Master's Thesis of Data Science

Efficient Exploration in Reinforcement Learning for Online Slate Recommender System

강화학습 기반 온라인 슬레이트
추천 시스템에서의 효율적 탐색 방법

February 2023

Graduate School of Data Science
Seoul National University
Data Science Major

Seung Joon Park

Efficient Exploration in Reinforcement Learning for Online Slate Recommender System

Adviser Min-hwan Oh

Submitting a master's thesis of
Data Science

December 2022

Graduate School of Data Science
Seoul National University
Data Science Major

Seung Joon Park

Confirming the master's thesis written by
Seung Joon Park
January 2023

Chair	Wen-Syan Li	(Seal)
Vice Chair	Min-hwan Oh	(Seal)
Examiner	Hyung-Sin Kim	(Seal)

Abstract

Deep reinforcement learning (RL) is a promising approach for recommender systems, of which the ultimate goal is to maximize the long-term user value. However, practical exploration strategies for real-world applications have not been addressed. We propose an efficient exploration strategy for deep RL-based recommendation, RESR. We develop a latent state learning scheme and an off-policy learning objective with randomized Q-values to foster efficient learning. Online simulation experiments conducted with synthetic and real-world data validate the effectiveness of our method.

Keyword: Deep RL, Recommender System, Exploration, Simulation, POMDP

Student Number: 2021-28044

Table of Contents

Chapter 1

INTRODUCTION

Online recommender systems are widely applied in various domains where the system repeatedly interacts with users in real-time and receives user feedback. An *online* recommender system adapts to the input received, learns "on-the-go," and improves its recommendations as data is gathered.[1]

Bandit algorithms are one of the long-studied online algorithms applied to web recommendations. Multi-armed bandits [1, 2] and contextual bandits [3, 4] have been widely used in various recommendation problem settings. These bandit approaches are known to achieve superior performances in terms of immediate rewards (or other desired user behavior, such as click-through rates). However, there are often problem settings where the recommender needs to plan for the outcome following a recommended *action*. For example, a streaming service user may initially be attracted by the provocative "clickbait" content but soon leave the service disappointed. As a result of the immediate reward received when a click is made on clickbait content, a bandit-based recommender will continue to recommend sensational content to the user. A method that can look ahead more than a single-step response would be beneficial in

[1] In contrast, *batch* (or offline) recommender systems are applied to a static dataset and cannot adapt their recommendations over time. Typical methods used in batch recommender systems include collaborative filtering, content-based filtering, etc. To keep batch recommender systems up-to-date, they need periodic retraining, which is computationally intensive and time-consuming, especially when the dataset is large.

such settings. Hence, we consider an RL-based recommendation in this work.

In a recent work [5], a slate-based recommendation using deep RL was proposed. In the setting, multiple items are presented to users at once. The list of items recommended is called a *slate*. This setup is arguably one of the most common types of interactions in mobile or web applications. Then, the recommender aims to find the best slate to be recommended to users whose states are known to the recommender in Ie et al. [5]. Extending this slate-based recommendation problem setting, we incorporate users' latent states that affect the feedback on the recommended items. Furthermore, the previously proposed method by Ie et al. [5] leaves the exploration strategy unmentioned. Exploration involves taking actions that are not necessarily maximizing the expected reward. Since information about the environment and the reward function is unknown to the agent, it is crucial to improve the policy with a limited number of trials quickly. This process is referred to as efficient exploration, where an RL agent learns the optimal policy using a small number of data samples collected from the environment. The previous works perform a naive exploration strategy of dithering (e.g., ϵ-greedy policy) that can lead to inefficient learning in terms of sample complexity.

For this problem, we propose RESR, **R**andomized **E**xploration for **S**late-based **R**ecommendation, a method considering efficient exploration and user latent state learning. Efficient exploration is an essential factor contributing to a better recommendation system in practice. Efficient exploration leads to finding a good policy in a proper amount of time which is crucial for the recommender system. If the algorithm requires high sample complexity, it will lose many users when the model is under-trained. In addition, the data-collecting process is not free. Efficient exploration would reduce the cost for the policy to achieve high performance.

Practical issues when applying the algorithm are also considered. We suggest learning a user's latent state from sequential data such as past click history. Moreover, we adopt and train a multinomial choice model [6] to calculate the probability of a user selecting an item. In the simulated experiments, we show that RESR outperforms

the other RL-based off-policy algorithm SlateQ [5]. Our algorithm exhibits stable learning while considering multiple learning objectives. We evaluate performance using both the traditional click-through rate (CTR) perspective and reward, which is a better metric for measuring the long-term value [5] of a user.

In summary, the main contributions of this work are:

- We formulate a sequential decision-making problem of a slate-based recommender system as a partially observable Markov decision process (POMDP) using users' latent states. This modeling approach is a generalization of the existing setup of Ie et al. [5].
- We propose RESR, an RL algorithm that enables efficient exploration for the slate recommendation problem. RESR utilizes randomized value functions to approximate the posterior distribution through sampling, which leads to efficient learning by balancing exploitation and exploration.
- Our proposed method shows superior performances compared to the baseline in various environment settings. The performances are measured with respect to the sample complexity, CTR, and user retention.

Chapter 2

RELATED WORKS

RL-based Recommender System

The process of recommending items to users and receiving feedback is a sequential decision problem [7]. Conventional approaches have dealt with the recommendation problem as a prediction problem, maintaining a static view of the world that a user's interest would remain the same over time [8]. In a dynamically changing world, however, the data from the past may no longer be relevant. In contrast to the traditional static recommendation process, interactive recommender systems (IRS) refers to the process where user feedback is provided to refine the system continuously [9, 10]. Previously, RL has shown success in games [11, 12] where long-term planning is required from the interaction between the agent and the environment. Similarly, in IRS, learning from feedback and capturing user transition via RL have been suggested [13, 14, 15]. An RL-based recommender system aims to maximize each user's long-term satisfaction with the system [16, 5]. The reward can be formulated to promote desirable user behavior, such as user engagement (longer watch time, clicks, etc.). A policy gradient-based algorithm, REINFORCE [17, 16], and the value-based algorithms [18, 5] have both been shown to increase long-term user engagement in live experiments at YouTube. Previous research mainly focused on dealing with extremely large state and action spaces. This practical issue arises from having millions of items to recommend and

becomes even worse when considering a list of items (slates) to recommend. This paper focuses on the well-known *exploration vs. exploitation dilemma* that has not been addressed much in the RL-based recommender system. Exploration is taking an action that has not been taken before and exploitation is taking the best-known action. We provide more sophisticated means to explore in contrast to taking random actions. We suggest an efficient exploration method for the RL-based recommender system utilizing randomized Q-functions.

Value-based Deep RL

Deep RL is a combination of deep learning and RL [19]. It utilizes the strength of deep neural networks to generalize, even with a high number of parameters [20]. RL has gained popularity for its success in addressing challenging tasks, most notably in games [11, 12]. DQN is a model-free value-based algorithm that has been proposed by [11] which has been successful in achieving superhuman level control in several ATARI games. The input here is the pixels, and neural networks are used as function approximators for Q-values. The value-based RL algorithms aim to build a value function of states. The value functions estimate how good it is for the agent to be in a given state [21]. One of the most popular value-based algorithms is the Q-learning [22] which keeps lookup tables for value functions of every state-action pair denoted $Q(s, a)$. Bellman equation is used to update the $Q(s, a)$ as rewards are received and new states are visited. DQN utilizes deep neural networks and stochastic gradient descent to update $Q(s, a)$. Experience replay is used to store the trajectory of the Markov Decision Process and mini-batch samples are used to update the Q-function. Neural networks give generalization power of computing $Q(s, a)$ compared to keeping a lookup table of discrete state-action pairs. The generalization is related to the exploration of intractably large state-action spaces. When applying Deep RL to recommender systems, efficient exploration remains a challenging problem that we would like to solve in this paper.

