Deep Reinforcement Learning for Page-wise Recommendations

Xiangyu Zhao
Data Science and Engineering Lab
Michigan State University
zhaoxi35@msu.edu

Long Xia
Data Science Lab
JD.com
xialong@jd.com

Liang Zhang
Data Science Lab
JD.com
zhangliang16@jd.com

Zhuoye Ding
Data Science Lab
JD.com
dingzhuoye@jd.com

Dawei Yin
Data Science Lab
JD.com
yindawei@acm.org

Jiliang Tang
Data Science and Engineering Lab
Michigan State University
tangjili@msu.edu

ABSTRACT

Recommender systems can mitigate the information overload problem by suggesting users' personalized items. In real-world recommendations such as e-commerce, a typical interaction between the system and its users is – users are recommended a page of items and provide feedback; and then the system recommends a new page of items. To effectively capture such interaction for recommendations, we need to solve two key problems – (1) how to update recommending strategy according to user's *real-time feedback*, and 2) how to generate a page of items with proper display, which pose tremendous challenges to traditional recommender systems. In this paper, we study the problem of page-wise recommendations aiming to address aforementioned two challenges simultaneously. In particular, we propose a principled approach to jointly generate a set of complementary items and the corresponding strategy to display them in a 2-D page; and propose a novel page-wise recommendation framework based on deep reinforcement learning, DeepPage, which can optimize a page of items with proper display based on real-time feedback from users. The experimental results based on a real-world e-commerce dataset demonstrate the effectiveness of the proposed framework.

KEYWORDS

Recommender Systems, Deep Reinforcement Learning, Actor-Critic, Item Display Strategy, Sequential Preference.

ACM Reference Format:
Xiangyu Zhao, Long Xia, Liang Zhang, Zhuoye Ding, Dawei Yin, and Jiliang Tang. 2018. Deep Reinforcement Learning for Page-wise Recommendations. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Recommender systems are intelligent E-commerce applications. They assist users in their information-seeking tasks by suggesting items (products, services, or information) that best fit their needs and preferences. Recommender systems have become increasingly popular in recent years, and have been utilized in a variety of domains including movies, music, books, search queries, and social tags[25, 26]. Figure 1 illustrates a typical example of the interactions between an e-commerce recommender system and a user – each time the system recommends a page of items to the user; next the user browses these items and provides real-time feedback and then the system recommends a new page of items. This example suggests two key challenges to effectively take advantage of these interactions for e-commerce recommender systems – 1) how to efficiently capture user's dynamic preference and update recommending strategy according to user's real-time feedback; and 2) how to generate a page of items with proper display based on user's preferences.

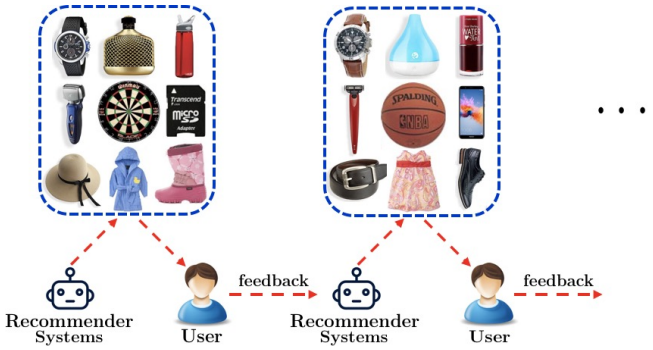

Figure 1: An example to illustrate the interactions between recommender systems and users.

1.1 Real-time Feedback

Most existing recommender systems consider the recommendation procedure as a static process and make recommendations following a fixed greedy strategy. However, these approaches may fail in capturing the dynamic nature of the users' preferences, and they become infeasible to efficiently and continuously update their recommending strategies according to user's real-time feedback. Thus, in this work, we consider the recommendation procedure as sequential interactions between users and the recommender agent; and leverage Reinforcement Learning (RL) to automatically learn the optimal recommendation strategies. Recommender systems based on reinforcement learning have two major advantages. First, they are able to continuously update their strategies based on user's real-time feedback during the interactions, until the system converges to the optimal strategy that generates recommendations best

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

fitting users' dynamic preferences. Second, the optimal strategy is made by maximizing the expected long-term cumulative reward from users; while the majority of traditional recommender systems are designed to maximize the immediate (short-term) reward of recommendations [27]. Therefore, the system can identify items with small immediate rewards but making big contributions to the rewards for future recommendations.

1.2 Page-wise Recommendations

As mentioned in the example, users are typically recommended a page of items. To achieve this goal, we introduce a page-wise recommender system, which is able to jointly (1) generate a set of diverse and complementary items and (2) form an item display strategy to arrange the items in a 2-D page that can lead to maximal reward. Conventional RL methods could recommend a set of items each time. For instance, DQN can recommend a set of items with highest Q-values according to the current state[19]. However, these approaches recommend items based on the same state, which leads to the recommended items to be similar. In practice, a bundling of complementary items may receive higher rewards than recommending all similar items. For instance, in real-time news feed recommendations, a user may want to read diverse topics of interest[36]. In addition, page-wise recommendations need to properly display a set of generated items in a 2-D page. Traditional approaches treat it as a ranking problem, i.e., ranking items into a 1-D list according to the importance of items. In other words, user's most preferred item is posited in the top of list. However, in e-commerce recommender systems, a recommendation page is a 2-D grid rather than a 1-D list. Also eye-tracking studies [28] show that rather than scanning a page in a linear fashion, users do page chunking, i.e., they partition the 2-D page into chunks, and browse the chunk they prefer more. In addition, the set of items and the display strategy are generated separately; hence they may be not optimal to each other. Therefore, page-wise recommendations need principled approaches to simultaneously generate a set of complementary items and the corresponding display strategy in a 2-D page.

1.3 Contributions

In this paper, we tackle the two aforementioned challenges simultaneously by introducing a novel page-wise recommender system based on deep reinforcement learning. We summarize our major contributions as follows:

- We introduce a principled approach to generate a set of complementary items and properly display them in one 2-D recommendation page simultaneously;
- We propose a page-wise recommendation framework DeepPage, which can jointly optimize a page of items by incorporating real-time feedback from users;
- We demonstrate the effectiveness of the proposed framework in a real-world e-commerce dataset and validate the effectiveness of the components in DeepPage for accurate recommendations.

2 THE PROPOSED FRAMEWORK

In this section, we first give an overview of the proposed Actor-Critic based reinforcement learning recommendation framework with notations. Then we present the technical details of components in Actor and Critic, respectively.

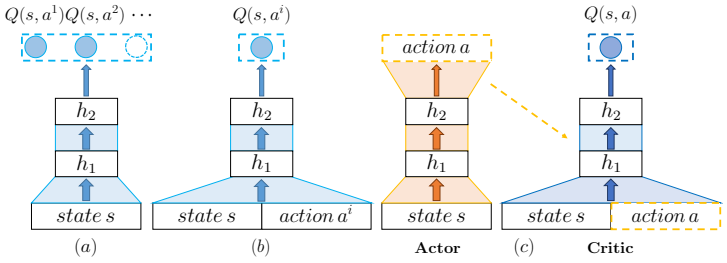

Figure 2: Framework architecture selection.

