. This is particularly helpful in a RS setting where ranking items which are likely to be interacted by the user in the top-k positions is the main goal.

Since $G(\cdot)$ already encodes the input sequence into a latent state $\mathbf{s}_t$, we can directly reuse this $\mathbf{s}_t$ as the state for the RL training. This sharing schema of the base model enables the transfer of knowledge between supervised learning and RL. Upon the shared base model $G(\cdot)$, we formulate another output layer to map the state into Q-values:

$$Q(\mathbf{s}_t, a_t) = \delta(\mathbf{s}_t\mathbf{h}_t^T + b) = \delta(G(x_{1:t})\mathbf{h}_t^T + b), \tag{3}$$

where δ denotes the activation function, $\mathbf{h}_t$ and b are trainable parameters of the Q-learning output layer.

When learning from logged implicit feedback data, it's often the case that there are no negative reward signals [14, 29]. Performing Q-learning solely based on positive reward signals (clicks, views, etc.), where the negative interaction signals are not provided, would lead to a model with a positive bias. As a result, such output Q-values based on only observed (positive) actions cannot be used for generating recommendation. To address this issue, we propose to introduce a negative reward sampling strategy for the RL training procedure. More precisely, the Q-learning loss function of SNQN is defined not only on positive action rewards but also on the sampled negative ones. To this end, we define the one-step time difference (TD) Q-loss of SNQN as:

$$\begin{aligned} L_q = \underbrace{(r(\mathbf{s}_t, a_t^+) + \gamma \max_{a'} Q(\mathbf{s}_{t+1}, a') - Q(\mathbf{s}_t, a_t^+))^2}_{L_p\text{: positive TD error}} \\ + \underbrace{\sum_{a_t^- \in N_t} (r(\mathbf{s}_t, a_t^-) + \gamma \max_{a'} Q(\mathbf{s}_t, a') - Q(\mathbf{s}_t, a_t^-))^2,}_{L_n\text{: negative TD error}} \end{aligned} \tag{4}$$

where a_t^+ and a_t^- are the positive action and negative action at timestamp t, respectively. N_t denotes the set of sampled unobserved (negative) actions. Note that in the negative TD error, the maximum operation is performed in $Q(\mathbf{s}_t, a')$ other than $Q(\mathbf{s}_{t+1}, a')$ because we assume that taking negative actions will not affect the user state as discussed in section 3.2. In our implementation, we assign a constant reward value r_n for negative actions (i.e.,$r(\mathbf{s}_t, a_t^-) = r_n$), while the positive reward $r(\mathbf{s}_t, a_t^+)$ we can define it according to the specific demands of the recommendation domain. For example, in e-commerce we can assign a higher reward to actions which lead to purchases rather than just clicks. We then jointly train the supervised loss and the RL loss on the replay buffer generated from the logged implicit feedback data:

$$L_{snqn} = L_s + L_q. \tag{5}$$

Figure 1a shows the architecture of SNQN. We use double Q-learning for better learning stability [7] and alternately train two copies of model parameters. The detail training procedure of SNQN can be found in Algorithm 1 in the appendix.

3.4 Supervised Advantage Actor-Critic

Actor-Critic (AC) methods have been successfully used in the RL research area. The key idea of AC methods is the introduction of a critic that evaluates the goodness of an action taken and assigns higher weights to actions with high cumulative rewards. In the proposed SNQN method, the supervised component can be seen as the actor which aims at imitating the logged user behavior. A simple solution for the critic is to use the output Q-values from the

RL head, as these Q-values measure the cumulative rewards the system gains given the state-action pair. However, these Q-values are sensitive to the specific timestamp of the sequence. For example, a bad action in an early timestamp of a long sequence could also have a high Q-value since Q-values are based on the cumulative gains of all the following actions in this sequence.