Chapter 3

PROBLEM STATEMENT

We formulate our problem of recommending a slate consisting of arbitrarily chosen K number of items. We have a set of items $[N] \coloneqq \{1, \ldots, N\}$. There exist N items (i.e., documents, videos, etc.) that are available for recommendation. Our task is to recommend a slate of K ($K < N$) items to the user. The recommender is formed as an RL agent that interacts with its environment in episodes, resulting sequence of observations, actions, and rewards. We use o_t, a_t, and r_t to denote the observation, the action, and the reward received at timestep t. The action a_t is the list of K items recommended to the user for selection at t. We model the user to have a D-dimensional latent state vector s_t at timestep t which evolves as a user clicks a new item. A sequential recommendation task is generally formulated as a *Markov Decision Process* (MDP), in which the next state is only dependent on the previous state and the action taken. In reality, the recommender cannot fully observe the true user state. To address this issue, the problem must be formulated as a *Partially Observable Markov Decision Process* (POMDP) which consists of a 6-tuple $(\mathcal{S}, \mathcal{A}, P, R, \Omega, \mathcal{O})$ where

- $\mathcal{S}$: set of user's latent states, which are possibly continuous.
- $\mathbb{P}$: transition dynamics $\mathbb{P}(s'|s, a) : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ which is the conditional probability that the user transitions to state s' when action a is taken at user

state s.

- R: reward function $R : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ which maps state-action pair to a real-valued scalar.
- Ω: set of observations of the user history.
- $\mathcal{O}$: observation dynamics $\mathcal{O}(o|s', a) : \mathcal{S} \times \mathcal{A} \times \Omega \rightarrow [0, 1]$ which is the conditional probability that the agent receives $o \in \Omega$ when the next state is s' after the action a has been taken.

The exact solution to POMDPs can be computationally intractable. One of the main challenges is the estimation of the user's state which is unknown. We use a pseudo-state, $\hat{s}$, which is an approximation of the true user state. We aim to find an approximate solution to the POMDP problem using a pseudo-state. Let $\hat{\mathcal{S}}$ be the set of pseudo-states. The goal is to learn a policy, $\pi : \hat{\mathcal{S}} \rightarrow \mathcal{A}$, that maps the current pseudo-state to a recommended action for the user. In RESR, the policy is learned by a combination of model-free off-policy Q-learning and model-based user choice model. At each timestep t, the agent chooses an action a_t based on a policy π that is learned throughout the episodes.

We try to capture latent user representation s_t from a sequence of w recent observations. A sequence of w recent observations $o_t = \{(a_{t-w}, c_{t-w}), (a_{t-w+1}, c_{t-w+1}), \ldots, (a_{t-1}, c_{t-1})\}$ is given to the agent at time t where c_j is the additional user information including the user response (e.g., clicked item, time watched, etc.) when the slate a_j was given to the user at time j. We process sequential interaction data to learn user representations $\hat{s}$ of arbitrary dimensions. The goal is to learn a policy that maximizes the expected cumulative reward. The latent representation is learned along with the policy with the goal of maximizing the expected cumulative reward. This approach is more general compared to Ie et al. [5] as we account for the latent states of users.

Intractability also arises from large action space, as we have to construct a slate of K items. In order to relieve the combinatorial action space problem, we make the fol-

lowing mild assumptions introduced by Ie et al. [5]: 1) a user selects one item from the slate (including no-choice option); 2) The reward r and the state transition $\mathbb{P}(s'|s,a)$ depends only on the item $i \in a$ selected by the user. Under these assumptions, the state-action value of a user given the combinatorial $\binom{|N|}{K}K!$ possible slates can now be represented by summing K tractable item-wise action-value functions. This method will be discussed more thoroughly in Section 4.1.

Chapter 4

METHOD

4.1 Tractable Decomposition of Action Space

In order to relieve combinatorial action space, the decomposition of slate Q-values into tractable item-wise Q-values is used. This approach has been introduced in `SlateQ` [5], a value-based model-free RL algorithm for recommender systems. `SlateQ` addresses the problem of combinatorial action space in selecting slates. Selecting K items to recommend to the user results in a combinatorial action space of $\binom{|N|}{K}K!$. Choosing a slate is intractable to solve when $|N|$, the size of the item set, is large. Therefore, we utilize `SlateQ` which is a method of decomposing the value function of intractable actions space into a sum of tractable functions of slate's components. Thus, only item-wise Q-function q is learned to estimate $Q(s, a)$ which is the action value of recommending slate a to a user with latent state s. The equation to represent $Q(s, a)$ using the item-wise q function is

$$Q(s, a) = \sum_{i \in a} p(i|s, a)q(s, i),$$

where $q(s, i)$ represents the expected future rewards of choosing an item i for a user with state s and following the optimal policy afterward, and $p(i|s, a)$ is the proba-

bility of a user with state s clicking the item when slate a is given. Throughout the experiment, `SlateQ` serves as the baseline for comparison with our method.

4.2 Latent User Representation

The true states of users are not observed or known to the agent. Hence, we propose a learning method for capturing latent user representation. We denote the latent state representation function as ϕ which can be implemented in any method that processes sequential data, such as RNNs and transformers [23]. In the experiments, we learn the latent representation of a user using an LSTM [24] layer which takes sequential data as input to predict the user latent vector $\hat{s}$ of predetermined dimension, D. If there exists any side information about users (e.g., demographics), it may also be added to the input. The true user latent state at timestep t, s_t, is estimated using data o_t.

$$s_t \approx \hat{s}_t := \phi(o_t; \psi), \tag{4.1}$$

where ψ denotes the parameters of function ϕ.

4.3 User Choice Model

The user choice model determines the selection of an item from the slate recommended by the recommender system. We model the user's choice by the multinomial logit model [6]. The scores are evaluated based on the similarity between the user's latent state, s, and the feature vectors $x_1, x_2, \ldots, x_K$ of K recommended items. A user may also choose not to select any item. The probability of not choosing any item on a slate is denoted by $p(0|s, a)$. Therefore, the user's probability of selecting an item i given

user's state s and slate a, $p(i|s,a)$, is defined as

$$p(i|s,a) := \begin{cases} \dfrac{\rho}{\rho + \sum_{j \in a} u(s,x_j)} & \text{if } i = 0 \\ \dfrac{u(s,x_i)}{\rho + \sum_{j \in a} u(s,x_j)} & \text{if } i \in a \\ 0 & \text{otherwise} \end{cases}$$

where $u(\cdot,\cdot)$ is a non-negative utility score and ρ represents a base score for not choosing any item on the slate. As the true user choice model is not known to the agent, we train an affinity function g that returns the affinity score of a user and an item. Likewise, pseudo-state $\hat{s}$ is used in replacement of true s. Thus, the random utility function u is modeled as $u(s,x_i) \approx \exp\left(g(\hat{s},x_i;\beta)\right)$ where β is the parameter of the affinity function. It is more likely that the item will be chosen if the affinity score is high. The probability of choosing item i when latent user state estimate $\hat{s}$ and slate a is given is denoted by $p(i|\hat{s},a;\beta)$.

$$p(i|\hat{s},a;\beta) := \begin{cases} \dfrac{1}{1 + \sum_{j \in a} \exp\left(g(\hat{s},x_j;\beta)\right)} & \text{if } i = 0 \\ \dfrac{\exp\left(g(\hat{s},x_i;\beta)\right)}{1 + \sum_{j \in a} \exp\left(g(\hat{s},x_j;\beta)\right)} & \text{if } i \in a \\ 0 & \text{otherwise} \end{cases} \tag{4.2}$$

When fitting g, we have set $\rho = 1$ as the affinity score of an outside option of not choosing any item. During the learning process, the value of ρ does not matter as scores of the other items will be re-scaled to match that of the true distribution. Let $\mathcal{D}$ denote a dataset that consists of $|\mathcal{D}|$ samples. The loss over $\mathcal{D}$ can be computed using the user choice response variable denoted $y_\ell \in \{0,1\}^{K+1}$. The user choice response variable is the one-hot encoded response of the user's choice where $y_{\ell i}$ is the i-th element of y_ℓ. The zeroth element $y_{\ell 0}$ represents the no-choice option. The cross-entropy loss, $\mathcal{L}^{\text{CE}}$,

is then

$$\mathcal{L}^{\mathrm{CE}}(\psi, \beta) := -\frac{1}{|\mathcal{D}|} \sum_{\ell \in \mathcal{D}} \sum_{i \in a \cup \{0\}} y_{\ell i} \log p(i|\hat{s}_\ell, a_\ell; \beta).$$

Note ψ is the parameter of the latent state representation function in Eq.(4.1).