2.1 Framework Overview

As mentioned in Section 1.1, we model the recommendation task as a Markov Decision Process (MDP) and leverage Reinforcement Learning (RL) to automatically learn the optimal recommendation strategies, which can continuously update recommendation strategies during the interactions and the optimal strategy is made by maximizing the expected long-term cumulative reward from users. In practice, conventional RL methods like POMDP[27] and Q-learning[31] become infeasible with the increasing number of items for recommendations. Thus, we leverage Deep Reinforcement Learning[14] with (adapted) artificial neural networks as the non-linear approximators to estimate the action-value function in RL. This model-free reinforcement learning method does not estimate the transition probability and not store the Q-value table. Hence it can support huge amount of items in recommender systems.

There are two major challenges when we apply deep reinforcement learning to the studied problem – (a) the large (or even continuous) and dynamic action space (item space), and (b) the computational cost to select an optimal action (a set of items). In practice, only using discrete indices to denote items is not sufficient since we cannot know the relations between different items only from indices. One common way is to use extra information to represent items with continuous embeddings[13]. Besides, the action space of recommender systems is dynamic as items arriving and leaving. Moreover, computing the Q-value for all state-action pairs is a time-consuming task because of the enormous state and action spaces.

To tackle these challenges, in this paper, our recommending policy builds upon the Actor-Critic framework [30], shown in Figure 2 (c). The Actor-Critic architecture is preferred from the studied problem since it is suitable for large and dynamic action space, and can also reduce redundant computation simultaneously compared to alternative architectures as shown in Figures 2 (a) and (b). The conventional Deep Q-learning architectures shown in Figure 2 (a) inputs only the state space and outputs Q-values of all actions. This architecture is suitable for the scenario with high state space and small/fixed action space like Atari[19], but cannot handle large and dynamic action space scenario, like recommender systems. Also, we cannot leverage the second conventional deep Q-learning architecture as shown in Fig.2(b) because of its temporal complexity. This architecture inputs a state-action pair and outputs the Q-value correspondingly, and makes use of the optimal action-value function $Q^*(s,a)$. It is the maximum expected return achievable by the

optimal policy, and should follow the Bellman equation [3] as:

$$Q^*(s,a) = \mathbb{E}_{s'}\left[r + \gamma \max_{a'} Q^*(s',a')|s,a\right]. \quad (1)$$

In practice, selecting an optimal a', $|\mathcal{A}|$ evaluations is necessary for the inner operation "$\max_{a'}$". In other words, this architecture computes Q-value for all $a' \in \mathcal{A}$ separately, and then selects the maximal one. This prevents Equation (1) from being adopted in practical recommender systems.

In the Actor-Critic framework, the Actor architecture inputs the current state s and aims to output a deterministic action (or recommending a deterministic page of M items), i.e., $s \rightarrow a = \{a^1, \cdots, a^M\}$. The Critic inputs only this state-action pair rather than all potential state-action pairs, which avoids the aforementioned computational cost as follows:

$$Q(s,a) = \mathbb{E}_{s'}\left[r + \gamma Q(s',a')|s,a\right]. \quad (2)$$

where the Q-value function $Q(s,a)$ is a judgment of whether the selected action matches the current state, i.e., whether the recommendations match user's preference. Finally, according to the judgment from Critic, the Actor updates its' parameters in a direction of boosting recommendation performance so as to output proper actions in the next iteration. With the above design intuitions, we formally define the tuple of five elements $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$ of MDP as follows:

- **State space** $\mathcal{S}$: A state $s \in \mathcal{S}$ is defined as user's current preference, which is generated based on user's browsing history, i.e., the items that a user browsed and her corresponding feedback.
- **Action space** $\mathcal{A}$: An action $a = \{a^1, \cdots, a^M\} \in \mathcal{A}$ is to recommend a page of M items to a user based on current state s.
- **Reward** $\mathcal{R}$: After the RA takes an action a at the state s, i.e., recommending a page of items to a user, the user browses these items and provides her feedback. She can skip (not click), click, or order these items, and the agent receives immediate reward $r(s,a)$ according to the user's feedback.
- **Transition** $\mathcal{P}$: Transition $p(s'|s,a)$ defines the state transition from s to s' when RA takes action a.
- **Discount factor** γ: $\gamma \in [0,1]$ defines the discount factor when we measure the present value of future reward. In particular, when $\gamma = 0$, RA only considers the immediate reward. In other words, when $\gamma = 1$, all future rewards can be counted fully into that of the current action.

Specifically, we model the recommendation task as a MDP in which a recommender agent (RA) interacts with environment $\mathcal{E}$ (or users) over a sequence of time steps. At each time step, the RA takes an action $a \in \mathcal{A}$ according to $\mathcal{E}$'s state $s \in \mathcal{S}$, and receives a reward $r(s,a)$ (or the RA recommends a page of items according to user's current preference, and receives user's feedback). As the consequence of action a, the environment $\mathcal{E}$ updates its state to s' with transition $p(s'|s,a)$. The goal of reinforcement learning is to find a recommendation policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$, which can maximize the cumulative reward for the recommender system. Next we will elaborate the Actor and Critic architectures for the proposed framework.

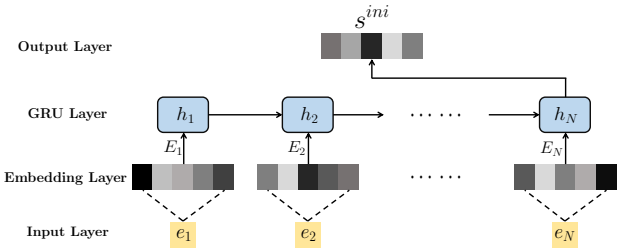

Figure 3: Encoder to generate initial state s^{ini}.

2.2 Architecture of Actor Framework

The Actor is designed to generate a page of recommendations according to user's preference, which needs to tackle three challenges – 1) setting an initial preference at the beginning of a new recommendation session, 2) learning the real-time preference in the current session, which should capture the dynamic nature of user's preference in current session and spatial item display patterns in a page, and 3) jointly generating a set of recommendations and displaying them in a 2-D page. To address these challenges, we propose an Actor framework with the Encoder-Decoder architecture.

2.2.1 ***Encoder for Initial State Generation Process.*** Figure 3 illustrates the model for generating initial preference. We introduce a RNN with Gated Recurrent Units (GRU) to capture users' sequential behaviors as user's initial preference. The inputs of GRU are user's last clicked/ordered items $\{e_1, \cdots, e_N\}$ (sorted in chronological order) before the current session, while the output is the representation of users' initial preference by a vector. The input $\{e_1, \cdots, e_N\}$ is dense and low-dimensional vector representations of items [1]. We add an item-embedding layer to transform e_i into a low-dimensional dense vector via $E_i = \tanh(W_E e_i + b_E) \in \mathbb{R}^{|E|}$ where we use "tanh" activate function since $e_i \in (-1, +1)$.

We leverage GRU rather than Long Short-Term Memory (LSTM) because that GRU outperforms LSTM for capturing users' sequential preference in recommendation task [10]. Unlike LSTM using input gate i_t and forget gate f_t to generate a new state, GRU utilizes an update gate z_t:

$$z_t = \sigma(W_z E_t + U_z h_{t-1}). \quad (3)$$

GRU leverages a reset gate r_t to control the input of the former state h_{t-1}:

$$r_t = \sigma(W_r E_t + U_r h_{t-1}). \quad (4)$$

Then the activation of GRU is a linear interpolation between the previous activation h_{t-1} and the candidate activation $\hat{h}_t$:

$$h_t = (1 - z_t)h_{t-1} + z_t \hat{h}_t, \quad (5)$$

where candidate activation function $\hat{h}_t$ is computed as:

$$\hat{h}_t = \tanh[WE_t + U(r_t \cdot h_{t-1})]. \quad (6)$$

We use the final hidden state h_t as the representation of the user's initial state s^{ini} at the beginning of current recommendation

[1] These item representations are pre-trained using users' browsing history by a company, i.e. each item is treated as a word and the clicked items in one recommendation session as a sentence, and item representations are trained via word embedding[13]. The effectiveness of these item representations is validated by their business such as searching, ranking, bidding and recommendations.