Instead of the absolute Q-value, what we actually would like to measure is how much "advantage" we obtain by applying an action, compared to the average case (i.e. average Q-values). This advantage can help us alleviate the bias introduced from the sequence timestamp. However, calculating the average Q-values along the whole action space would introduce additional computation cost, especially when the candidate item set is large. For this reason, we have introduced negative samples in the proposed SNQN methods. As a result, a concise solution is to calculate the average among the sampled actions (including both positive and negative examples) as an approximation. Based on this motivation, the average Q-values can be defined as:

$$\overline{Q}(\mathbf{s}_t, a) = \frac{\sum_{a' \in a_t^+ \cap N_t} Q(\mathbf{s}_t, a')}{|N_t| + 1}. \tag{6}$$

The advantage of an observed (positive) action is formulated as:

$$A(\mathbf{s}_t, a_t^+) = Q(\mathbf{s}_t, a_t^+) - \overline{Q}(\mathbf{s}_t, a). \tag{7}$$

We can then use this advantage to re-weight the actor (i.e. the supervised head). If a positive action has higher advantage over the average, we increase its weight in the supervised training procedure, and vice versa.

To enhance stability, we stop the gradient flow and fix the Q-values when they are used to calculate the average and advantage. We then train the actor and critic jointly. The training loss of SA2C is formulated as:

$$L_{sa2c} = L_a + L_q, \text{ where } L_a = L_s \cdot A(\mathbf{s}_t, a_t^+). \tag{8}$$

Figure 1b illustrates the architecture of SA2C. During the training procedure, the learning of Q-values can be unstable [26], particularly in the early stage. To mitigate these issues, we pre-train the model using SNQN in the first T training steps (batches). When the Q-values become more stable, we start to use the advantage to re-weight the actor and perform updates according to the architecture of Figure 1b. We use double Q-learning and the training procedure of SA2C is similar to Algorithm 1 except for the computation of advantage and the re-weighting of L_s.

3.5 Discussion

We provide a brief discussion about the connections between our algorithms and some related methods.

By assigning a negative reward r_n to unobserved actions, SNQN explicitly introduces negative action signals in the RL head. As a result, both the supervised head and the RL head of SNQN can be used to generate recommendations[4]. Compared to SQN, which can only use the supervised head to generate recommendation, SNQN provides a more flexible choice to switch between the two heads. For example, if we want the agent to imitate more the logged user data, we can use the supervised head. On the contrary, if we want the RS to be more reward-driven, we can use the RL head. Moreover, Reddy et al. [27] has shown that imitation learning through expert demonstrations can be achieved with promising performance, by assigning a constant positive reward for matching the demonstrated actions and a constant negative reward for other behaviors. From that perspective, the introduced negative sampling strategy of SNQN makes the RL component a good imitation learning agent.

A related work to SA2C is [1], in which the authors propose the use of an off-policy corrected policy-gradient method. Policy-gradient uses the cumulative reward to re-weight the cross-entropy loss. However, it's a Monte Carlo (MC)-based method which needs the interaction session to end first and then calculate the cumulative rewards at each timestamp. In contrast, SA2C calculates the advantage of an action through the RL output layer, which is a more fine-grained estimate of the value of a potential action during the session. Furthermore, the cumulative reward can also introduce bias from the sequence timestamp, while the advantage estimates in SA2C can be seen as normalized Q-values which help to alleviate this influence. The effect of off-policy correction will be discussed in the experimental section.

4 EXPERIMENTS

In this section, we report experiments [5] on two real-world datasets to evaluate the proposed SNQN and SA2C in the e-commerce scenario. Both datasets contain click and purchase interactions. We use the supervised head to generate recommendations without special mention. We address the following research questions:

RQ1: How do the proposed methods perform when integrated with different base models?

RQ2 What is the performance if we use the Q-leaning head to generate recommendation?

RQ3: What is the performance if we introduce an additional off-policy correction term in the actor of SA2C?

RQ4: How does the negative sampling strategy affect the performance?

4.1 Experimental Settings

4.1.1 Datasets. We conduct experiments with two publicly accessible session-based recommendation datasets: RC15[6] and RetailRocket[7]. The description of datasets can be found in the appendix.

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 sequences as the training set. For validation and test, the evaluation is done by providing the events (i.e., interacted items) of a sequence one-by-one and then checking the rank of the ground-truth item for the next timestamp. The ranking is performed among the whole item set. Each experiment is repeated five times, and the average performance is reported.

The recommendation quality is measured with two metrics: Hit Ratio (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

[4] Recommendation can be generated from the RL head by selecting highest Q-values.