Finally, we set our learning objective for the Q-function. The loss function uses squared error loss between the predicted Q-values and the target Q-values as in Deep Q-Network (DQN) [25] algorithm. The Q-function is parametrized by θ. The parameter for the online Q-network is denoted θ, and the parameter for the target is denoted θ'. Target Q-value is calculated by summing sampled reward r_t with the discounted sum of future rewards. This can be written as $r_t + \gamma \max_{a'} Q(s', a'; \theta')$. The target Q can be represented using decomposed item-wise q. Thus, the DQN loss $\mathcal{L}^{\mathrm{RL}}$ computed to update the parameter is

$$\mathcal{L}^{\mathrm{RL}}(\psi, \theta) := \frac{1}{|\mathcal{D}|} \sum_{\ell \in \mathcal{D}} \left[(r_\ell + \gamma \max_{a'} \sum_{j \in a'} p(j|\hat{s}'_\ell, a') q(\hat{s}'_\ell, x_j; \theta') - q(\hat{s}_\ell, x_i; \theta) \right]^2,$$

where r_ℓ is the reward received, $\hat{s}_\ell$ is the current pseudo-state of user, and $\hat{s}'_\ell$ is the next pseudo-state of user for the ℓ-th sample. x_i is the feature vector of item i consumed by the user for the ℓ-th sample. The item-wise Q-network takes the pseudo-state of a user, $\hat{s}$, and the feature of the item i, x_i, to compute $q(\hat{s}, x_i)$. In the experiments, we used a 3-layer neural network for the item-wise Q-function. The target network $q(s', a'; \theta')$ is used to compute the next state-action value for added stability. The target is updated periodically using the hard update. In the experiments, the parameters of the target network were replaced by those of the online network every 4,000 steps. As we have used B randomized Q-functions as explained in Section 4.4, the final $\mathcal{L}^{\mathrm{RL}}$ is the mean of all losses computed by B number of Q-functions.

4.4 Exploration via Randomized Q-Functions

Our method utilizes randomized Q-functions [26] for efficient exploration. Osband et al. [27] proposed *bootstrapped DQN* which is an algorithm that performs deep exploration without any dithering strategy such as ϵ-greedy. Exploration is done in ϵ-greedy by taking a random action with probability ϵ at each step. Dithering strategies can lead to inefficient learning, as randomly chosen actions may lead to failure. Moreover, the ϵ-greedy strategy introduces a hyperparameter which complicates the learning. The exploration rate, ϵ, is the hyperparameter in the ϵ-greedy policy. The common heuristic is to decrease ϵ slowly over time, starting from a large number. Randomized Q-functions are used to approximately sample from the posterior distribution of the true Q-function. Sampling one Q-function is similar to *Thompson Sampling* [28, 29] where we draw parameters from a posterior distribution. In order to approximate posterior using sample distribution, each Q-function is trained on a sub-sample of the data. The data samples for each timestep t are stored in a replay buffer with a bootstrap mask e_t that indicates which Q-network to train on.

Let B ($B \geq K$) be the number of item-wise Q-functions. We develop the idea of bootstrapped DQN, which samples one Q-function out of multiple functions per episode. For each episode, `RESR` samples K number of item-wise Q-functions out of B functions instead of one. We sample K number of Q-functions to determine the slate to be recommended. Sampling K value functions may reduce the risk of sampling a Q-function with bad estimates. Relying solely on one function's value estimates may lead to suboptimal decisions. We can make use of multiple value estimates in slate recommendations to avoid this situation. Also, sampling K instead of one Q-function enables a simple heuristic when building slates that maximize the expected cumulative return. `SlateQ` [5] needs slate optimization to construct a slate that gives maximum

expected values considering $p(i|s,a)$. The original optimization of choosing a slate

$$\underset{\substack{a\subseteq[N]\\|a|=K}}{\text{maximize}} \sum_{i\in a} p(i|s,a)q(s,i),$$

can be solved in polynomial time by the linear program (LP) below [30].

$$\text{maximize} \sum_{i\in[N]} \frac{z_i u(s,i) q(s,i)}{\rho + \sum_j z_j u(s,j)}$$
$$\text{subject to} \sum_{i\in[N]} z_i = K; 0 \le z_i \le 1, \text{for all } i \in [N],$$

where $u(s,i)$ is the unnormalized affinity score between the user with state s and the item i ($e^{u(s,x_i)}$), q is the decomposed item-wise Q-values of the items within the slate, and $z_i \in \{0,1\}$ is a binary variable for each item i indicating whether i is the constituent of slate a.

Using K sampled Q-functions enables a simple heuristic for building a slate of K items. For each slot of a slate, we choose the item with the maximum weighted Q-value $u(s,i)q(s,i)$ using the corresponding sampled item-wise Q-function. To prevent recommending the same item more than once, we do this sequentially for each Q-function. We only choose an item not included in the previous slot(s). The complexity of this heuristic is $O(K\log(N))$ on average, while that of the original LP is $O(N)$. Since $K \ll N$, the heuristic takes much less time to build a recommendation slate. This heuristic is especially useful when the number of items for recommendation is large. We implemented this heuristic for training (calculating target Q-value) and serving (action selection) in the real-world data experiment setting where $N = 1,682$. We provide two ways to construct a slate for a user with pseudo-state $\hat{s}_t$ at timestep t.

First, we can choose a slate with the highest $Q(\hat{s},a)$ estimate using K sampled value functions. We compute $Q(\hat{s}_t,a) = \sum_{k=1}^{K} p(i_k|\hat{s}_t,a)q_k(\hat{s}_t,i_k)$ and pick a slate $a_t \in \arg\max_a Q(\hat{s}_t,a)$.

The second method uses the greedy heuristics aforementioned to construct a slate. Each sampled item-wise Q-function is assigned to a slot of a slate. We place an item with the highest item-wise Q-value of the corresponding Q-function. We need to exclude picking the same item if it has already been chosen for the previous slot(s). The steps for the greedy slate construction can be written as

1. Initialize $a_t = \emptyset$
2. Repeat the following K times: $a_t \leftarrow a_t \cup \underset{i \in [N] \setminus a_t}{\arg\max}(g(\hat{s}_t, x_i) q_k(\hat{s}_t, i))$

In an effort to diversify Q-functions, we use the cosine loss function to increase the dissimilarity of outputs between two similar Q-function networks. Diversifying Q-functions may prevent value functions from collapsing. The "collapse" refers to a situation where all random Q-functions converge to the same function resulting in a loss of diversity, potentially leading to suboptimal performance. However, diversifying Q-functions may not always be suitable in terms of performance. If the convergence is done correctly to the optimal value function, diversification of the Q-functions may induce a suboptimal estimate of the expected long-term reward. We could prevent this situation by decreasing the degree of diversification. In the experiments, we only provide results where the tuning parameter of diversification loss is fixed. Further investigations can be conducted where the parameter slowly decreases over time.

The diversification loss is computed for each K sampled item-wise Q-function q_k and its most similar counterpart q_b. We denote $\mathcal{L}^{\text{div}}$ for the diversification loss computed using Q-function diversification.

Now, our full learning objective is:

$$\mathcal{L}^{\text{total}} = \mathcal{L}^{\text{RL}} + \alpha^{\text{CE}} \mathcal{L}^{\text{CE}} + \alpha^{\text{div}} \mathcal{L}^{\text{div}}, \tag{4.3}$$

where α^{CE} and α^{div} correspond to the tuning parameters for cross-entropy loss and diversification loss respectively. We compare the method with this diversification and without the diversification. α^{div} is set to 0 for the method without diversification.

Algorithm 1 Q-function Diversification Loss

Set $L = 0$
for $\ell = 1, \ldots, |\mathcal{D}|$ **do**
  Compute $q_b(\hat{s}_\ell, \cdot)$ for all $b \in \mathbb{B} = \{1, \ldots, B\}$
  for each sampled function index k **do**
    $L = L + \max_{b \in \mathbb{B} \setminus k} \cos(q_k(\hat{s}_\ell, \cdot), q_b(\hat{s}_\ell, \cdot))$
  end for
end for
$\mathcal{L}^{\text{div}} \leftarrow \frac{1}{|\mathcal{D}|} L$

Incorporating all the methods above, we derive an algorithm for RESR. Before we introduce the algorithm, let an oracle $\mathcal{O}_{[N]}$ be accessible by the agent. The oracle returns a set of available actions constructed using K items in $[N]$. We denote $\mathcal{O}_{[N]}(s_t)$ to represent a set of available slates for user state s_t. In practice, a_t can be chosen by either applying $\arg\max$ operator over actions to $Q(\hat{s}_t, a)$ or using the greedy slate construction heuristic. This process is represented by the policy $\pi(\hat{s}_t)$ in the algorithm. The full version of the algorithm using $\arg\max$ operator for slate construction is shown in Appendix 3.