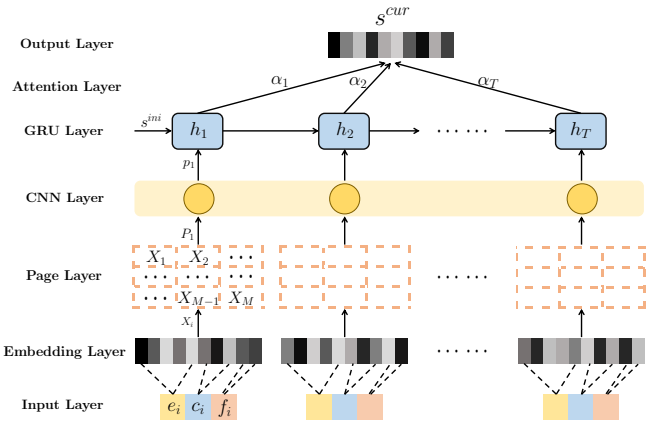

Figure 4: Encoder to generate current state s^{cur}.

session as:

$$s^{ini} = h_t. \tag{7}$$

2.2.2 ***Encoder for Real-time State Generation Process.*** Figure 4 illustrates the model to generate real-time preference in current session. In the page-wise recommender system, the inputs $\{x_1, \cdots, x_M\}$ for each recommendation page are the representations of the items in the page and user's corresponding feedback, where M is the size of a recommendation page and x_i is a tuple as:

$$x_i = (e_i, c_i, f_i), \tag{8}$$

where e_i is the aforementioned item representation. To assist the RA in capturing user's preference among different categories of items and generating complementary recommendations, we incorporate item's category c_i. The item's category c_i is an one-hot indicator vector where $c_i(i) = 1$ if this item belongs to the i^{th} category and other entities are zero. The one-hot indicator vector is extremely sparse and high-dimensional; hence we add a category-embedding layer transforming c_i into a low-dimensional dense vector $C_i = \tanh(W_C c_i + b_C) \in \mathbb{R}^{|C|}$.

In addition to information from items, e_i and c_i, we also want to capture user's interests or feedback in current recommendation page. Thus, we introduce user's feedback vector f_i, which is an one-hot vector to indicate user's feedback for item i, i.e., skip/click/order. Similarly, we transform f_i into a dense vector $F_i = \tanh(W_F f_i + b_F) \in \mathbb{R}^{|F|}$ via the embedding layer. Finally, we get a low-dimensional dense vector $X_i \in \mathbb{R}^{|X|}$ ($|X| = |E| + |C| + |F|$) by concatenating E_i, C_i and F_i as:

$$\begin{aligned} X_i &= concat(E_i, C_i, F_i) \\ &= \tanh\left(concat(W_E e_i + b_E, W_C c_i + b_C, W_F f_i + b_F)\right). \end{aligned} \tag{9}$$

Note that all item-embedding layers share the same parameters W_E and b_E, which reduces the number of parameters and achieves better generalization. We apply the same constraints for category and feedback embeddings.

Then, we reshape the transformed item representations $\{X_1, \cdots, X_M\}$ as the original arrangement in the page. In other words, we arrange the item representations in one page P_t as 2D grids similar to one image. For instance, if one recommendation page has h rows and w columns ($M = h \times w$), we will get a $h \times w|X|$ matrix P_t. To learn item spatial display strategy in one page that leads to maximal reward, we introduce Convolutional Neural Network (CNN). CNN is a successful architecture in computer vision applications because of its capability to apply various learnable kernel filters on image to discover complex spatial correlations [12]. Hence, we utilize 2D-CNN followed by fully connected layers to learn the optimal item display strategy as:

$$p_t = conv2d(P_t), \tag{10}$$

where p_t is a low-dimensional dense vector representing the information from the items and user's feedback in page P_t as well as the spatial patterns of the item display strategy of page P_t.

Next, we feed $\{p_1, \cdots, p_T\}$ into another RNN with Gated Recurrent Units (GRU) to capture user's real-time preference in the current session. The architecture of this GRU is similar to the one in Section 2.2.1, but we utilize the final hidden state s^{ini} in Section 2.2.1 as the initial state in current GRU. Furthermore, to capture the user's real-time preference in the current session, we employ attention mechanism [2], which allows the RA to adaptively focus on and linearly combine different parts of the input sequence:

$$s^{cur} = \sum_{t=1}^{T} \alpha_t h_t, \tag{11}$$

where the weighted factors α_t determine which parts of the input sequence should be emphasized or ignored when making predictions. Here we leverage location-based attention mechanism [17] where the weighted factors are computed from the target hidden state h_t as follows:

$$\alpha_t = \frac{exp(W_\alpha h_t + b_\alpha)}{\sum_j exp(W_\alpha h_j + b_\alpha)}. \tag{12}$$

This GRU with attention mechanism is able to dynamically select more important input, which is helpful to capture the user's real-time preference in the current session. Note that - 1) the length of this GRU is flexible according to that of the current recommendation session. After each user-agent interaction, i.e., user browse one page of generated recommendations and give feedback to RA, we can add one more GRU unit, and use this page of items, corresponding categories and feedback as the input of the new GRU unit; 2) in fact, the two RNNs in Section 2.2.1 and Section 2.2.2 can be integrated into one RNN, we describe them separately to clearly illustrate their architecture, and clearly validate their effectiveness in Section 4.4.

2.2.3 ***Decoder for Action Generation Process.*** In this subsection, we will propose the action a^{cur} generation process, which generates (recommends) a new page of items to users. In other words, given user's current preference s^{cur}, we aim to recommend a page of items and displays them properly to maximize the reward. It is the inverse process of what the convolutional layer does. Hence, we use Deconvolution Neural Network (DeCNN) [35] to restore one page from the low-dimensional representation s^{cur}. It provides a sophisticated and automatic way to generate a page of recommendations with the corresponding display as:

$$a^{cur} = deconv2d(s^{cur}). \tag{13}$$

Note that - 1) the size of a^{cur} and P are different, since a^{cur} only contains item-embedding E_i, while P also contains item's category

embedding C_i and feedback-embedding F_i. For instance, if one recommendation page has h rows and w columns ($M = h \times w$), P is a $h \times w|X|$ matrix, while a^{cur} is a $h \times w|E|$ matrix; and 2) the generated item embeddings in a^{cur} may be not in the real item embedding set, thus we need to map them to valid item embeddings, which will be provided in later sections.

2.3 The Architecture of Critic Framework

The Critic is designed to leverage an approximator to learn an action-value function $Q(s, a)$, which is a judgment of whether the action (or a recommendation page) a generated by Actor matches the current state s. Note that we use "s" as s^{cur} in the last subsection for simplicity. Then, according $Q(s, a)$, the Actor updates its' parameters in a direction of improving performance to generate proper actions (or recommendations) in the following iterations.

