Reinforcement Learning to Optimize Long-term User Engagement in Recommender Systems

Lixin Zou[*]
Tsinghua University
zoulx15@mails.tsinghua.edu.cn

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

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

Jiaxing Song
Tsinghua University
jxsong@mail.tsinghua.edu.cn

Weidong Liu
Tsinghua University
liuwd@mail.tsinghua.edu.cn

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

ABSTRACT

Recommender systems play a crucial role in our daily lives. Feed streaming mechanism has been widely used in the recommender system, especially on the mobile Apps. The feed streaming setting provides users the interactive manner of recommendation in never-ending feeds. In such a manner, a good recommender system should pay more attention to user stickiness, which is far beyond classical instant metrics and typically measured by **long-term user engagement**. Directly optimizing long-term user engagement is a non-trivial problem, as the learning target is usually not available for conventional supervised learning methods. Though reinforcement learning (RL) naturally fits the problem of maximizing the long term rewards, applying RL to optimize long-term user engagement is still facing challenges: user behaviors are versatile to model, which typically consists of both instant feedback (*e.g.,* clicks) and delayed feedback (*e.g.,* dwell time, revisit); in addition, performing effective off-policy learning is still immature, especially when combining bootstrapping and function approximation.

To address these issues, in this work, we introduce a RL framework — FeedRec to optimize the long-term user engagement. FeedRec includes two components: 1) a Q-Network which designed in hierarchical LSTM takes charge of modeling complex user behaviors, and 2) a S-Network, which simulates the environment, assists the Q-Network and voids the instability of convergence in policy learning. Extensive experiments on synthetic data and a real-world large scale data show that FeedRec effectively optimizes the long-term user engagement and outperforms state-of-the-arts.

CCS CONCEPTS

• **Information systems** → **Recommender systems**; *Personalization*; • **Theory of computation** → *Sequential decision making*.

KEYWORDS

Reinforcement learning; Long-term user engagement; Recommender system

1 INTRODUCTION

Recommender systems assist user in information-seeking tasks by suggesting goods (*e.g.,* products, news, services) that best match users' needs and preferences. In recent feed streaming scenarios, users are able to constantly browse items generated by the never-ending feeds, such as the news streams in Yahoo News[1], the social streams in Facebook[2], and the product streams in Amazon[3]. Specifically, interacting with the product streams, the users could click on the items and view the details of the items. Meanwhile, (s)he could also skip unattractive items and scroll down, and even leave the recommender system due to the appearance of many redundant or uninterested items. Under such circumstance, optimizing clicks will not be the only golden rule anymore. It is critical to maximizing the users' satisfaction of interactions with the feed streams, which falls in two folds: instant engagement, *e.g.,* click, purchase; long-term engagement, say stickiness, typically representing users' desire to stay with the streams longer and open the streams repeatedly [11].

However, most traditional recommender systems only focus on optimizing instant metrics (*e.g.,* click through rate [12], conversion rate[19]). Moving more deeply with interaction, a good feed streaming recommender system should be able to not only bring about higher click through rate but also keep users actively interacting with the system, which typically is measured by long-term delayed metrics. Delayed metrics usually are more complicated, including dwell time on the Apps, depth of the page-viewing, the internal time between two visits, and so on. Unfortunately, due to the difficulty of modeling delayed metrics, directly optimizing the delayed metrics is very challenging. While only a few preliminary work[28] starts investigating the optimization of some long-term/delayed

---

metrics, a systematical solution to optimize the overall engagement metrics is wanted.

Intuitively, reinforcement learning (RL), which was born to maximize long-term rewards, could be a unified framework to optimize the instant and long-term user engagement. Applying RL to optimize long-term user engagement itself is a non-trivial problem. As mentioned, the long-term user engagement is very complicated (*i.e.,* measured in versatile behaviors, *e.g.,* dwell time, revisit), and would require a very large number of environment interactions to model such long term behaviors and build a recommendation agent effectively. As a result, building a recommender agent from scratch through real online systems would be prohibitively expensive, since numerous interactions with immature recommendation agent will harm user experiences, even annoy the users. An alternative is to build a recommender agent offline through making use of the logged data, where the off-policy learning methods can mitigate the cost of the trial-and-error search. Unfortunately, current methods including Monte Carlo (MC) and temporal-difference (TD) have limitations for offline policy learning in realistic recommender systems: MC-based methods suffer from the problem of high variance, especially when facing enormous action space (*e.g.,* billions of candidate items) in real-world applications; TD-based methods improve the efficiency by using bootstrapping techniques in estimation, which, however, is confronted with another notorious problem called *Deadly Triad* (*i.e.,* the problem of instability and divergence arises whenever combining function approximation, bootstrapping, and offline training [24]). Unfortunately, state-of-the-art methods [33, 34] in recommender systems, which are designed with neural architectures, will encounter inevitably the *Deadly Triad* problem in offline policy learning.