Algorithm 2 RESR

Input: Current user state s_t. Observation of user, o_t, with window size w.
for each episode **do**
  Sample K item-wise Q-functions from $\{q_b$ for all $b \in \mathbb{B}\}$
  for each timestep t **do**
    Compute pseudo-state of user $\hat{s}_t = \phi(o_t)$
    Choose action using sampled value functions $a_t \leftarrow \pi(\hat{s}_t) \in \mathcal{O}_{[N]}(s_t)$
    Observe user response c_t and receive reward r_t
    Create next observation $o_{t+1} = \{(a_{t-w+1}, c_{t-w+1}), \ldots, (a_t, c_t)\}$
    Sample bootstrap mask $e_t \sim M$
    Add (o_t, r_t, o_{t+1}, e_t) to buffer
  end for
end for

The structure of RESR is presented in Figure 4.1. The user interaction history is given as input to ϕ which returns the pseudo-state $\hat{s}_t$. It is then fed into the user choice model and Q-network to make recommendations. The Q-network and the affinity func-

tion are trained regularly using the sampled batch from the replay buffer.

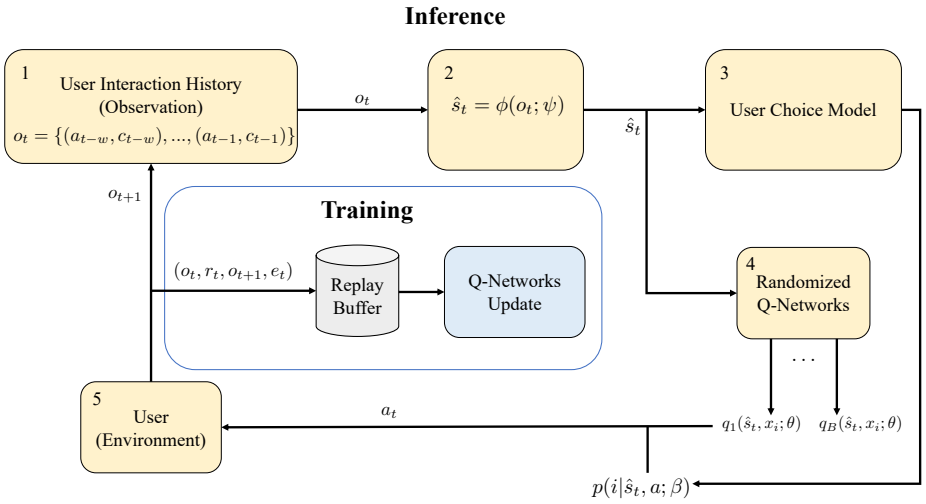


Figure 4.1: Schematic view of RESR

Chapter 5

EXPERIMENTS

5.1 Online Simulation Environment

We present two scenarios to evaluate the performance of our method. One is the fully simulated environment for item recommendation, and the other is the movie recommendation scenario reflecting some aspects of the real world. The first scenario was motivated by Mladenov et al. [31]. They modeled the user to have a slowly evolving state under partial observability. Temporal abstraction [32] is used to overcome the error introduced by state estimation. The second scenario uses embedding vectors of users and items extracted from a real-world dataset. In our experiments, we empirically show the effectiveness of our exploration strategy. We utilized Recsim NG [33], a platform for simulating various recommender systems. We test our RL-based recommender system that interacts with the user. The specification of the two scenarios is as below.

5.2 User Arrival and Departure

This section explains the macro-level settings of the environment that determine how a user enters and leaves the system. Both scenarios use Poisson distribution to determine

how many users enter the system at timestep t. The rate λ_t of the Poisson distribution is defined as $\lambda_t = \text{clicks}_{t-1} \times \tau$ where $clicks_{t-1}$ is the total number of clicks incurred during timestep $t-1$ and τ is a constant parameter unknown to the agent. User departure for arbitrary user x who is already in the system happens at a probability p_{out_x} where $p_{\text{out}_x} = \frac{1}{1+\exp(-\text{fatigue}_x + b)}$ and fatigue_x is the current fatigue level of user x and b is the constant corresponding to the fatigue level making $p_{\text{out}_x} = 0.5$. Next, we discuss the micro-level environment settings that determine how a user within the system interacts with the RL agent (recommender).

5.3 Fully Simulated Recommendation

Two main agents interact in the experiment. One is the user who chooses to select or not to select one of the items from the slate recommended by the recommender. The user's choice is modeled by the multinomial logit model. The score is evaluated based on the similarity between the user state and the recommended items. The other is the recommender (or the RL agent) that selects a slate of K items to be presented to the user based on the policy learned from user feedback.

Item An item $i \in [N]$ has a stationary feature vector $x_i \in \mathbb{R}^D$ that is fixed throughout the episodes. Item feature x_i is sampled from a multivariate normal distribution $\mathcal{N}(\boldsymbol{\mu}_f, \Sigma_f)$ where $\boldsymbol{\mu}_f$ is determined by v_i, the topic of the item i. We assume the number of topics to match the feature dimension for simplicity.

Item i also has a scalar quality, κ_i, that is also dependent on the topic v_i. We decide κ_i be sampled from $\mathcal{N}(\mu_d, \sigma_d^2)$. Each topic has a mean value for the quality, μ_d, which is determined by the corresponding mean quality value of topic $d \in \{1, \ldots, D\}$. σ_d^2 represents the variance of the distribution. The quality of the item affects the user transition to the next state. This intuition is borrowed from that of Mladenov et al. [31]. Intuitively, a person would have an evolving interest that leads to consuming more if the item selected is of good quality and consuming less or choose not to select if the

item quality is not good. To put it in other words, a person who consumes good quality item has a state change in the direction of the item feature vector. Whereas a person who consumes bad quality item has a state change in the opposite direction of the item feature. This relation is represented in Eq. (5.1).

User We initialized the true user state to be sampled from a multivariate normal distribution $s_0 \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_D)$ where $\mathbf{0}$ is a zero vector and $\mathbf{I}_D$ is an identity matrix of size D. A user's latent state at time t, s_t, is determined based on the item chosen. The change in the previous user state s_{t-1} to s_t when item i is consumed is represented as

$$s_t = (1 + \delta)\{s_{t-1} + \eta\kappa_i(x_i - s_{t-1})\} \tag{5.1}$$

where κ_i corresponds to the *quality* scalar of item i and η is the sensitivity parameter. The magnitude scale parameter $\delta \neq -1$ affects the magnitude of the next state. The magnitude of the interest vector is increased/decreased depending on the quality κ_i (e.g., $\delta = 0.01$ if $\kappa_i > 0$ and $\delta = -0.01$ if $\kappa_i < 0$).

Since the true user state is unknown to the agent, an LSTM layer is used to output the user pseudo-state $\hat{s}_t$ from o_t with window size w. Layer norm is applied to the LSTM layer for added stability. In the experiments, we set the window size of the observation to 10.

5.4 Simulation using the Real-World Data

In the previous scenario, the items and the users had latent features extracted from some normal distribution. Random feature vectors may not reflect the true distribution of users and item latent features. We tried to close the gap between the simulation and reality by extracting these latent features using real-world data. MovieLens 100K dataset[1] is used to extract the latent feature vectors. The dataset contains 100,000 ratings from 943 users on 1,682 movie items. The embedding layers are trained using a

[1] https://grouplens.org/datasets/movielens/100k/

supervised learning objective and supervised contrastive learning [34]. We initialize the embedding layer using autoencoders for disentangled representation. Supervised contrastive learning is used to make movies of the same genre have similar vector representations. Finally, a Siamese neural network [35, 36] is used to encode the similarity between users and items (movies). The final dot product of user-item embeddings should match the similarity in the dataset. Below is the diagram of the feature extraction process and the resulting t-SNE plotting of items on two-dimensional space.