Thus we need to feed user's current state s and action a (or a recommendation page) into the critic. To generate user's current state s, The RA follows the same strategy from Equation (3) to Equation (11), which uses embedding layers, 2D-CNN and GRU with attention mechanism to capture user's current preference. For action a, because a^{cur} generated in Equation (13) is a 2D matrix similar to an image, we utilize the same strategy in Equation (10), a 2D-CNN, to degrade a^{cur} into a low-dimensional dense vector a as:

$$a = conv2d(a^{cur}). \tag{14}$$

Then the RA concatenates current state s and action a, and feeds them into a Q-value function $Q(s, a)$. In real recommender systems, the state and action spaces are enormous, thus estimating the action-value function $Q(s, a)$ for each state-action pair is infeasible. In addition, many state-action pairs may not appear in the real trace such that it is hard to update their values. Therefore, it is more flexible and practical to use an approximator function to estimate the action-value function. In practice, the action-value function is usually highly nonlinear. Thus we choose Deep neural networks as approximators. In this work, we refer to a neural network approximator as deep Q-value function (DQN).

3 TRAINING AND TEST PROCEDURE

In this section, we discuss the training and test procedures. We propose two polices, i.e., online-policy and off-policy, to train and test the proposed framework based on online environment and offline historical data, respectively. Off-policy is necessary because the proposed framework should be pre-trained offline and be evaluated before launching them online to ensure the quality of the recommendations and mitigate possible negative impacts on user experience. After the offline stage, we can apply the framework online, and then the framework can continuously improve its strategies during the interactions with users.

3.1 The Training Procedure

As aforementioned in Section 2.2, we map user's preference s^{cur} to a new page of recommendations (a^{cur}). In a page of M items, a^{cur} contains item-embeddings of M items, i.e., $\{e_1, \cdots, e_M\}$. However, a^{cur} is a *proto-action*, because the generated item embedding $e_i \in a^{cur}$ may be not in the existing item-embedding space $\mathcal{I}$. Therefore, we need to map from proto-action a^{cur}_{pro} to *valid-action* $\mathbf{a}^{cur}_{val}$ where we have $\{\mathbf{e}_i \in \mathcal{I} | \forall \{\mathbf{e}_i \in \mathbf{a}^{cur}_{val}\}$. With this modification, an illustration of the proposed Actor-Critic recommending framework is demonstrated in Figure 5 where we omit Encoders for state generation part.

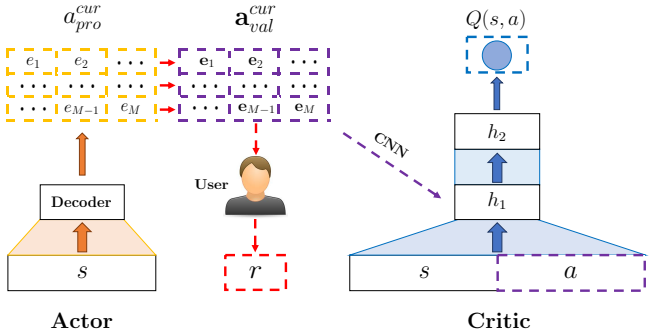

Figure 5: An illustration of the proposed framework.

3.1.1 ***Online Training Procedure.*** When we train the proposed framework in online environment, RA can interact with users by sequentially choosing recommendation items over a sequence of time steps. Thus, in online environment, the RA is able to receive real-time feedback for the recommended items from users. In this setting, for each e_i in a^{cur}_{pro}, we select the most similar $\mathbf{e}_i \in \mathcal{I}$ as the *valid item-embedding* in $\mathbf{a}^{cur}_{val}$. In this work, we select *cosine similarity* as:

$$\begin{aligned} \mathbf{e}_i &= \arg\max_{\mathbf{e} \in \mathcal{I}} \frac{e_i^\top \cdot \mathbf{e}}{\|e_i\| \|\mathbf{e}\|} \\ &= \arg\max_{\mathbf{e} \in \mathcal{I}} e_i^\top \cdot \frac{\mathbf{e}}{\|\mathbf{e}\|}. \end{aligned} \tag{15}$$

To decrease the computational cost of Equation (15), we pre-compute $\frac{\mathbf{e}}{\|\mathbf{e}\|}$ for all $\mathbf{e} \in \mathcal{I}$ and adopt item recalling mechanism to reduce the number of relevant items [2] .

Algorithm 1 Mapping Algorithm.

Input: User's browsing history s_h, item-embedding space $\mathcal{I}$, the size of recommendation page M.
Output: Valid recommendation page $\mathbf{a}^{cur}_{val}$.

1: Generate proto-action a^{cur}_{pro} according **Eq.(3)** to **Eq.(11)**
2: **for** $m = 1, M$ **do**
3: Select the most similar item as $\mathbf{e_m}$ according to **Eq.(15)**
4: Add item $\mathbf{e_m}$ into $\mathbf{a}^{cur}_{val}$ (at the same location as e_m in a^{cur}_{pro})
5: Remove item $\mathbf{e_m}$ from $\mathcal{I}$
6: **end for**
7: **return** $\mathbf{a}^{cur}_{val}$

We present the mapping algorithm in Algorithm 1. The Actor first generates proto-action a^{cur}_{pro} (line 1). For each e_m in a^{cur}_{pro}, the RA selects the most similar item in terms of cosine similarity (line

[2] In general, user's preference in current session should be related to user's last clicked/ordered items before the current session(say $\mathcal{L}$). Thus for each item in $\mathcal{L}$, we collect a number of most similar items in terms of cosine similarity from the whole item space, and combine all collected items as the initial item-embedding space $\mathcal{I}$ of current recommendation session. During the current session, when a user clicks or orders an item, we will also add its a number of most similar items into the item-embedding space $\mathcal{I}$.

3), and then adds this item into $\mathbf{a}_{val}^{cur}$ at the same position as e_m in a_{pro}^{cur} (line 4). Finally, the RA removes this item from the item-embedding space (line 5), which prevents recommending the same item repeatedly in one recommendation page.

Then the RA recommends the new recommendation page $\mathbf{a}_{val}^{cur}$ to user, and receives the immediate feedback (reward) from user. The reward r is the summation of rewards of all items in this page:

$$r = \sum_{m=1}^{M} reward(\mathbf{e_m}). \tag{16}$$

3.1.2 ***Offline Training Procedure.*** When we use user's historical browsing data to train the proposed Actor-Critic framework, user's browsing history, the new recommendation page $\mathbf{a}_{val}^{cur}$ and user's corresponding feedback (reward) r are given in the data. Thus, there is a gap between a_{pro}^{cur} and $\mathbf{a}_{val}^{cur}$, i.e., no matter what proto-action a_{pro}^{cur} outputted by the Actor, the valid-action $\mathbf{a}_{val}^{cur}$ is fixed. This will disconnect the Actor and the Critic.