[5] The implementation code and data can be found at https://drive.google.com/file/d/185KB520pBLgwmiuEe7JO78kUwUL_F45t/view?usp=sharing

[6] https://recsys.acm.org/recsys15/challenge/

[7] https://www.kaggle.com/retailrocket/ecommerce-dataset

can define HR for clicks as:

$$\text{HR(click)} = \frac{\text{\#hits among clicks}}{\text{\#clicks in test}}. \tag{9}$$

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 [16].

Since our experiments focus on the e-commerce scenario, which aims to promote purchases, for evaluation purposes we assign a higher reward to actions leading to purchases (i.e. conversions) compared to actions leading to only clicks. If a recommended item is not interacted by the user, we give this action a zero reward. As a result, the cumulative reward for evaluation is proportional to HR. For example, a higher HR(purchase) means a larger portion of recommended items would lead to purchase reward, and thus a higher cumulative reward on purchases. For simplicity, we only report the results with HR. The results for cumulative reward show almost same trend as HR.

4.1.3 Baselines. We integrated the proposed SNQN and SA2C with four state-of-the-art sequential recommendation models: GRU [9], Caser [39], NItNet [45], and SASRec [17]. The detail description of baselines and parameter settings can be found in the appendix.

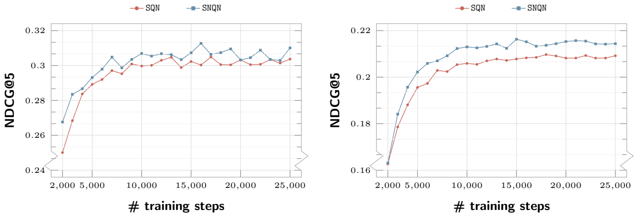
4.2 Performance Comparison (RQ1)

Table 1 shows the performance of top-k recommendations on RC15. The results on RetailRocket are illustrated in the appendix. We can have the following observations:

(1) On both datasets, the proposed SNQN achieves better performance than the supervised base model and SQN, on both click and purchase recommendations. This demonstrates that the introduced negative sampling strategy on the RL head does improve the learning performance also on the supervised component. This can be attributed to the shared recommendation model $G(\cdot)$ between the supervised part and the RL part. We also observe that SNQN achieves faster convergence than SQN. Figure 2 shows the comparison between model convergence under the same learning rate on the validation set of RC15, using GRU as the base model $G(\cdot)$. Results on RetailRocket and other base models lead to the same conclusion. This further demonstrates that the introduced negative sampling helps the model to learn faster and improves its performance.

(2) SA2C achieves better performance than SAC in most cases. This indicates that the advantage estimate used in SA2C is a more effective critic compared with the raw Q-values used in SAC. This can be attributed to the fact that the advantage estimation helps to alleviate the sequence timestamp bias. (3) SA2C always achieves the highest NDCG, which suggests that SA2C is more effective in pushing good actions (recommended items) to top ranking positions. This is due to the fact that positive actions are weighted (advantaged) in a more effective manner during the training procedure of SA2C.

To conclude, the proposed SNQN and SA2C introduce significant improvements compared to existing methods, especially SA2C which provides the best performance in most cases.



(a) Purchase predictions

(b) Click predictions

Figure 2: Model convergence comparison on RC15

4.3 Recommendation from Q-learning (RQ2)

Table 2 shows the performance comparison when we use the Q-learning head to generate recommendations. We compare the performance of SNQN with a simple double Q-learning (DQN) algorithm with the same negative sampling strategy but without a supervised head upon the base model. The performance of SA2C is not significantly different with SNQN as the two methods are essentially identical with regards to the Q-learning head. We use the same base model GRU and the same hyper-parameters for DQN and SNQN. Results on the other base models show identical trends. We observe that SNQN achieves better performance than DQN in all evaluation metrics on both purchase and click predictions. Combined with the results of Table 1 and Table 6, we observe that joint training of supervised learning and RL with shared base models helps to improve the performance of each component. Based on this finding, we believe that transfer learning between supervised learning and RL would be a promising research direction. RL makes supervised models to be more reward-driven, while supervised learning improves the data efficiency of RL.