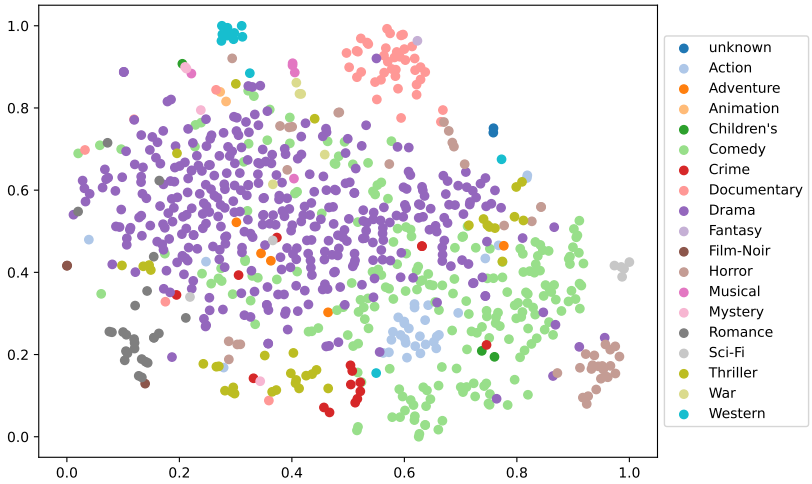


Figure 5.1: Extracted item features visualized using t-SNE

As we are given ratings that range from 1 to 5, we set the low ratings to represent low similarity and high ratings to represent high similarity. More details on the extraction of feature vectors are provided in Appendix A.2.2.

5.5 Results

5.5.1 Fully Simulated Recommendation

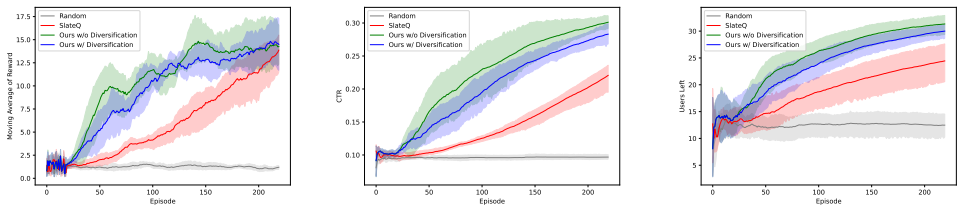
We experiment on different settings for the slate size $K = \{2, 3\}$. The number of items for recommendation $N = 100$. The number of users in the system is initially set to 4 and new user(s) enter every timestep according to the Poisson distribution with rate $\lambda_t = \text{clicks}_t \times \tau$. The τ is set to 0.5. Each user x has a probability of leaving the system with $p_{\text{out}_x} = \frac{1}{1+\exp(-\text{fatigue}_x + b)}$. In the experiments, we set $b = 4$. User transition occurs after a user chooses an item. The transition will occur depending on the item's quality, as shown in Eq. (5.1). The sensitivity parameter η is set to 0.1. We were able to reach a high-performing policy faster compared to the baseline `SlateQ` algorithm using ϵ-greedy for exploration. The total number of episodes per run is 220, each with 100 interaction steps. The value of ϵ is linearly decayed over the total number of iterations (22,000 steps) from 1.0 to 0.01. All policies have the agent acting randomly to collect data for the initial 20 episodes. The results were averaged over five seeds and rounded to the nearest hundredth. The detail of the experiment setting is in Appendix A.2.

In Table 5.1, the mean result of the experiment is provided for $K = 2$. Implementations of `RESR` with Q-function diversification is denoted `RESR` + Div., and `RESR` without Q-function diversification is denoted `RESR`. We can see that `RESR`s outperform `SlateQ`. Random policy, which corresponds to constructing a slate with randomly selected items, shows the worst performance for all metrics.

Policy	Reward	CTR	Users Left
Random	1.26	0.10	12.46
`SlateQ`	6.37	0.14	18.69
`RESR`	**10.86**	**0.22**	**24.96**
`RESR`+Div.	9.90	0.20	23.34

Table 5.1: Mean of each performance metric for fully simulated experiment where $K = 2$.

In addition to the mean of each performance metric, we present learning curves and 95% confidence intervals (shaded areas) in Figure 5.2. We provide learning curves for reward, CTR, and users left in the system. Figure 5.2a represents the moving average of the mean rewards received during one episode. The moving average is calculated over the mean rewards received during one episode with the window set to 20. Figure 5.2b represents performance in average CTR per episode. Figure 5.2c shows the user retention measured by average users left calculated over five runs. Both RESR with and without diversification show efficient learning compared to SlateQ. The curves drawn by RESRs are steeper than the baseline, which means that our method achieves a high level of performance faster. The areas of the reward under the curve (AUC) are 71% and 56% higher for RESR without diversification and with diversification compared to SlateQ.



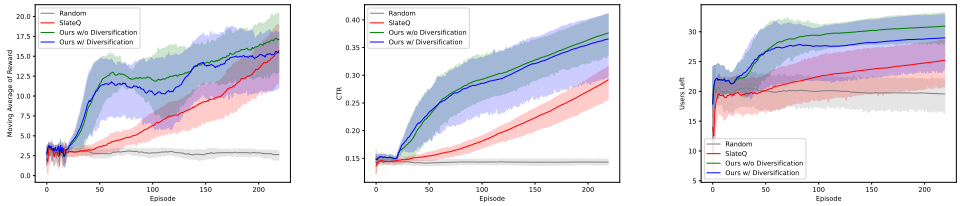
(a) Average of episodic reward (b) Average CTR (c) Average users left

Figure 5.2: Learning curves of different performance metrics for $K = 2$.

Table 5.2 summarizes the result for $K = 3$. Experiment settings are the same as the previous one where $K = 2$. We observe that our method outperforms SlateQ. RESR exhibit efficient learning as AUCs for reward are 58% and 47% higher without diversification and with diversification, respectively, compared to SlateQ. Efficient learning is shown in Figure 5.3 where RESR exhibits steep learning curves. The setting $K = 3$ yields a higher reward compared to $K = 2$ as more choice is given to the user, which increases the likelihood of an item getting chosen.

Policy	Reward	CTR	Users Left
Random	2.86	0.14	19.95
SlateQ	7.97	0.20	22.26
RESR	**12.55**	**0.28**	**28.21**
RESR+Div.	11.72	0.28	26.91

Table 5.2: Mean of each performance metric for fully simulated experiment where $K = 3$.



(a) Average of episodic reward (b) Average CTR (c) Average users left

Figure 5.3: Learning curves of different performance metrics for $K = 3$.

5.5.2 Simulation using the Real-World Data

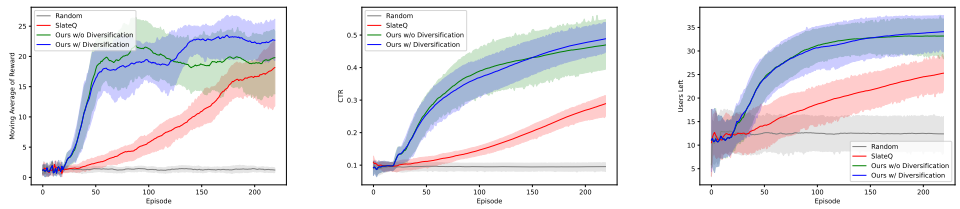
In the experiment, we sampled 100 users using different random seeds with the same assumption that a user only chooses one item at a time and the transition is affected only by the item chosen. There are 4 users in the system at the beginning. The macro & micro settings of the environment are the same as the fully simulated experiment except for the sensitivity parameter η in the user state transition. We set η to 0.01 because the extracted user feature elements have a smaller variance compared to the fully simulated version where we sampled features from a normal distribution with mean 0 and variance 0.7^2. Users enter and leave the system depending on the clicks occurred and each user's fatigue level. The item vectors stay the same, whereas user interests evolve based on the previous item selection as shown in Eq. (5.1).

Table 5.3 shows the mean result of each performance metric where $K = 2$. The results are averaged over five runs with different random seeds. Figure 5.4 shows that RESRs' initial and final performance outperform the baseline SlateQ. RESRs have bet-

ter sample complexity , achieving higher rewards with fewer data observed. The AUCs for reward are 98% and 110% higher without diversification and with diversification, respectively, compared to `SlateQ`. The performance gap is more significant for the real-world data simulation case than fully simulated experiments. The diversification of Q-functions led to higher rewards when experimenting with features extracted from real-world data, especially as the learning continued. The results show that our method is robust and applicable to settings where real-world data is used.