To overcome the aforementioned issues of complex behaviors and offline policy learning, we here propose an RL-based framework, named FeedRec, to improve long-term user engagement in recommender systems. Specifically, we formalize the feed streaming recommendation as a *Markov decision process* (MDP), and design a Q-Network to directly optimize the metrics of user engagement. To avoid the problem of instability of convergence in offline Q-Learning, we further introduce a S-Network, which simulates the environments, to assist the policy learning. In Q-Network, to capture the information of versatile user long term behaviors, a fine user behavior chain is modeled by LSTM, which consists of all rough behaviors, *e.g.,* click, skip, browse, ordering, dwell, revisit, *etc.* When modeling such fine-grained user behaviors, two problems emerges: the numbers for specific user actions is extremely imbalanced (*i.e.,* clicks is much fewer than skips)[37]; and long-term user behavior is more complicated to represent. We hence further integrated hierarchical LSTM with temporal cell into Q-Network to characterize fine-grained user behaviors.

On the other hand, in order to make effective use of the historical logged data and avoid the *Deadly Triad* problem in offline Q-Learning, we introduce an environment model, called S-network, to simulate the environment and generate simulated user experiences, assisting offline policy learning. We conduct extensive experiments on both the synthetic dataset and a real-world E-commerce dataset. The experimental results show the effectiveness of the proposed algorithm over the state-of-the-art baselines for optimizing long user engagement.

Contributions can be summarized as follow:

(1) We propose a reinforcement learning model — FeedRec to directly optimize the user engagement (both instant and long term user engagement) in feed streaming recommendation.
(2) To model versatile user behaviors, which typically includes both instant engagement (*e.g.,* click and order) and long term engagement (*e.g.,* dwell time, revisit, *etc*), Q-Network with hierarchical LSTM architecture is presented.
(3) To ensure convergence in off-policy learning, an effective and safe training framework is designed.
(4) The experimental results show that our proposed algorithms outperform the state-of-the-art baseline.

2 RELATED WORK

2.1 Traditional recommender system

Most of the existing recommender systems try to balance the instant metrics and factors, *i.e.,* the diversity, the novelty in recommendations. From the perspective of the instant metrics, there are numerous works focusing on improving the users' implicit feedback clicks [10, 12, 27], explicit ratings [3, 17, 21], and dwell time on recommended items [30]. In fact, the instant metrics have been criticized to be insufficient to measure and represent real engagement of users. As the supplementary, methods [1, 2, 4] intended to enhance user's satisfaction through recommending diverse items have been proposed. However, all of these works can not model the iterative interactions with users. Furthermore, none of these works could directly optimize delayed metrics of long-term user engagement.

2.2 Reinforcement learning based recommender system

Contextual bandit solutions are proposed to model the interaction with users and handle the notorious explore/exploit dilemma in online recommendation [8, 13, 20, 26, 31]. On one hand, these contextual bandit settings assume that the user's interests remain the same or smoothly drift which can not hold under the feed streaming mechanism. On the other hand, although Wu et al. [28] proposed to optimize the delayed revisiting time, there is no systematical solution to optimizing delayed metrics for user engagement. Apart from contextual bandits, a series of MDP based models [5, 14, 15, 23, 32, 35, 39] are proposed in recommendation task. Arnold et al. [5] proposed a modified DDPG model to deal with the problem of large discrete action spaces. Recently, Zhao et al. combined pagewise, pairwise ranking technologies with reinforcement learning[33, 34]. Since only the instant metrics are considered, the above methods fail to optimize delayed metrics of user engagement. In this paper, we proposed a systematically MDP-based solution to track user's interests shift and directly optimize both instant metrics and delayed metrics of user engagement.