4.4 Effect of Off-Policy Correction (RQ3)

Chen et al. [1] introduced an off-policy correction term (propensity score) for the policy-gradient method. The propensity score is defined as $\rho = \frac{\pi_\theta(a|\mathbf{s})}{\beta(a|\mathbf{s})}$. In this subsection, we investigate the effect of this propensity score when introduced into the actor component of SA2C. In that case, the training loss of the actor becomes:

$$L_{a-off} = L_s \cdot A(\mathbf{s}_t, a_t^+) \cdot \rho. \tag{10}$$

We also introduce another NDCG-based off-policy corrected evaluation metric [42] which is formulated as

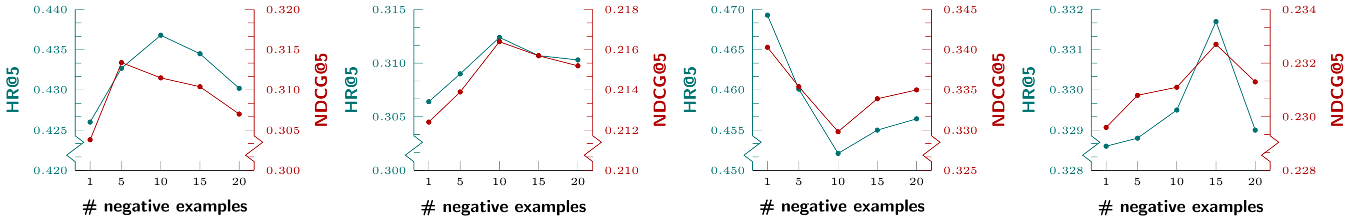
$$NG_{off} = \frac{\sum \frac{NDCG}{\beta}}{\sum \frac{1}{\beta}}. \tag{11}$$

In this implementation, we use the item frequency to approximate the behavior policy β, which is also adopted in [38]. Table 3 shows the result when generating top-10 recommendations with GRU as the base model. Results on the other base models lead to the same conclusion. We note the following observations:

(1) On the standard evaluation metric NDCG, off-policy correction doesn't improve the score. The reason for this is that the normal NDCG is actually defined on non-corrected data, so the non-corrected actor performs better at this evaluation metric.

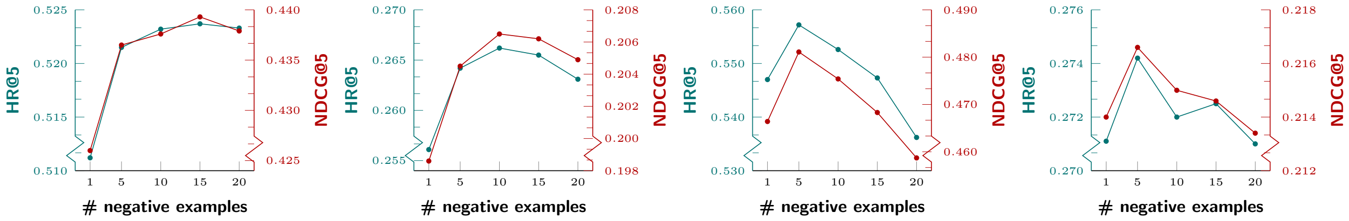
Table 1: Top-k recommendation performance comparison of different models ($k = 5, 10, 20$) on RC15 dataset. Recommendations are generated from the supervised head. NG is short for NDCG. Boldface denotes the highest score.

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-SNQN	0.4368	0.3115	0.5428	0.3460	0.6316	0.3686	0.3124	0.2164	0.4067	0.2469	0.4856	0.2669
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
GRU-SA2C	**0.4514**	**0.3297**	**0.5606**	**0.3652**	**0.6420**	**0.3859**	**0.3287**	**0.2307**	**0.4214**	**0.2606**	**0.5000**	**0.2806**
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-SNQN	0.4781	0.3460	0.5876	0.3816	0.6657	0.4015	0.2800	0.1951	0.3682	0.2237	0.4465	0.2436
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
Caser-SA2C	**0.4917**	**0.3635**	**0.6000**	**0.3989**	**0.6796**	**0.4192**	**0.2948**	**0.2068**	**0.3835**	**0.2356**	**0.4596**	**0.2549**
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-SNQN	0.3969	0.2803	0.5039	0.3152	0.5876	0.3363	0.3153	0.2176	0.4098	0.2482	0.4896	0.2686
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
NItNet-SA2C	**0.4382**	**0.3171**	**0.5403**	**0.3505**	**0.6259**	**0.3722**	**0.3410**	**0.2395**	**0.4348**	**0.2699**	**0.5113**	**0.2897**
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-SNQN	0.4435	0.3163	0.5581	0.3535	0.6450	0.3742	0.3284	0.2267	0.4271	0.2588	0.5083	0.2794
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
SASRec-SA2C	**0.4705**	**0.3385**	**0.5756**	**0.3728**	**0.6648**	**0.3956**	**0.3444**	**0.2407**	**0.4402**	**0.2719**	**0.5194**	**0.2920**