From existing work [7, 14] and Section 3.1.1, we learn that a_{pro}^{cur} and $\mathbf{a}_{val}^{cur}$ should be similar, which is the prerequisite to connect the Actor and the Critic for training. Thus, we choose to minimize the difference between a_{pro}^{cur} and $\mathbf{a}_{val}^{cur}$:

$$\min_{\theta^\pi} \sum_{b=1}^{B} \left(\| a_{pro}^{cur} - \mathbf{a}_{val}^{cur} \|_F^2 \right), \tag{17}$$

where B is the batch size of samples in each iteration of SGD. Equation(17) updates Actor's parameters in the direction of pushing a_{pro}^{cur} and $\mathbf{a}_{val}^{cur}$ to be similar. In each iteration, given user's browsing history, the new recommendation page $\mathbf{a}_{val}^{cur}$, the RA generates proto-action a_{pro}^{cur} and then minimizes the difference between a_{pro}^{cur} and $\mathbf{a}_{val}^{cur}$, which can connect the Actor and the Critic. Next, we can follow conventional methods to update the parameters of Actor and Critic. The reward r is the summation of rewards of all items in page $\mathbf{a}_{val}^{cur}$.

3.1.3 ***Training Algorithm.*** In this work, we utilize DDPG [14] algorithm to train the parameters of the proposed Actor-Critic framework. The Critic can be trained by minimizing a sequence of loss functions $L(\theta^\mu)$ as:

$$L(\theta^\mu) = \mathbb{E}_{s,a,r,s'} \left[\left(r + \gamma Q_{\theta^{\mu'}}(s', f_{\theta^{\pi'}}(s')) - Q_{\theta^\mu}(s,a) \right)^2 \right], \tag{18}$$

where θ^μ represents all parameters in Critic. The critic is trained from samples stored in a replay buffer [20]. Actions stored in the replay buffer are generated by valid-action $\mathbf{a}_{val}^{cur}$, i.e., $a = conv2d(\mathbf{a}_{val}^{cur})$. This allows the learning algorithm to leverage the information of which action was actually executed to train the critic [7].

The first term $y = r + \gamma Q_{\theta^{\mu'}}(s', f_{\theta^{\pi'}}(s'))$ in Equation (18) is the target for the current iteration. The parameters from the previous iteration $\theta^{\mu'}$ are fixed when optimizing the loss function $L(\theta^\mu)$. In practice, it is often computationally efficient to optimize the loss function by stochastic gradient descent, rather than computing the full expectations in the above gradient. The derivatives of loss function $L(\theta^\mu)$ with respective to parameters θ^μ are presented as follows:

$$\begin{aligned} \nabla_{\theta^\mu} L(\theta^\mu) = \mathbb{E}_{s,a,r,s'} \big[& (r + \gamma Q_{\theta^{\mu'}}(s', f_{\theta^{\pi'}}(s')) \\ & - Q_{\theta^\mu}(s,a)) \nabla_{\theta^\mu} Q_{\theta^\mu}(s,a) \big]. \end{aligned} \tag{19}$$

We update the Actor using the policy gradient:

$$\nabla_{\theta^\pi} f_{\theta^\pi} \approx \mathbb{E}_s \left[\nabla_{\hat{a}} Q_{\theta^\mu}(s, \hat{a}) \nabla_{\theta^\pi} f_{\theta^\pi}(s) \right], \tag{20}$$

where $\hat{a} = f_{\theta^\pi}(s)$, i.e., $\hat{a}$ is generated by proto-action $\mathbf{a}_{pro}^{cur}$ ($\hat{a} = conv2d(a_{pro}^{cur})$). Note that proto-action $\mathbf{a}_{pro}^{cur}$ is the actual action outputted by Actor. This guarantees that policy gradient is taken at the actual output of policy f_{θ^π} [7].

The online training algorithm for the proposed framework DeepPage is presented in Algorithm 2. In each iteration, there are two stages, i.e., 1) transition generating stage (lines 7-10), and 2) parameter updating stage (lines 11-16). For transition generating stage (line 7): given the current state s_t, the RA first recommends a page of items a_t according to Algorithm 1 (line 8); then the RA observes the reward r_t and updates the state to s_{t+1} (lines 9); and finally the RA stores transitions (s_t, a_t, r_t, s_{t+1}) into the replay buffer $\mathcal{D}$ (line 10). For parameter updating stage (line 11): the RA samples minibatch of transitions (s, a, r, s') from $\mathcal{D}$ (line 12), and then updates parameters of Actor and Critic (lines 13-16) following a standard DDPG procedure [14].

Algorithm 2 Parameters Online Training for DeepPage with DDPG.

1: Initialize actor network f_{θ^π} and critic network Q_{θ^μ} with random weights
2: Initialize target network $f_{\theta^{\pi'}}$ and $Q_{\theta^{\mu'}}$ with weights $\theta^{\pi'} \leftarrow \theta^\pi, \theta^{\mu'} \leftarrow \theta^\mu$
3: Initialize the capacity of replay buffer $\mathcal{D}$
4: **for** *session* $= 1, G$ **do**
5: Receive initial observation state s_1
6: **for** $t = 1, T$ **do**
7: **Stage 1: Transition Generating Stage**
8: Select an action a_t according to **Alg.1** (policy f_{θ^π})
9: Execute action a_t and observe the reward r_t according to **Eq. (16)** and new state s_{t+1} according to **Section 2.2.2**
10: Store transition (s_t, a_t, r_t, s_{t+1}) in $\mathcal{D}$
11: **Stage 2: Parameter Updating Stage**
12: Sample minibatch of $\mathcal{N}$ transitions (s, a, r, s') from $\mathcal{D}$
13: Set $y = r + \gamma Q_{\theta^{\mu'}}(s', f_{\theta^{\pi'}}(s'))$
14: Update Critic by minimizing $\frac{1}{\mathcal{N}} \sum_n \left(y - Q_{\theta^\mu}(s,a) \right)^2$ according to **Eq. (19)**
15: Update Actor using the sampled policy gradient according to **Eq. (20)**
16: Update the target networks:
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$
$$\theta^{\pi'} \leftarrow \tau\theta^\pi + (1-\tau)\theta^{\pi'}$$
17: **end for**
18: **end for**

The offline training procedure is similar with Algorithm 2. The two differences are: 1) in line 8, offline training follows off-policy $b(s_t)$, and 2) before line 13, offline training first minimizes the difference between a_{pro}^{cur} and $\mathbf{a}_{val}^{cur}$ according to Equation (17).

In the algorithm, we introduce widely used techniques to train our framework. For example, we utilize a technique known as *experience replay* [15] (lines 12), and introduce separated evaluation

and target networks [19] (lines 2,16), which can help smooth the learning and avoid the divergence of parameters. For the soft updates of target networks (lines 16), we used $\tau = 0.01$. Moreover, we leverage *prioritized sampling strategy* [22] to assist the framework learning from the most important historical transitions.

3.2 The Test Procedure

After the training stage, the proposed framework learns parameters Θ^{π} and Θ^{μ}. Now we formally present the test procedure of the proposed framework DeepPage. We design two test methods, i.e., 1) Online test: to test DeepPage in online environment where RA interacts with users and receive real-time feedback for the recommended items from users, and 2) Offline test: to test DeepPage based on user's historical browsing data.

3.2.1 ***Online Test.*** The online test algorithm in one recommendation session is presented in Algorithm 3. The online test procedure is similar with the transition generating stage in Algorithm 2. In each iteration of the recommendation session, given the current state s_t, the RA recommends a page of recommendations a_t to user following policy $f_{\Theta^{\pi}}$ (line 4). Then the RA observes the reward r_t from user (line 5) and updates the state to s_{t+1} (line 6).

Algorithm 3 Online Test for DeepPage.