3 PROBLEM FORMULATION

3.1 Feed Streaming Recommendation

In the feed streaming recommendation, the recommender system interacts with a user $u \in \mathcal{U}$ at discrete time steps. At each time step t, the agent feeds an item i_t and receives a feedback f_t from the

user, where $i_t \in \mathcal{I}$ is from the recommendable item set and $f_t \in \mathcal{F}$ is user's feedback/bevahior on i_t, including clicking, purchasing, or skipping, leaving, *etc*. The interaction process forms a sequence $X_t = \{u, (i_1, f_1, d_1), \ldots, (i_t, f_t, d_t)\}$ with d_t as the dwell time on the recommendation, which indicates user's preferences on the recommendation. Given X_t, the agent needs to generate the i_{t+1} for next-time step with the goal of maximizing long term user engagement, *e.g.*, the total clicks or browsing depth. In this work, we focus on how to improving the expected quality of all items in feed streaming scenario.

3.2 MDP Formulation of Feed Streams

A MDP is defined by $M = \langle S, A, P, R, \gamma \rangle$, where S is the state space, A is the action space, $P : S \times A \times S \to \mathbb{R}$ is the transition function, $R : S \times A \to \mathbb{R}$ is the mean reward function with $r(s, a)$ being the immediate goodness of (s, a), and $\gamma \in [0, 1]$ is the discount factor. A (stationary) policy $\pi : S \times A \to [0, 1]$ assigns each state $s \in S$ a distribution over actions, where $a \in A$ has probability $\pi(a|s)$. In feed streaming recommendation, $\langle S, A, P \rangle$ are set as follow:

- **State** S is a set of states. We design the state at time step t as the browsing sequence $s_t = X_{t-1}$. At the beginning, $s_1 = \{u\}$ just contains user's information. At time step t, $s_t = s_{t-1} \oplus \{(i_{t-1}, f_{t-1}, d_{t-1})\}$ is updated with the old state s_{t-1} concentrated with the tuple of recommended item, feedback and dwell time $(i_{t-1}, f_{t-1}, d_{t-1})$.
- **Action** A is a finite set of actions. The actions available depends on the state s, denoted as $A(s)$. The $A(s_1)$ is initialized with all recalled items. $A(s_t)$ is updated by removing recommended items from $A(s_{t-1})$ and action a_t is the recommending item i_t.
- **Transition** P is the transition function with $p(s_{t+1}|s_t, i_t)$ being the probability of seeing state s_{t+1} after taking action i_t at s_t. In our case, the uncertainty comes from user's feedback f_t *w.r.t.* i_t and s_t.

3.3 User Engagement and Reward Function

As aforementioned, unlike traditional recommendation, instant metrics (click, purchase, *etc*) are not the only measurements of the user engagement/satisfactory, and long term engagement is even more important, which is often measured in delayed metrics, *e.g.*, browsing depth, user revisits and dwells time on the system. Reinforcement learning provides a way to directly optimize both instant and delayed metrics through the designs of reward functions.

The reward function $R : S \times A \to \mathbb{R}$ can be designed in different forms. We here instantiate it linearly by assuming that user engagement reward $r_t(\boldsymbol{m}_t)$ at each step t is in the form of weighted sum of different metrics:

$$r_t = \boldsymbol{\omega}^\top \boldsymbol{m}_t, \tag{1}$$

where $\boldsymbol{m}_t$ is a column vector consisted of different metrics, $\boldsymbol{\omega}$ is the weight vector. Next, we give some instantiations of reward function *w.r.t.* both instant metrics and delayed metrics.

Instant metrics. In the instant user engagement, we can have clicks, purchase (in e-commerce), *etc*. The shared characteristics of instant metrics are that these metrics are triggered instantly by the current action. We here take click as an example, the number of clicks in t-th feedback is defined as the metric for click m_t^c,

$$m_t^c = \#clicks(f_t).$$

Delayed metrics. The delayed metrics include browsing depth, dwell time on the system, user revisit, *etc*. Such metrics are usually adopted for measuring long-term user engagement. The delayed metrics are triggered by previous behaviors, some of which even hold long-term dependency. We here provide two example reward functions for delayed metrics:

Depth metric. The depth of browsing is a special indicator that the feed streaming scenario differs from other types of recommendation due to the infinite scroll mechanism. After viewing the t-th feed, the system should reward this feed if the user remained in the system and scrolled down. Intuitively, the metric of depth m_t^d can be defined as:

$$m_t^d = \#scans(f_t)$$

where $\#scans(f_t)$ is the number of scans in the t-th feedback.

Return time metric. The user will use the system more often when (s)he is satisfied with the recommended items. Thus, the interval time between two visits can reflect the user's satisfaction with the system. The return time m_t^r can be designed as the reciprocal of time:

$$m_t^r = \frac{\beta}{v^r},$$

where v^r represents the time between two visits and β is the hyper-parameter.