Policy	Reward	CTR	Users Left
Random	1.33	0.10	12.42
`SlateQ`	8.69	0.17	18.71
RESR	17.14	**0.35**	**27.77**
RESR+Div.	**18.17**	0.34	27.76

Table 5.3: Mean of each performance metric for real-world data experiment where $K = 2$.



(a) Average of episodic reward (b) Average CTR (c) Average users left

Figure 5.4: Learning curves of different performance metrics for real-world data experiment where $K = 2$.

Chapter 6

CONCLUSION

A slate-based recommendation problem is formulated as a POMDP. We consider latent user states and provide a solution in a generalized setting. We develop RESR, which utilizes multiple randomized Q-functions and approximate sampling for a slate-based recommendation. The method contributes to efficient exploration for faster learning. We experiment in a simulated environment where the goal is to maximize cumulative reward from user clicks. To further validate the algorithm, we test using user and item features extracted from a real-world dataset. The performance results from various experiment settings show the efficacy of our method.

Chapter A

APPENDIX

A.1 Notation

We provide a table of notations for easy reference.

N	Number of items
K	Slate size (number of items recommended at once)
D	Dimension of true latent state
w	Observation window size
o_t	Observation at time t
s_t	True user latent state at time t which is not observed
$\hat{s}_t$	Pseudo-state of user at time t
a_t	Slate selected at time t
c_t	Additional user information at time t (e.g. selected item, duration of interaction, age, etc.)

$p(i\|s_t, a_t)$	Probability of choosing item i when slate a_t is given to user with latent state s_t at time t
$p(i\|\hat{s}_t, a_t)$	Probability of choosing item i when slate a_t is given to user with pseudo-state $\hat{s}_t$ at time t
ϕ	Latent state representation function that takes observation o_t as input and outputs pseudo-state $\hat{s}_t$
g	Affinity function that takes user state and item feature as input and outputs affinity score
$Q(\cdot, \cdot)$	Q-function that takes pseudo-state and slate as input and outputs estimated value of recommending corresponding slate
$q(\cdot, \cdot)$	Item-wise Q-function that takes pseudo-state and item feature as input and outputs estimated value for corresponding item
ϕ	Parameters of function ϕ
β	Parameters of function g
θ	Parameters of online q function
θ'	Parameters of target q function
x_i	Feature vector of item i
ρ	Base affinity score for no-choice option
$\mathcal{D}$	Training dataset of size $\|\mathcal{D}\|$

B	Number of randomized qs
$\mathcal{L}^{\text{CE}}$	Cross-entropy loss of user choice model of batch size m
$\mathcal{L}^{\text{RL}}$	DQN objective loss of batch size m
$\mathcal{L}^{\text{div}}$	Diversification loss computed from Q-function diversification for batch size m
α^{CE}	Tuning parameter for cross-entropy loss
α^{div}	Tuning parameter for diversification loss
$\mathcal{L}^{\text{total}}$	$\mathcal{L}^{\text{RL}} + \alpha^{\text{CE}}\mathcal{L}^{\text{CE}} + \alpha^{\text{div}}\mathcal{L}^{\text{div}}$
λ	Rate of Poisson distribution for user entrance
τ	Scale parameter for total number of clicks
clicks_t	Total number of clicks occurred in system for time t
p_{out_x}	Current probability of user x leaving system
fatigue_x	Current fatigue level of user x
b	Base fatigue level constant setting $p_{\text{out}_x} = 0.5$
v_i	Topic of item i
μ_f	Mean of multivariate normal distribution that item feature is sampled from
Σ_f	Covariance matrix of multivariate normal distribution that item feature is sampled from

κ_i	Quality of item i
μ_d	Expected quality of normal distribution where item quality with topic d is sampled from
σ_d	Standard deviation of normal distribution where item quality with topic d is sampled from
δ	Magnitude scale parameter of user state transition
η	Sensitivity parameter of user state transition

A.2 Details of the Experiment

The experiments were done using RecSim NG, a probabilistic platform for multi-agent recommender systems simulation. More information can be found in the white paper [33] and the code repository at `https://github.com/google-research/recsim_ng`.

A.2.1 Fully Simulated Recommendation

Macro-level Environment Settings

User Arrival The rate of Poisson distribution at which new users enter the system, λ, is determined by

$$\lambda_t = 0.5 \times \text{clicks}_{t-1}$$

where clicks_t is the number of clicks incurred by users in the system in the previous timestep.

User Departure The probability of user x leaving the system, p_{out_x}, is determined

by the fatigue level of that user as shown below.

$$p_{\text{out}_x} = \frac{1}{1 + \exp\left(-\text{fatigue}_x + 4\right)}$$

The minimum value of fatigue is 0. The initial value of fatigue for each user is 0. As shown below, fatigue increases by 1 if the user consumes a bad-quality item and decreases by 0.5 if the user consumes a good-quality item.

$$\text{fatigue}_x = \begin{cases} \min\left(0, \text{fatigue}_x - 0.5\right) & \text{if } q_i \geq 0 \\ \text{fatigue}_x + 1 & \text{if } q_i < 0 \end{cases}$$

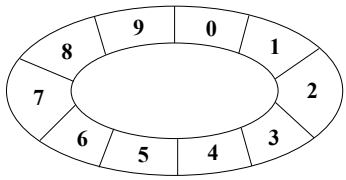
Micro-level Environment Settings

A fully simulated recommendation experiment was conducted using synthetic data generated as described in Section 5.3. Here we provide a summary of each entity and its generation process. The dimensions of both the user state and the item feature vectors are set to 10 (i.e., $D = 10$).

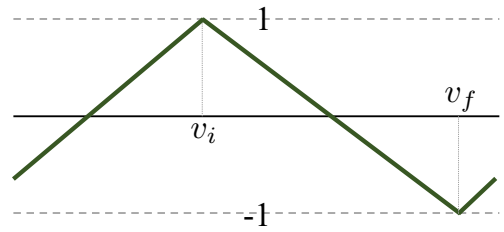
Item Each element of the item i's feature vector x_i is sampled independently from a normal distribution where mean μ_{fv} is determined by the item's topic v_i. All elements have a standard deviation of 0.7. We set the number of topics to 10.

Topics are presented as a circular array where the distance represents the similarity between the items. This is depicted in Figure A.1a. The closer the distance, the more similar topics are. The features are generated in a manner that similar topics have similar feature values. Item i's topic $v_i \in \{1, 2, \ldots, 10\}$ decides v_ith element of $\boldsymbol{\mu}_f$. The mean value for v_ith element of vector $\boldsymbol{\mu}_f$ is 1; for the farthest away (most different) topic, v_f, the mean of v_fth element of $\boldsymbol{\mu}_f$ is -1. Elements in between v_i and v_f have linearly decayed from 1 to -1. The visual representation of how each element μ_{fv} of $\boldsymbol{\mu}_f$ is determined is shown in Figure A.1.

The quality of an item i is sampled from μ_d which corresponds to the mean quality



(a) Circular alignment of topics



(b) Mean of an item feature elements

Figure A.1: Left shows the circular alignment of the topics. When the total number of topics is 10, topic 0 is close to 9, 1; farthest away from 5. Right describes how the mean of an item feature's v-th element, $\boldsymbol{\mu}_v$, is determined. Current topic v_i and farthest away topic v_f determine the mean of the element to be sampled from. If $v_i = 0$ then the item feature is sampled from a normal distribution with $\boldsymbol{\mu}_v = \{1.0, 0.6, 0.2, -0.2, -0.6, -1.0, -0.6, -0.2, 0.2, 0.6\}$.

of items with topic d. We have set $\mu_d = 0, \sigma_d = 0.1$ for all $d \in \{1, \ldots, D\}$.

User Each element of the 10-dimensional latent user state is sampled from a normal distribution $N(0, 1)$. A user's state s changes after an item has been consumed. The user state transition differs by the quality of the item selected. In summary, the user is more likely to choose the item if the quality is good and vice versa if the quality is bad. The user state at time t after item i has been chosen at time $t-1$ can be represented as

$$s_t = \begin{cases} (1+\delta) \times [s_{t-1} + \eta\kappa_i(x_i - s_{t-1}) + \xi] & \text{if } \kappa_i \geq 0 \\ (1-\delta) \times [s_{t-1} + \eta\kappa_i(x_i - s_{t-1}) + \xi] & \text{if } \kappa_i < 0 \end{cases}$$

- s_t: User latent state at time t
- κ_i: Quality scalar of the chosen item
- x_i: Item feature vector of the chosen item
- δ: Magnitude scale parameter

- η: Sensitivity parameter
- ξ: Gaussian noise

In the experiment of the simulated environment, we set $\delta = 0.01$, $\eta = 0.1$, and $\xi = 0$.