(a) SNQN for purchase **(b) SNQN for click** **(c) SA2C for purchase** **(d) SA2C for click**

Figure 3: Effect of number of negative samples on RC15



(a) SNQN for purchase **(b) SNQN for click** **(c) SA2C for purchase** **(d) SA2C for click**

Figure 4: Effect of number of negative samples on RetailRocket

(2) On NG_{off}, the off-policy correction helps the model to achieve better performance for click predictions but not for purchases. The reason for this is that clicks account for the biggest part of the dataset. Hence the off-policy correction term is actually better defined to correct the click data, leading to a better performance of NG_{off} for clicks, while the high variance of the off-policy correction for the small portion of purchase data leads to less of an improvement. This observation indicates that perhaps we should design different corrections for different kinds of interactions.

Table 2: Recommendation from the RL head. Boldface denotes the highest score. DQN denotes only a Q-learning head is used without the supervised head.

	Methods	purchase		click	
		HR@5	NG@5	HR@5	NG@5
RC15	DQN	0.3642	0.2476	0.2096	0.1353
	SNQN	**0.3698**	**0.2497**	**0.2286**	**0.1495**
Retail Rocket	DQN	0.2952	0.2204	0.1368	0.0961
	SNQN	**0.3124**	**0.2422**	**0.1546**	**0.1103**

Table 3: Effect of off-policy correction. w/o means without off-policy correction in the actor while w means the opposite. Boldface denotes the highest score.

	Methods	purchase		click	
		NDCG	NG_{off}	NDCG	NG_{off}
RC15	w/o	**0.3652**	**0.1077**	**0.2606**	0.0767
	w	0.3551	0.1064	0.2595	**0.0781**
Retail Rocket	w/o	**0.4897**	**0.2171**	**0.2308**	0.0861
	w	0.4771	0.2147	0.2238	**0.0872**

In our experiment, we found that computing the off-policy correction term involves a lot of normalization techniques (e.g., clipping and smoothing) [1]. The behavior policy β can also be a long-tail distribution [38]. This introduces substantial noise and high variance into the training procedure. Designing more effective and stable off-policy correction terms remains an open research problem.

4.5 Hyperparameter Study (RQ4)

In this section, we conduct a series of experiments to demonstrate the effect of negative sampling of the RL component. Figure 3 and Figure 4 show the recommendation accuracy with different sizes of negative examples (i.e. $|N_t|$) on RC15 and RetailRocket, respectively (the base model is GRU). Here, it is made evident that, on both click and purchase predictions, the recommendation performance initially increases and then decreases (except in Figure 3c). When more negative actions are introduced, the model has more data to learn from. By introducing negative actions, the model does not only learn that actions leading to purchases are better than actions leading to clicks, but also learns to draw a contrast between negative (uninteracted) and positive actions. Increasing the sample size means that the model can have access to more diverse negative signals and, thus, leads to better performance. However, a very large negative sample size may bias the model to negative values and degrade performance. Regarding Figure 3c, we observe that the model also achieves a good performance with small sample sizes. The reason could be attributed to that a small sample size would introduce more noise into estimation of the advantage. This noise may help the model to find a better local optimal with higher performance but also requires more update steps to converge. We have observed in our experiments that SA2C needs more iterations to converge when the sample size is small.