1: Initialize Actor with the trained parameters Θ^{π}
2: Receive initial observation state s_1
3: **for** $t = 1, T$ **do**
4: Execute an action a_t according to **Alg.1** (policy $f_{\Theta^{\pi}}$)
5: Observe the reward r_t from user according to **Eq. (16)**
6: Observe new state s_{t+1} according to **Section 2.2.2**
7: **end for**

3.2.2 ***Offline Test.*** The intuition of our offline test method is that, for a given recommendation session (offline data), the RA reranks the items in this session. If the proposed framework works well, the clicked/ordered items in this session will be ranked at the top of the new list. The reason why RA only reranks items in this session rather than items in the whole item space is that for the offline dataset, we only have the ground truth rewards of the existing items in this session. The offline test algorithm in one recommendation session is presented in Algorithm 4. In each iteration of an offline test recommendation session, given the state s_t (line 2), the RA recommends an page of recommendations a_t following policy $f_{\Theta^{\pi}}$ (lines 4). For each item e_i in a_t, we add it into new recommendation list $\mathcal{L}$ (line 6), and record e_i's reward r_i from user's historical browsing data (line 7). Then we can compute the overall reward r_t of a_t (line 9) and update the state to s_{t+1} (line 10). Finally, we remove all items e_i in a_t from the item set $\mathcal{I}$ of the current session (line 11).

4 EXPERIMENTS

In this section, we conduct extensive experiments with a dataset from a real e-commerce company to evaluate the effectiveness of the proposed framework. We mainly focus on two questions: (1) how the proposed framework performs compared to representative baselines; and (2) how the components in Actor and Critic contribute to the performance. We first introduce experimental settings. Then we seek answers to the above two questions. Finally, we study the impact of important parameters on the performance of the proposed framework.

Algorithm 4 Offline Test of DeepPage Framework.

Input: Item set $\mathcal{I} = \{\mathbf{e}_1, \cdots, \mathbf{e}_N\}$ and corresponding reward set $\mathcal{R} = \{r_1, \cdots, r_N\}$ of a session.
Output:Recommendation list $\mathcal{L}$ with new order

1: Initialize Actor with well-trained parameters Θ^{π}
2: Receive initial observation state s_1
3: **while** $|\mathcal{I}| > 0$ **do**
4: Select an action a_t according to **Alg.1** (policy $f_{\Theta^{\pi}}$)
5: **for** $e_i \in a_t$ **do**
6: Add e_i into the end of $\mathcal{L}$
7: Record reward r_i from user's historical browsing data
8: **end for**
9: Compute the overall reward r_t of a_t according to **Eq. (16)**
10: Execute action a_t and observe new state s_{t+1} according to **Section 2.2.2**
11: Remove all $e_i \in a_t$ from $\mathcal{I}$
12: **end while**

4.1 Experimental Settings

We evaluate our method on a dataset of September, 2017 from a real e-commerce company. We randomly collect 1,000,000 recommendation sessions (9,136,976 items) in temporal order, and use the first 70% sessions as the training/validation set and the later 30% sessions as the test set. For a new session, the initial state is collected from the previous sessions of the user. In this work, we leverage $N = 10$ previously clicked/ordered items to generate the initial state. Each time the RA recommends a page of $M = 10$ items (5 rows and 2 columns) to users [3]. The reward r of one skipped/clicked/ordered item is empirically set as 0, 1, and 5, respectively. The dimensions of item-embedding/ category-embedding/ feedback-embedding are $|E| = 50$, $|C| = 35$, and $|F| = 15$. We set the discounted factor $\gamma = 0.95$, and the rate for soft updates of target networks $\tau = 0.01$. For the parameters of the proposed framework such as γ, we select them via cross-validation. Correspondingly, we also do parameter-tuning for baselines for a fair comparison. We will discuss more details about parameter selection for the proposed framework in the following subsections.

For online test, we leverage the summation of all rewards in one recommendation session as the metric. For offline test, we select **Precision@20**, **Recall@20**, **F1-score@20** [9], **NDCG@20** [11] and **MAP** [33], as the metrics to measure the performance. The difference of ours from traditional Learn-to-Rank methods is that we rank both clicked and ordered items together, and set them by different rewards, rather than only rank clicked items as that in the Learn-to-Rank setting.

[3]This is based on offline historical data collected from mobile Apps, i.e., to fit the screen size of mobile phones, one page has only 5 rows and 2 columns.

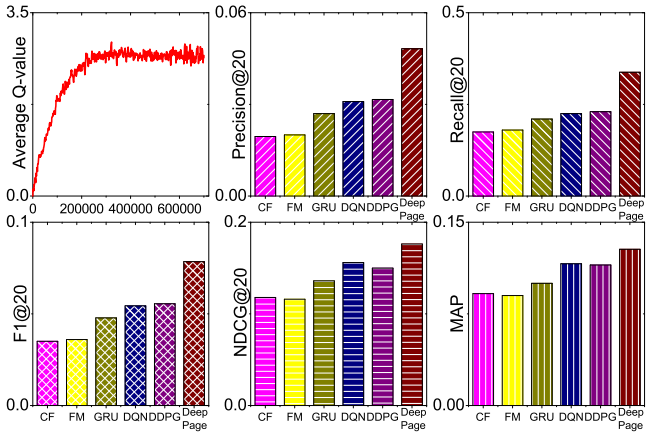

Figure 6: Overall performance comparison in offline test.

4.2 Performance Comparison for Offline Test

To answer the first question, we compare the proposed framework with the following representative baseline methods:

- **CF** [4]: Collaborative filtering is a method of making automatic predictions about the interests of a user by collecting preference information from many users, which is based on the hypothesis that people often get the best recommendations from someone with similar tastes to themselves.
- **FM** [24]: Factorization Machines combine the advantages of support vector machines with factorization models. Compared with matrix factorization, higher order interactions can be modeled using the dimensionality parameter.
- **GRU** [10] : This baseline utilizes the Gated Recurrent Units (GRU) to predict what user will click/order next based on the browsing histories. To make a fair comparison, it also keeps previous $N = 10$ clicked/ordered items as initial states.
- **DQN** [19]: We use a Deep Q-network with five fully-connected layers in this baseline. The input is the concatenation of embeddings of users' historical clicked/ordered items (state) and a page of recommendations (action), and train this baseline by Eq.(1).
- **DDPG** [7]: In this baseline, we use conventional Deep Deterministic Policy Gradient with five fully connected layers in both Actor and Critic. The input for Actor is the concatenation of embeddings of users' historical clicked/ordered items (state). The input for Critic is the concatenation of embeddings of state and a page of recommendations (action).

We leverage offline training strategy to train DDPG and DeepPage as mentioned in Section 3.1.1. The results are shown in Figure 6. We make following observations:

- Figure 6 (a) illustrates the training process of DeepPage. We can observe that the framework approaches to convergence when the model are trained by 500,000 offline sessions.
- CF and FM perform worse than other baselines. These two baselines ignore the temporal sequence of the users' browsing history, while GRU can capture the temporal sequence, and DQN, DDPG and DeepPage are able to continuously update their strategies during the interactions.
- DQN and DDPG outperform GRU. We design GRU to maximize the immediate reward for recommendations, while DQN and DDPG are designed to achieve the trade-off between short-term and long-term rewards. This result suggests that introducing reinforcement learning can improve the performance of recommendations.
- DeepPage performs better than conventional DDPG. Compared to DDPG, DeepPage jointly optimizes a page of items and uses GRU to learn user's real-time preference. More details about the impact of mode components on DeepPage will be discussed in the following subsection.