The true user choice model computes the similarity between user state s and item feature using the dot product. (i.e. $u(s, x_i) = e^{s \cdot x_i}$). The score for not choosing any item $\rho = 7$ in the experiments. The probability of a user with state s choosing i given a slate a can be written as

$$P(i|a) = \frac{\exp(s \cdot x_i)}{7 + \Sigma_{l \in a} \exp(s \cdot x_l)}$$

The agent learns the choice model via cross-entropy loss computed from previous user choices. Latent user representation is learned using DQN objective and cross-entropy loss. The training hyperparameters for the overall experiments is provided below.

Table A.1: Training Hyperparameters

Initial ϵ for `SlateQ`	1
Final ϵ for `SlateQ`	0.01
Total decaying steps	22000
Number of randomized item-wise Q-functions B	10
Learning rate	0.00015
Optimizer	RMSprop
ρ / momentum / ϵ / centered / clipnorm in RMSprop	0.95 / 0.0 / 1e-07 / True / 1.0
Discount factor γ	0.99
Batch size	64
Masking distribution	Bernoulli(0.9)
Update period (steps)	4
Target update period (steps)	4000
Learning starts (steps)	2000
LSTM hidden layer size	32
Item-wise Q-network hidden layer size	32-32
Random seeds	250369352, 45901546, 492513979, 74141201, 58295048

A.2.2 Simulation using the Real-World Data

The environment settings for the real-world data simulation are the same as the previous fully simulated experiment except for the sensitivity parameter η in the user state transition. We decreased η from 0.1 to 0.01 as the average standard deviation of extracted user feature elements is about 0.1 compared to 0.7 in the fully simulated experiment. We provide a detailed feature extraction process from the MovieLens 100K dataset.

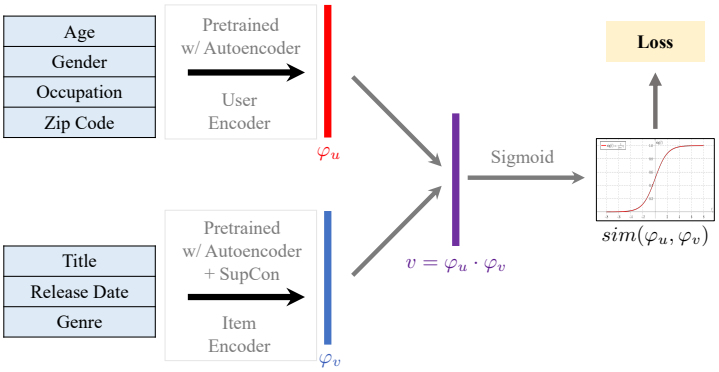


Figure A.2: Feature extraction process from real-world data

First, we preprocess the input data. For user data, we normalize the *age* and represent categorical features *occupation* and *zip code* as one-hot encoded vectors. The dimension of the input is 818 after this preprocessing process. For item data, we have the *title* column in natural language. We use a pretrained *RoBERTa base* [37] model to utilize its 768-dimensional word embedding. We truncated the *title* to have a maximum word length of 8. This results in a total of $768 \times 8 = 6144$ dimensions for the *title*. The *release date* column has been normalized and the *genre* has been represented in a one-hot vector. The final input dimension of item data is 6166.

Second, we train autoencoders using both user and item data. This process disentangles the feature vectors when mapped to a latent dimension. The encoder part is used to encode the input data to 32-dimensional embedding vectors. We added a batch normalization layer before the item input to hinder item features from having large numbers. Supervised contrastive learning [34] (SupCon) with genre labels and item latent embeddings as inputs is done to cluster embeddings of the same genre. An item may have multiple genres, so we used items with only one genre for supervised contrastive learning.

Finally, we train a Siamese neural network with cross-entropy loss using the whole dataset. Siamese network consists of a layer that takes input embedding and maps it into the same dimension. Newly mapped embeddings for a user and an item is then taken as arguments of the dot product layer. The result of the dot product are then fed into the sigmoid function to yield an estimated similarity score. The cross-entropy loss is calculated using this estimate and label, which is arranged in two different ways: hard label and soft label. A hard target label is used for the initial round of training. Ratings of 1 to 3 are labeled as 0, and ratings of 4 and 5 are labeled as 1. We fix the embedding layer for 90 epochs and we fine-tune the whole architecture for 10 epochs. Next, we train another round using a soft target label. Each rating corresponds to a different level of similarity using a sigmoid function σ. In the experiment, each rating corresponds to the following $\{1 : \sigma(-4), 2 : \sigma(-2), 3 : \sigma(0), 4 : \sigma(2), 5 : \sigma(4)\}$ ($\sigma(-4) = 0.02, \sigma(-2) = 0.12, \sigma(0) = 0.5, \sigma(2) = 0.88, \sigma(4) = 0.98$). In the second round of training, we do not fix any layers and train for 100 epochs to yield the final feature vectors of users and items.

Number of layers (Encoder)	3
Number of layers (Decoder)	3
Layer size (Encoder-User)	$\{128, 64, 32\}$
Layer size (Decoder-User)	$\{64, 128, 818\}$
Layer size (Encoder-Item)	$\{1024, 128, 32\}$
Layer size (Decoder-Item)	$\{128, 1024, 6166\}$
SupCon temperature	0.05
Learning rate	0.001
Optimizer	Adam
Training epochs	100
Training batch size	256
Random Seed	1299827

Table A.2: Training configurations for autoencoders and SupCon

Round1 initial learning rate	0.01
Round1 initial training epochs	90
Round1 fine-tuning learning rate	0.0001
Round1 fine-tuning training epochs	10
Round2 learning rate	0.0005
Round2 training epochs	100
Learning rate schedule	Exponential decay
Learning rate decay steps & rate	200, 0.96
Optimizer	Adam
Training batch size	512
Random Seed	1299827

Table A.3: Training configurations for Siamese network

A.3 Algorithm

Algorithm 3 RESR with Exact Slate Construction

Parameters: Observation window size w. Masking distribution M. Update period U. Target update period C. Slate size K. Buffer size N. Number of randomized value functions B.

Initialize experience replay buffer RB to capacity N
Initialize ϕ with random weights ψ
Initialize affinity function g with random weights β
Initialize B item-wise value function $q_{b=1}^{B}$ using random weights θ
Initialize B target item-wise action-value function $q_{b=1}^{B}$ with weights $\theta' = \theta$
for each episode **do**
  Sample K value functions from Uniform$\{1, \ldots, B\}$ without replacement
  for each timestep t **do**
    Fetch user history $o_t = \{(a_{t-w}, c_{t-w}), (a_{t-w+1}, c_{t-w+1}) \ldots, (a_{t-1}, c_{t-1})\}$
    Compute latent user state $\hat{s}_t = \phi(o_t)$
    Compute $p(i|\hat{s}_t, a)$ using Eq. (4.2)
    Compute $Q(\hat{s}_t, a)$. $Q(\hat{s}_t, a) = \sum_{k=1}^{K} p(i_k|\hat{s}_t, a) q_k(\hat{s}_t, i_k)$
    Pick a slate according to $a_t \in \arg\max_a Q(\hat{s}_t, a)$
    Observe user response c_t and receive reward r_t
    Next observation $o_{t+1} = \{(a_{t-w+1}, c_{t-w+1}), (a_{t-w+2}, c_{t-w+2}), \ldots, (a_t, c_t)\}$
    Sample bootstrap mask $e_t \sim M$
    Add (o_t, r_t, o_{t+1}, e_t) to buffer RB
    if t mod U = 0 **then**
      Sample random minibatch from RB
      Compute loss. $L^{\text{total}} = L^{\text{RL}} + \alpha^{\text{CE}} L^{\text{CE}} + \alpha^{\text{div}} L^{\text{div}}$
      Update parameters ψ, β, θ using SGD
    end if
    if $t \; mod \; C$ = 0 **then**
      Reset $\theta' \leftarrow \theta$
    end if
  end for
end for

Bibliography

[1] S. Pandey, D. Agarwal, D. Chakrabarti, and V. Josifovski, "Bandits for taxonomies: A model-based approach," in *Proceedings of the 2007 SIAM International Conference on Data Mining*, pp. 216–227, SIAM, 2007.