From the above examples—click metric, depth metric and return time metric, we can clearly see $\boldsymbol{m}_t = [m_t^c, m_t^d, m_t^r]^\top$. Note that in MDP setting, cumulative rewards will be maximized, that is, we are actually optimizing total browsing depth, and frequency of visiting in the future, which typically are long term user engagement.

4 POLICY LEARNING FOR RECOMMENDER SYSTEMS

To estimate the future reward (*i.e.*, the future user stickiness), the expected long-term user engagement for recommendation i_t is presented with the Q-value as,

$$Q^\pi(s_t, i_t) = \mathbb{E}_{i_k \sim \pi}[\underbrace{r_t}_{\text{current rewards}} + \underbrace{\sum_{k=1}^{T-t} \gamma^k r_{t+k}}_{\text{future rewards}}], \tag{2}$$

where γ is the discount factor to balance the importance of the current rewards and future rewards. The optimal $Q^*(s_t, i_t)$, having the maximum expected reward achievable by the optimal policy, should follow the optimal Bellman equation [24] as,

$$Q^*(s_t, i_t) = \mathbb{E}_{s_{t+1}} \left[r_t + \gamma \max_{i'} Q^* (s_{t+1}, i') | s_t, i_t \right]. \tag{3}$$

Given the Q^*, the recommendation i_t is chosen with the maximum $Q^*(s_t, i_t)$. Nevertheless, in real-world recommender systems, with enormous users and items, estimating the action-value function $Q^*(s_t, i_t)$ for each state-action pairs is infeasible. Hence, it is more flexible and practical to use function approximation, *e.g.*, neural networks, to estimate the action-value function, *i.e.*, $Q^*(s_t, i_t) \approx Q(s_t, i_t; \theta_q)$. In practice, neural networks are excellent to track user's interests in recommendation [10, 12, 36]. In this paper, we refer to a neural network function approximator with
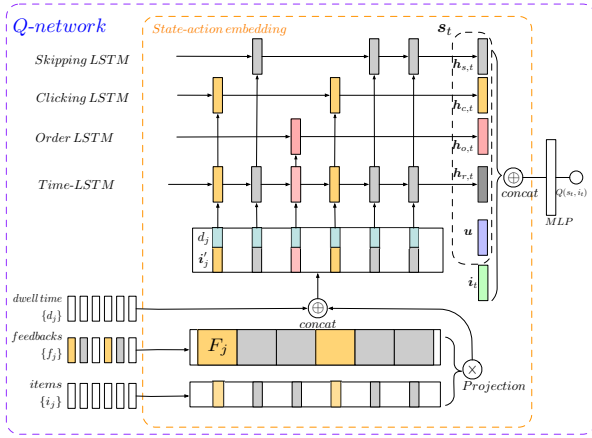
Figure 1: The architecture of Q-Network.

parameter θ_q as a Q-Network. The Q-Network can be trained by minimizing the mean-squared loss function, defined as follows:

$$
\begin{aligned}
\ell(\theta_q) &= \mathbb{E}_{(s_t, i_t, r_t, s_{t+1}) \sim \mathcal{M}} \left[(y_t - Q(s_t, i_t; \theta_q))^2 \right] \quad (4) \\
y_t &= r_t + \gamma \max_{i_{t+1} \in I} Q(s_{t+1}, i_{t+1}; \theta_q),
\end{aligned}
$$

where $\mathcal{M} = \{(s_t, i_t, r_t, s_{t+1})\}$ is a large replay buffer storing the past feeds, from which samples are taken in mini-batch training. By differentiating the loss function with respect to θ_q, we arrive at the following gradient:

$$
\begin{aligned}
\nabla_{\theta_q} \ell(\theta_q) &= \mathbb{E}_{(s_t, i_t, r_t, s_{t+1}) \sim \mathcal{M}} \left[(r + \gamma \max_{i_{t+1}} Q(s_{t+1}, i_{t+1}; \theta_q) \right. \\
&\quad \left. - Q(s_t, i_t; \theta_q)) \nabla_{\theta_q} Q(s_t, i_t; \theta_q) \right] \quad (5)
\end{aligned}
$$

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.

4.1 The Q-Network

The design of Q-Network is critical to the performances. In long term user engagement optimization, the user interactive behaviors is versatile (*e.g.*, not only click, purchasing but also dwell time, revisit, skip, *etc*), which makes modeling non-trivial. To effective optimize such engagement, we have to first harvest previous information from such behaviors into Q-Network.