Table 4: Effect of negative rewards settings on RC15.

	r_n	purchase		click	
		HR@5	NG@5	HR@5	NG@5
SNQN	0	0.4368	0.3115	0.3124	0.2164
	-0.5	0.4324	0.3043	0.3118	0.2158
	-1.0	0.4345	0.3091	0.3108	0.2160
	-2.0	0.4269	0.3072	0.3128	0.2173
SA2C	0	0.4514	0.3297	0.3287	0.2307
	-0.5	0.4479	0.3263	0.3326	0.2332
	-1.0	0.4486	0.327	0.332	0.2333
	-2.0	0.4511	0.3274	0.3321	0.2335

Table 4 shows the effect of different negative reward settings (i.e., r_n) on RC15 dataset when using GRU as the base model. Results on RetailRocket lead to same conclusion. Generally speaking, r_n can be seen as the strength of negative signals. We can see from Table 4 that different r_n settings make no significant difference regarding the recommendation performance. However, through the performance comparison between SNQN and SQN, we can find that whether there are negative samples or not in the RL training procedure will dramatically affect the recommendation accuracy. This conforms with the finding in [27, 44].

5 CONCLUSION AND FUTURE WORK

In this paper, we propose two learning frameworks (SNQN and SA2C) to explore the usage of RL under recommendation settings. SNQN combines supervised learning and RL with the shared base model and introduces negative sampling into the RL training procedure. The explicitly introduced negative comparison signals help the RL output layer to perform good ranking. Based on the sampled actions, SA2C first computes the advantage of actions which can be seen as normalized Q-values and then use this advantage estimate as a critic to re-weight the actor. To verify the effectiveness of our methods, we integrate them into four state-of-the-art recommendation models and conduct experiments on two real-world e-commerce datasets. Our experimental findings demonstrate that the proposed SNQN and SA2C are effective in further improving the recommendation performance, compared to existing self-supervised RL methods.

Future work will involve online tests and multiple Q-heads with more kinds of rewards, such as diversity and novelty of recommendation, leading to a multi-objective optimization problem. Moreover, we are interested in utilizing the supervised component to sample negative actions for the training phase of the RL head. Finally, exploring shared base model and the supervised head to select candidate actions to address the “extrapolation error” problem [4] of off-policy RL may also be a promising direction.

Acknowledgements. The funding for Xin Xin was supported by the Natural Science Foundation of China (62106105, 62102234, 62072279, 61902219, 61972234), the National Key R&D Program of China (2020YFB1406704), the Key Scientific and Technological Innovation Program of Shandong Province (2019JZZY010129), the Natural Science Foundation of Shandong Province (ZR2021QF129), and the Fundamental Research Funds of Shandong University.

REFERENCES

[1] 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.

[2] 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.

[3] 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*.

[4] Scott Fujimoto, David Meger, and Doina Precup. 2019. Off-policy deep reinforcement learning without exploration. In *International Conference on Machine Learning*. 2052–2062.

[5] 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).

[6] 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.

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

[8] 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.

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

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

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

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

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

[14] Yifan Hu, Yehuda Koren, and Chris Volinsky. 2008. Collaborative filtering for implicit feedback datasets. In *2008 Eighth IEEE International Conference on Data Mining*. Ieee, 263–272.

[15] 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).

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

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

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

[19] Jens Kober, J Andrew Bagnell, and Jan Peters. 2013. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research* 32, 11 (2013), 1238–1274.

[20] 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.

[21] Xiao Lin, Hongjie Chen, Changhua Pei, Fei Sun, Xuanji Xiao, Hanxiao Sun, Yongfeng Zhang, Wenwu Ou, and Peng Jiang. 2019. A pareto-efficient algorithm for multiple objective optimization in e-commerce recommendation. In *Proceedings of the 13th ACM Conference on Recommender Systems*. 20–28.

[22] Yougang Lyu, Zihan Wang, Zhaochun Ren, Pengjie Ren, Zhumin Chen, Xiaozhong Liu, Yujun Li, Hongsong Li, and Hongye Song. 2022. Improving legal judgment prediction through reinforced criminal element extraction. *Information Processing & Management* 59, 1 (2022), 102780.

[23] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* (2013).

[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.

[25] 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.

[26] 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).

[27] Siddharth Reddy, Anca D Dragan, and Sergey Levine. 2019. SQIL: Imitation learning via reinforcement learning with sparse rewards. *arXiv preprint arXiv:1905.11108* (2019).

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

[29] 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.

[30] 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.