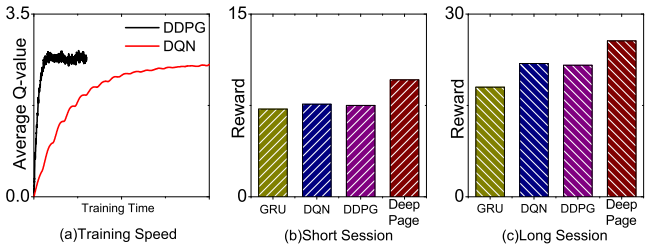

Figure 7: Overall performance comparison in online test.

4.3 Performance Comparison for Online Test

Following [7], we build a simulated online environment (adapted to our case) for online test. We compare DeepPage with GRU, DQN and DDPG. Note that we do not include **CF** and **FM** baselines since CF and FM are not applicable to the online environment.

Here we utilize online training strategy to train DDPG and DeepPage (both Actor-Critic framework) as mentioned in Section 3.1.2. Baselines are also applicable to be trained via the rewards generated by simulated online environment. Note that we use data different from the training set to build the simulated online environment.

As the test stage is based on the simulator, we can artificially control the length of recommendation sessions to study the performance in short and long sessions. We define short sessions with 10 recommendation pages, while long sessions with 50 recommendation pages. The results are shown in Figure 7. It can be observed:

- DDPG performs similar to DQN, but the training speed of DDPG is much faster than DQN, as shown in Figure 7 (a). DQN computes Q-value for all potential actions, while DDPG can reduce this redundant computation. This result indicates that Actor-Critic framework is suitable for practical recommender systems with enormous action space.
- In short recommendation sessions, GRU, DQN and DDPG achieve comparable performance. In other words, GRU model and reinforcement learning models like DQN and DDPG can recommend proper items matching users' short-term interests.
- In long recommendation sessions, DQN and DDPG outperform GRU significantly. GRU is designed to maximize the immediate reward for recommendations, while reinforcement learning models like DQN and DDPG are designed to achieve the trade-off between short-term and long-term rewards. This result suggests that introducing reinforcement learning can improve the performance of recommendations.
- DeepPage performs better than conventional DQN and DDPG. DeepPage can learn user's real-time preference and optimizes a page of items. We detail the effect of model components of DeepPage in the following subsection.

Table 1: Performance comparison of different components.

Methods	Precision @20	Recall @20	F1score @20	NDCG @20	MAP
DeepPage-1	0.0479	0.3351	0.0779	0.1753	0.1276
DeepPage-2	0.0475	0.3308	0.0772	0.1737	0.1265
DeepPage-3	0.0351	0.2627	0.0578	0.1393	0.1071
DeepPage-4	0.0452	0.3136	0.0729	0.1679	0.1216
DeepPage-5	0.0476	0.3342	0.0775	0.1716	0.1243
DeepPage-6	0.0318	0.2433	0.0528	0.1316	0.1039
DeepPage-7	0.0459	0.3179	0.0736	0.1698	0.1233
DeepPage	**0.0491**	**0.3576**	**0.0805**	**0.1872**	**0.1378**

4.4 Effectiveness of Components

To validate the effectiveness of each component, we systematically eliminate the corresponding model components by defining following variants of DeepPage:

- DeepPage-1: This variant is to evaluate the performance of the embedding layer. We remove the embedding layer for items, categories and feedback.
- DeepPage-2: In this variant, we evaluate the contribution of category and feedback information, hence, this variant does not contain one-hot indicator vectors of category and feedback.
- DeepPage-3: This variant is to evaluate the effectiveness of GRU to generate initial state, so we eliminate the GRU in Figure 3.
- DeepPage-4: In this variant, we evaluate the contribution of CNNs as shown in Figure 4, thus we remove CNNs and directly feed the outputs of embedding layers (concatenate embeddings of all items as one vector) into GRU units.
- DeepPage-5: This variant is to evaluate the effectiveness of attention mechanism in Figure 4, therefore, we eliminate attention layer and use the hidden state of last GRU unit as the input of DeCNN.
- DeepPage-6: In this variant, we evaluate the GRU to generate local state, thereby, we remove this GRU in Figure 4 and concatenate outputs of all CNNs as a vector, and feed it into DeCNN.
- DeepPage-7: This variant is to evaluate the performance of DeCNN to generate a new page of items, hence, we replace it by fully-connected layers, which output a concatenated vector of M item-embeddings.

The offline results are shown in Table 1 (we omit similar online observations because of the space limitation). We make following observations:

- DeepPage-1 and DeepPage-2 validate that incorporating category/feedback information and the embedding layer can boost the performance of recommendation.
- DeepPage-3 and DeepPage-6 perform worse, which suggests that setting user's initial preference at the beginning of a new recommendation session, and capturing user's real-time preference in current session is helpful for accurate recommendations.
- DeepPage-5 proves that incorporating attention mechanism can better capture user's real-time preference than only GRU.
- DeepPage outperforms DeepPage-4 and DeepPage-7, which indicates that item display strategy can influence the decision making process of users.

In a nutshell, DeepPage outperforms all its variants, which demonstrates the effectiveness of each component for recommendations.

5 RELATED WORK

In this section, we briefly review research related to our study. In general, the related work can be mainly grouped into the following categories.

The first category related to this paper is traditional recommendation techniques. Recommender systems assist users by supplying a page of items that might interest users. Efforts have been made on offering meaningful recommendations to users. Collaborative filtering[16] is the most successful and the most widely used technique, which is based on the hypothesis that people often get the best recommendations from someone with similar tastes to themselves[4]. Another common approach is content-based filtering [21], which tries to recommend items with similar properties to those that a user ordered in the past. Knowledge-based systems[1] recommend items based on specific domain knowledge about how certain item features meet users' needs and preferences and how the item is useful for the user. Hybrid recommender systems are based on the combination of the above mentioned two or more types of techniques[5]. The other topic closely related to this category is deep learning based recommender system, which is able to effectively capture the non-linear and non-trivial user-item relationships, and enables the codification of more complex abstractions as data representations in the higher layers[37]. For instance, Nguyen et al.[23] proposed a personalized tag recommender system based on CNN. It utilizes constitutional and max-pooling layer to get visual features from patches of images. Wu et al.[34] designed a session-based recommendation model for real-world e-commerce website. It utilizes the basic RNN to predict what user will buy next based on the click histories.

The second category is about reinforcement learning for recommendations and personalization, which is different from the traditional item recommendations. In this work, we consider the recommending procedure as sequential interactions between users and the recommender agent; and leverage deep reinforcement learning to automatically learn the optimal recommendation strategies. Indeed, reinforcement learning has been widely examined in the recommendation field. The MDP-Based CF model in Shani et al.[27] can be viewed as approximating a partial observable MDP (POMDP) by using a finite rather than unbounded window of past history to define the current state. Furthermore, Mahmood et al.[18] adopted the reinforcement learning technique to observe the responses of users in a conversational recommender, with the aim to maximize a numerical cumulative reward function modeling the benefit that users get from each recommendation session. Taghipour et al.[31, 32] modeled web page recommendation as a Q-Learning problem and learned to make recommendations from web usage data as the actions rather than discovering explicit patterns from the data. The system inherits the intrinsic characteristic of reinforcement learning which is in a constant learning process. Sunehag et al.[29] introduced agents that successfully address sequential decision problems with high-dimensional combinatorial slate-action spaces. Zheng et al.[38] proposed a reinforcement learning framework to do online personalized news recommendation, which can

effectively model the dynamic news features and user preferences, and plan for future explicitly, in order to achieve higher reward (e.g., CTR) in the long run. Feng et al.[8] presented a multi-agent reinforcement learning model, MA-RDPG which can optimize ranking strategies collaboratively for multi-scenario ranking problems. Cai et al.[6] employed a reinforcement mechanism design framework for solving the impression allocation problem of large e-commerce websites, while taking the rationality of sellers into account.