4.1.1 Raw Behavior Embedding Layer. The purpose of this layer is to take all raw behavior information, related to long term engagement, to distill users' state for further optimization. Given the observation $s_t = \{u, (i_1, f_1, d_1) \dots, (i_{t-1}, f_{t-1}, d_{t-1})\}$, we let f_t be all possible types of user bevahiors on i_t, including clicking, purchasing, or skipping, leaving *etc*, while d_t for the dwell time of the behavior. The entire set of $\{i_j\}$ are first converted into embedding vectors $\{\boldsymbol{i}_j\}$. To represent the feedback information into the item embedding, we project $\{\boldsymbol{i}_j\}$ into a feedback-dependent space by multiplying the embedding with a projection matrix as follow:

$$
\boldsymbol{i}'_j = F_{f_j} \boldsymbol{i}_j,
$$

where $F_{f_j} \in \mathbb{R}^{H \times H}$ is a projection matrix for a specific feedback f_j. To futher model time information, in our work, a time-LSTM[38] is
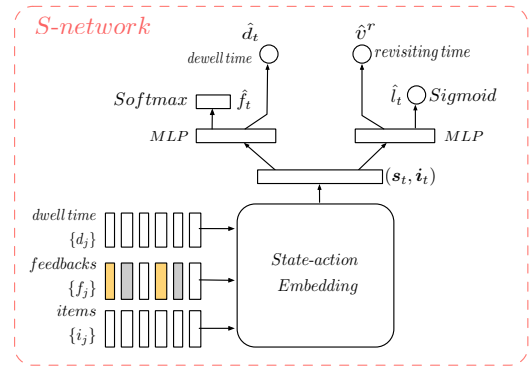

Figure 2: The architecture of S-Network.

used to track the user state over time as:

$$
\boldsymbol{h}_{r,j} = Time\text{-}LSTM(\boldsymbol{i}'_j, d_j), \quad (6)
$$

where Time-LSTM models the dwell time by inducing a time gate controlled by d_j as follow:

$$
\begin{aligned}
\boldsymbol{g}_j &= \sigma \left(\boldsymbol{i}'_j W_{ig} + \sigma \left(d_j W_{gg} \right) + b_g \right) \\
\boldsymbol{c}_j &= \boldsymbol{p}_j \odot \boldsymbol{c}_{j-1} + \boldsymbol{e}_j \odot \boldsymbol{g}_j \odot \sigma \left(\boldsymbol{i}'_j W_{ic} + \boldsymbol{h}_{m-1} W_{hc} + b_c \right) \\
\boldsymbol{o}_j &= \sigma(\boldsymbol{i}'_j W_{io} + d_j W_{do} + \boldsymbol{h}_{j-1} W_{ho} + \boldsymbol{w}_{co} \odot \boldsymbol{c}_j + b_o),
\end{aligned}
$$

where $\boldsymbol{c}_j$ is the memory cell. $\boldsymbol{g}_j$ is the time dependent gate influencing the memory cell and output gate. $\boldsymbol{p}_j$ is the forget gate. $\boldsymbol{e}_j$ is the input gate. $\boldsymbol{o}_j$ is the output gate. W_* and b_* are the weight and bias term. $\odot$ is the element-wise product, σ is the sigmoid function. Given the $\boldsymbol{c}_j$ and $\boldsymbol{o}_j$, the hidden state $\boldsymbol{h}_{r,j}$ is modeled as

$$
\boldsymbol{h}_{r,j} = o_j \odot \sigma(c_j).
$$

4.1.2 Hierarchical Behavior Layer. To capture the information of versatile user behaviors, all rough behaviors are sequentially fed into raw Behavior Embedding Layer indiscriminate. In realistic, the numbers for specific user actions is extremely imbalanced (*e.g.*, clicks are fewer than skips, and the purchase is even much sparser). As a result, directly utilizing the output of raw Behavior Embedding Layer will cause the Q-Network losing the information from the sparse user behaviors, *e.g.*, purchase information will be buried by skips information. Moreover, each type of user behaviors has its own characteristics: click on an item usually represents the users' current preferences, purchase on an item may imply the shifting of user interest, and causality of skipping is a little complex, which could be casual browsing, neutral, or annoyed, *etc*.