[2] J. Langford and T. Zhang, "The epoch-greedy algorithm for contextual multi-armed bandits," *Advances in neural information processing systems*, vol. 20, no. 1, pp. 96–1, 2007.

[3] L. Li, W. Chu, J. Langford, and R. E. Schapire, "A contextual-bandit approach to personalized news article recommendation," in *Proceedings of the 19th international conference on World wide web*, pp. 661–670, 2010.

[4] T. Lu, D. Pál, and M. Pál, "Contextual multi-armed bandits," in *Proceedings of the Thirteenth international conference on Artificial Intelligence and Statistics*, pp. 485–492, JMLR Workshop and Conference Proceedings, 2010.

[5] E. Ie, V. Jain, J. Wang, S. Narvekar, R. Agarwal, R. Wu, H.-T. Cheng, T. Chandra, and C. Boutilier, "Slateq: A tractable decomposition for reinforcement learning with recommendation sets," in *Proceedings of the Twenty-eighth International Joint Conference on Artificial Intelligence (IJCAI-19)*, pp. 2592–2599, 2019. See arXiv:1905.12767 for a related and expanded paper (with additional material and authors).

[6] D. McFadden, "Modelling the choice of residential location," 1977.

[7] G. Shani, D. Heckerman, R. I. Brafman, and C. Boutilier, "An mdp-based recommender system.," *Journal of Machine Learning Research*, vol. 6, no. 9, 2005.

[8] F. Mi and B. Faltings, "Adaptive sequential recommendation using context trees.," in *IJCAI*, pp. 4018–4019, 2016.

[9] X. Zhao, W. Zhang, and J. Wang, "Interactive collaborative filtering," in *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, pp. 1411–1420, 2013.

[10] H. Chen, X. Dai, H. Cai, W. Zhang, X. Wang, R. Tang, Y. Zhang, and Y. Yu, "Large-scale interactive recommendation with tree-structured policy gradient," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, pp. 3312–3320, 2019.

[11] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[12] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[13] X. Zhao, L. Xia, L. Zhang, Z. Ding, D. Yin, and J. Tang, "Deep reinforcement learning for page-wise recommendations," in *Proceedings of the 12th ACM Conference on Recommender Systems*, pp. 95–103, 2018.

[14] X. Zhao, L. Zhang, Z. Ding, L. Xia, J. Tang, and D. Yin, "Recommendations with negative feedback via pairwise deep reinforcement learning," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1040–1048, 2018.

[15] G. Zheng, F. Zhang, Z. Zheng, Y. Xiang, N. J. Yuan, X. Xie, and Z. Li, "Drn: A deep reinforcement learning framework for news recommendation," in *Proceedings of the 2018 world wide web conference*, pp. 167–176, 2018.

[16] M. Chen, A. Beutel, P. Covington, S. Jain, F. Belletti, and E. H. Chi, "Top-k off-policy correction for a reinforce recommender system," in *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, pp. 456–464, 2019.

[17] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," *Advances in neural information processing systems*, vol. 12, 1999.

[18] X. Chen, S. Li, H. Li, S. Jiang, Y. Qi, and L. Song, "Generative adversarial user model for reinforcement learning based recommendation system," in *International Conference on Machine Learning*, pp. 1052–1061, PMLR, 2019.

[19] V. François-Lavet, P. Henderson, R. Islam, M. G. Bellemare, J. Pineau, *et al.*, "An introduction to deep reinforcement learning," *Foundations and Trends® in Machine Learning*, vol. 11, no. 3-4, pp. 219–354, 2018.

[20] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals, "Understanding deep learning (still) requires rethinking generalization," *Communications of the ACM*, vol. 64, no. 3, pp. 107–115, 2021.

[21] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[22] C. J. C. H. Watkins, *Learning from delayed rewards*. PhD thesis, King's College, Cambridge United Kingdom, 1989.

[23] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[24] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[25] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[26] I. Osband, B. Van Roy, D. J. Russo, Z. Wen, *et al.*, "Deep exploration via randomized value functions.," *J. Mach. Learn. Res.*, vol. 20, no. 124, pp. 1–62, 2019.

[27] I. Osband, C. Blundell, A. Pritzel, and B. Van Roy, "Deep exploration via bootstrapped dqn," *Advances in neural information processing systems*, vol. 29, 2016.

[28] W. R. Thompson, "On the likelihood that one unknown probability exceeds another in view of the evidence of two samples," *Biometrika*, vol. 25, no. 3-4, pp. 285–294, 1933.

[29] D. J. Russo, B. Van Roy, A. Kazerouni, I. Osband, Z. Wen, *et al.*, "A tutorial on thompson sampling," *Foundations and Trends® in Machine Learning*, vol. 11, no. 1, pp. 1–96, 2018.

[30] K. D. Chen and W. H. Hausman, "Mathematical properties of the optimal product line selection problem using choice-based conjoint analysis," *Management Science*, vol. 46, no. 2, pp. 327–332, 2000.

[31] M. Mladenov, O. Meshi, J. Ooi, D. Schuurmans, and C. Boutilier, "Advantage amplification in slowly evolving latent-state environments," in *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, pp. 3165–3172, 2019.

[32] R. S. Sutton, D. Precup, and S. Singh, "Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning," *Artificial intelligence*, vol. 112, no. 1-2, pp. 181–211, 1999.

[33] M. Mladenov, C.-W. Hsu, V. Jain, E. Ie, C. Colby, N. Mayoraz, H. Pham, D. Tran, I. Vendrov, and C. Boutilier, "Recsim NG: Toward principled uncertainty modeling for recommender ecosystems," *arXiv preprint arXiv:2103.08057*, 2021.

[34] P. Khosla, P. Teterwak, C. Wang, A. Sarna, Y. Tian, P. Isola, A. Maschinot, C. Liu, and D. Krishnan, "Supervised contrastive learning," *Advances in Neural Information Processing Systems*, vol. 33, pp. 18661–18673, 2020.

[35] J. Bromley, I. Guyon, Y. LeCun, E. Säckinger, and R. Shah, "Signature verification using a "siamese" time delay neural network," *Advances in neural information processing systems*, vol. 6, 1993.

[36] G. Koch, R. Zemel, R. Salakhutdinov, *et al.*, "Siamese neural networks for one-shot image recognition," in *ICML deep learning workshop*, vol. 2, p. 0, Lille, 2015.

[37] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized BERT pretraining approach," *CoRR*, vol. abs/1907.11692, 2019.

초 록

온라인 추천 시스템에서 사용자의 장기적 가치를 최대화하기 위한 방법으로 강화학습을 활용할 수 있다. 일반적인 추천 시스템과 다르게 강화학습 기반 추천 시스템은 사용자의 선택에 따른 변화를 포착하고 장기적 차원에서 사용자의 가치를 높일 수 있다. 본 논문에서는 강화학습을 실제 적용하는 과정에서 필요한 효율적인 탐색 방법을 다룬다. 우선, 강화학습 에이전트와 사용자 및 아이템으로 이루어진 추천 문제를 부분 관찰 마르코프 의사결정 과정(POMDP)을 이용한 순차적 의사결정 문제로 구성한다. 슬레이트(Slate)라고 불리는 여러 개의 아이템으로 구성된 리스트를 사용자에게 추천하는 문제를 풀고자 한다. 본 논문은 바로 관측이 어려운 사용자의 잠재 상태를 다룬다는 점에서 과거 연구의 일반화된 문제를 연구한다. 본 논문에서 제시하는 알고리듬인 RESR은 효율적인 학습을 위한 사용자의 잠재 임베딩 및 사용자 선택 모형 학습 방법과 더불어 랜덤화된 여러 개의 Q 함수를 샘플링하여 사후분포를 근사하는 방법을 활용한다. 온라인 시뮬레이션 실험에서 알고리듬의 성능을 비교·분석한 결과 제시된 방법이 탐색 효율성 측면에서 나은 성능을 보이는 것을 확인할 수 있었다.

주요어: 딥 강화학습, 추천 시스템, 탐색, 시뮬레이션, POMDP

학번: 2021-28044