[31] Marco Tulio Ribeiro, Nivio Ziviani, Edleno Silva De Moura, Itamar Hata, Anisio Lacerda, and Adriano Veloso. 2014. Multiobjective pareto-efficient approaches for recommender systems. *ACM Transactions on Intelligent Systems and Technology (TIST)* 5, 4 (2014), 1–20.

[32] 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).

[33] 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.

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

[35] 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.

[36] 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.

[37] Aravind Srinivas, Michael Laskin, and Pieter Abbeel. 2020. Curl: Contrastive unsupervised representations for reinforcement learning. *arXiv preprint arXiv:2004.04136* (2020).

[38] Alex Strehl, John Langford, Lihong Li, and Sham M. Kakade. 2010. Learning from Logged Implicit Exploration Data. In *NIPS*.

[39] 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.

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

[41] 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.

[42] Nikos Vlassis, Aurélien Bibaut, Maria Dimakopoulou, and Tony Jebara. 2019. On the Design of Estimators for Bandit Off-Policy Evaluation. In *ICML*.

[43] Xin Xin, Alexandros Karatzoglou, Ioannis Arapakis, and Joemon M Jose. 2020. Self-Supervised Reinforcement Learning for Recommender Systems. *SIGIR* (2020).

[44] Xin Xin, Fajie Yuan, Xiangnan He, and Joemon M. Jose. 2018. Batch IS NOT Heavy: Learning Word Representations From All Samples. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Melbourne, Australia, 1853–1862. https://doi.org/10.18653/v1/P18-1172

[45] 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.

[46] 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.

[47] 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).

A APPENDIX

A.1 Training Procedure of SNQN

Algorithm 1 describes the training detail of SNQN.

Algorithm 1 Training procedure of SNQN

```
Input: user-item interaction sequence set $\mathcal{X}$, recommendation
    model $G(\cdot)$, reinforcement head $Q(\cdot)$, supervised head $f(\cdot)$,
    pre-defined reward function $r(\mathbf{s}, a)$
Output: all parameters in the learning space $\Theta$
Initialize all trainable parameters
Create $G'(\cdot)$ and $Q'(\cdot)$ as copies of $G(\cdot)$ and $Q(\cdot)$, respectively
repeat
    Draw a mini-batch of $(x_{1:t}, a_t^+)$ from $\mathcal{X}$
    Draw negative actions set $N_t$ for $x_{1:t}$
    $\mathbf{s}_t = G(x_{1:t})$, $\mathbf{s}'_t = G'(x_{1:t})$
    $\mathbf{s}_{t+1} = G(x_{1:t+1})$, $\mathbf{s}'_{t+1} = G'(x_{1:t+1})$
    Generate random variable $z \in (0, 1)$ uniformly
    if $z \leq 0.5$ then
        $a_*^+ = \text{argmax}_a\ Q(\mathbf{s}_{t+1}, a)$, $a_*^- = \text{argmax}_a\ Q(\mathbf{s}_t, a)$
        $L_p = (r(\mathbf{s}_t, a_t^+) + \gamma Q'(\mathbf{s}'_{t+1}, a_*^+) - Q(\mathbf{s}_t, a_t^+))^2$
        $L_n = \sum_{a_t^- \in N_t} (r(\mathbf{s}_t, a_t^-) + \gamma Q'(\mathbf{s}'_t, a_*^-) - Q(\mathbf{s}_t, a_t^-))^2$
        Calculate $L_s$ and $L_{snqn} = L_s + L_p + L_n$
        Perform updates by $\nabla_\Theta L_{snqn}$
    else
        $a_*^+ = \text{argmax}_a\ Q'(\mathbf{s}_{t+1}, a)$, $a_*^- = \text{argmax}_a\ Q'(\mathbf{s}_t, a)$
        $L_p = (r(\mathbf{s}_t, a_t^+) + \gamma Q(\mathbf{s}_{t+1}, a_*^+) - Q'(\mathbf{s}'_t, a_t^+))^2$
        $L_n = \sum_{a_t^- \in N_t} (r(\mathbf{s}_t, a_t^-) + \gamma Q(\mathbf{s}_t, a_*^-) - Q'(\mathbf{s}'_t, a_t^-))^2$
        Calculate $L_s$ and $L_{snqn} = L_s + L_p + L_n$
        Perform updates by $\nabla_\Theta L_{snqn}$
    end if
until converge
return all parameters in $\Theta$
```

A.2 Datasets Description

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 sessions whose length is smaller than 3 and then sample a subset of 200k sessions.