6 CONCLUSION

In this paper, we propose a novel page-wise recommendation framework DeepPage, which leverages Deep Reinforcement Learning to automatically learn the optimal recommendation strategies and optimizes a page of items simultaneously. Reinforcement learning based recommender systems have two advantages: (1) they can continuously update strategies during the interactions, and (2) they are able to learn a strategy that maximizes the long-term cumulative reward from users. Different from previous work, we propose a novel Actor-Critic framework, which can be applied in scenarios with large and dynamic item space and can reduce redundant computation significantly. Furthermore, the proposed framework is able to capture the real-time preference as well as optimize a page of items jointly, which can boost recommendation performance. We evaluate our framework with extensive experiments based on data from a real e-commerce company. The results show that (1) DeepPage can improve the recommendation performance; and (2) capturing users' real-time preference and jointly optimizing a page is helpful for accurate recommendation.

There are several interesting research directions. First, in addition to solving one task like recommendation in e-commence, we would like to handle multiple tasks such as search, bidding, advertisement and recommendation collaboratively in one reinforcement learning framework. Second, we would like to validate with more agent-user interaction patterns, e.g., adding items into shopping cart, and investigate how to model them mathematically for recommendations. Finally, the framework proposed in the work is quite general, and we would like to investigate more applications of the proposed framework.

REFERENCES

[1] Rajendra Akerkar and Priti Sajja. 2010. *Knowledge-based systems*. Jones & Bartlett Publishers.

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473* (2014).

[3] Richard Bellman. 2013. *Dynamic programming*. Courier Corporation.

[4] John S Breese, David Heckerman, and Carl Kadie. 1998. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*. Morgan Kaufmann Publishers Inc., 43–52.

[5] Robin Burke. 2002. Hybrid recommender systems: Survey and experiments. *User modeling and user-adapted interaction* 12, 4 (2002), 331–370.

[6] Qingpeng Cai, Aris Filos-Ratsikas, Pingzhong Tang, and Yiwei Zhang. 2018. Reinforcement Mechanism Design for Fraudulent Behaviour in e-Commerce. (2018).

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

[8] Jun Feng, Heng Li, Minlie Huang, Shichen Liu, Wenwu Ou, Zhirong Wang, and Xiaoyan Zhu. 2018. Learning to Collaborate: Multi-Scenario Ranking via Multi-Agent Reinforcement Learning. (2018).

[9] Asela Gunawardana and Guy Shani. 2009. A survey of accuracy evaluation metrics of recommendation tasks. *Journal of Machine Learning Research* 10, Dec (2009), 2935–2962.

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

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

[12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. 1097–1105.

[13] Omer Levy and Yoav Goldberg. 2014. Neural word embedding as implicit matrix factorization. In *Advances in neural information processing systems*. 2177–2185.

[14] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2015. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971* (2015).

[15] Long-Ji Lin. 1993. *Reinforcement learning for robots using neural networks*. Technical Report. Carnegie-Mellon Univ Pittsburgh PA School of Computer Science.

[16] Greg Linden, Brent Smith, and Jeremy York. 2003. Amazon. com recommendations: Item-to-item collaborative filtering. *IEEE Internet computing* 7, 1 (2003), 76–80.

[17] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. 2015. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025* (2015).

[18] Tariq Mahmood and Francesco Ricci. 2009. Improving recommender systems with adaptive conversational strategies. In *Proceedings of the 20th ACM conference on Hypertext and hypermedia*. ACM, 73–82.

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

[20] 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.

[21] Raymond J Mooney and Loriene Roy. 2000. Content-based book recommending using learning for text categorization. In *Proceedings of the fifth ACM conference on Digital libraries*. ACM, 195–204.

[22] Andrew W Moore and Christopher G Atkeson. 1993. Prioritized sweeping: Reinforcement learning with less data and less time. *Machine learning* 13, 1 (1993), 103–130.

[23] Hanh TH Nguyen, Martin Wistuba, Josif Grabocka, Lucas Rego Drumond, and Lars Schmidt-Thieme. 2017. Personalized Deep Learning for Tag Recommendation. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, 186–197.

[24] Steffen Rendle. 2010. Factorization machines. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on*. IEEE, 995–1000.

[25] Paul Resnick and Hal R Varian. 1997. Recommender systems. *Commun. ACM* 40, 3 (1997), 56–58.

[26] Francesco Ricci, Lior Rokach, and Bracha Shapira. 2011. Introduction to recommender systems handbook. In *Recommender systems handbook*. Springer, 1–35.

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

[28] Ramakrishnan Srikant, Sugato Basu, Ni Wang, and Daryl Pregibon. 2010. User browsing models: relevance versus examination. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 223–232.

[29] Peter Sunehag, Richard Evans, Gabriel Dulac-Arnold, Yori Zwols, Daniel Visentin, and Ben Coppin. 2015. Deep Reinforcement Learning with Attention for Slate Markov Decision Processes with High-Dimensional States and Actions. *arXiv preprint arXiv:1512.01124* (2015).

[30] Richard S Sutton and Andrew G Barto. 1998. *Reinforcement learning: An introduction*. Vol. 1. MIT press Cambridge.

[31] Nima Taghipour and Ahmad Kardan. 2008. A hybrid web recommender system based on q-learning. In *Proceedings of the 2008 ACM symposium on Applied computing*. ACM, 1164–1168.

[32] Nima Taghipour, Ahmad Kardan, and Saeed Shiry Ghidary. 2007. Usage-based web recommendations: a reinforcement learning approach. In *Proceedings of the 2007 ACM conference on Recommender systems*. ACM, 113–120.

[33] Andrew Turpin and Falk Scholer. 2006. User performance versus precision measures for simple search tasks. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, 11–18.

[34] Sai Wu, Weichao Ren, Chengchao Yu, Gang Chen, Dongxiang Zhang, and Jingbo Zhu. 2016. Personal recommendation using deep recurrent neural networks in NetEase. In *Data Engineering (ICDE), 2016 IEEE 32nd International Conference on.*

IEEE, 1218–1229.

[35] Li Xu, Jimmy SJ Ren, Ce Liu, and Jiaya Jia. 2014. Deep convolutional neural network for image deconvolution. In *Advances in Neural Information Processing Systems*. 1790–1798.

[36] Yisong Yue and Carlos Guestrin. 2011. Linear submodular bandits and their application to diversified retrieval. In *Advances in Neural Information Processing Systems*. 2483–2491.

[37] Shuai Zhang, Lina Yao, and Aixin Sun. 2017. Deep Learning based Recommender System: A Survey and New Perspectives. *arXiv preprint arXiv:1707.07435* (2017).

[38] Guanjie Zheng, Fuzheng Zhang, Zihan Zheng, Yang Xiang, Nicholas Jing Yuan, Xing Xie, and Zhenhui Li. 2018. DRN: A Deep Reinforcement Learning Framework for News Recommendation. (2018).