RetailRocket. This dataset is collected from a real-world e-commerce website. It contains session 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. Table 5 summarizes the statistics of the two datasets.

Table 5: 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

A.3 Baselines Description

We integrated the proposed SNQN and SA2C with four state-of-the-art sequential recommendation models:

- GRU [9]: 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.
- Caser [39]: This is a recently proposed CNN-based method, which captures sequential signals by applying convolution operations on the embedding matrix of previous items.
- NItNet [45]: NItNet uses dilated CNN for larger receptive field and residual connection to increase network depth.
- SASRec [17]: This baseline is based on self-attention and uses the Transformer [41] architecture. The output of the Transformer encoder is treated as the latent sequence state.

To further demonstrate the effectiveness of the proposed methods, we also compare SNQN, SA2C with SQN, SAC[43], respectively.

A.4 Parameter settings

For both datasets, the input sequences are composed of 10 interacted items. If the sequence length is less than 10, we complement the sequence with a padding item. We train all models with the Adam optimizer [18]. The mini-batch size is set as 256. For SNQN, the learning rate is set as 0.01 on RC15 and 0.005 on RetailRocket, which is the same setting with SQN [43]. For SA2C, we use the same learning rate with SNQN at the early pre-training stage. After that, the learning rate is set as 0.001 on both datasets. For a fair comparison, we use the basic uniform distribution for the negative sampling strategy of RL to eliminate influence from the sampler. The item embedding size is set as 64 for all models. For GRU, 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 use the published implementation [45] with the predefined settings. For SASRec, the number of heads in self-attention is set as 1, according to the original paper [17]. Note that, when SNQN and SA2C are integrated with a base model, the hyper-parameter setting of the base model remains exactly unchanged, for a fair comparison.

For the training of SNQN and SA2C, the discount factor γ is set as 0.5. The ratio between the click reward (r_c) and the purchase reward (r_p) is set as $r_p/r_c = 5$. These settings are the same as in [43] for a fair comparison. If without special mention, for one positive action we sample 10 negative actions in the training procedure. The reward for negative actions is set as $r_n = 0$.

A.5 Performance Comparison on RetailRocket

Table 6 shows the performance of top-k recommendations on the RetailRocket dataset.

Table 6: Top-k recommendation performance comparison of different models ($k = 5, 10, 20$) on RetailRocket. Recommendations are generated from the supervised head. NG is short for NDCG. Boldface denotes the highest score.

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-SNQN	0.5232	0.4376	0.5713	0.4544	0.6175	0.4650	0.2662	0.2065	0.3181	0.2233	**0.3656**	0.2353
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
GRU-SA2C	**0.5526**	**0.4754**	**0.5963**	**0.4897**	**0.6313**	**0.4985**	**0.2720**	**0.2150**	**0.3208**	**0.2308**	**0.3656**	**0.2422**
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-SNQN	0.3757	0.3179	0.4181	0.3317	0.4595	0.3422	0.2160	0.1721	0.2530	0.1841	0.2895	0.1934
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
Caser-SA2C	**0.3971**	**0.3446**	**0.4381**	**0.3578**	0.4733	**0.3667**	0.2170	**0.1759**	0.2528	**0.1875**	0.2873	**0.1963**
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-SNQN	0.6016	0.5062	0.6543	0.5234	**0.6921**	0.5330	0.2699	0.2065	0.3236	0.2240	0.3703	0.2358
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
NItNet-SA2C	**0.6226**	**0.5422**	**0.6573**	**0.5534**	0.6842	**0.5603**	**0.2787**	**0.2197**	**0.3271**	**0.2354**	**0.3719**	**0.2468**
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-SNQN	0.5776	0.4846	0.6310	0.5020	0.6719	0.5123	0.2815	0.2171	0.3381	0.2355	0.3888	0.2483
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
SASRec-SA2C	**0.5929**	**0.5080**	**0.6437**	**0.5246**	**0.6798**	**0.5337**	**0.2873**	**0.2242**	**0.3409**	**0.2416**	**0.3893**	**0.2538**