To better represent the user state, as shown in Figure 1, we propose a hierarchical behavior layer added to the raw behaviors embedding layers, that the major user behaviors, such as click, skip, purchase are tracked separately as

$$
\begin{aligned}
\boldsymbol{h}_{s,j} &= \text{LSTM-skip}(\boldsymbol{h}_{r,j}) \text{ if } f_j \text{ is skipping} \\
\boldsymbol{h}_{c,j} &= \text{LSTM-click}(\boldsymbol{h}_{r,j}) \text{ if } f_j \text{ is clicking} \\
\boldsymbol{h}_{o,j} &= \text{LSTM-order}(\boldsymbol{h}_{r,j}) \text{ if } f_j \text{ is ordering},
\end{aligned}
$$

where different user's behaviors is captured by the corresponding LSTM layer to avoid intensive behavior dominance and capture specific characteristics. Finally, the state-action embedding is formed

by concatenating different user's behavior layer and user profile as:

$$s_t = \text{concat}[h_{r,t}, h_{s,t}, h_{c,t}, h_{o,t}, u],$$

where u is the embedding vector for a specific user.

4.1.3 Q-value Layer. The approximation of Q-value is accomplished by MLP with the input of the dense state embedding and the item embedding as follow:

$$Q(s_t, i_t; \theta_q) = MLP(s_t, i_t).$$

The value of θ_q is updated by SGD with gradient calculated as Equation (5).

4.2 Off-Policy Learning Task

With the proposed Q-Learning based framework, we can train the parameters in the model through trial and error search before learning a stable recommendation policy. However, due to the cost and risk of deploying unsatisfactory policies, it is nearly impossible for training the policy online. An alternative way is to train a reasonable policy using the logged data $\mathcal{D}$, collecting by a logging policy π_b, before deploying. Unfortunately, the Q-Learning framework in Equation (4) suffers from the problem of *Deadly Trial*[24], the problem of instability and divergence arises whenever combining function approximation, bootstrapping and offline training.

To avoid the problem of instability and divergence in offline Q-Learning, we further introduce a user simulator (refers to as S-Network), which simulates the environment and assists the policy learning. Specifically, in each round of recommendation, aligning with real user feedback, the S-Network need to generate user's response f_t, the dwell time d_t, the revisited time v^r, and a binary variable l_t, which indicates whether the user leaves the platform. As shown in Figure 2, the generation of simulated user's feedback is accomplished using the S-Network $S(\theta_s)$, which is a multi-head neural network. State-action embedding is designed in the same architecture as it in Q-Network, but has separate parameters. The layer (s_t, i_t) are shared across all tasks, while the other layers (above (s_t, i_t) in Figure 2) are task-specific. As dwell time and user's feedback are inner-session behaviors, the prediction of $\hat{f}_t$ and $\hat{d}_t$ is calculated as follow,

$$
\begin{aligned}
\hat{f}_t &= \text{Softmax}(W_f x_f + b_f) \\
\hat{d}_t &= W_d x_f + b_d \\
x_f &= tanh(W_{xf}[s_t, i_t] + b_{xf})
\end{aligned}
$$

where W_* and b_* are the weight and bias term. $[s_t, i_t]$ is the concentration of state action feature. The generation of revisiting time and leaving the platform (inter-session behaviors) are accomplished as

$$
\begin{aligned}
\hat{l}_t &= \text{Sigmoid}(x_f^\top w_l + b_l) \\
\hat{v}^r &= W_v x_l + b_d \\
x_l &= tanh(W_{xl}[s_t, i_t] + b_{xl}).
\end{aligned}
$$

4.3 Simulator Learning

In this process, $S(s_t, i_t; \theta_s)$ is refined via mini-batch SGD using logged data in the $\mathcal{D}$. As the logged data is collected via a logging policy π_b, directly using such logged data to build the simulator will cause the selection base. To debias the effects of logging policy

---

Algorithm 1: Offline training of FeedRec.

Input: $\mathcal{D}, \epsilon, L, K$
Output: θ_q, θ_s

1   Randomly initialize parameters $\theta_q, \theta_s \leftarrow$ Uniform$(-0.1, 0.1)$;
2   #**Pretraining the S-Network.**
3   **for** $j = 1 : K$ **do**
4   Sample random mini-batches of (s_t, i_t, r_t, s_{t+1}) from $\mathcal{D}$;
5   Set f_t, d_t, v^r, l_t according to s_t, r_t, s_{t+1};
6   Update θ_s via mini-batch SGD w.r.t. the loss in Equation (7);
7   **end**
8   # Iterative training of S-Network and Q-Network.;
9   **repeat**
10   **for** $j = 1 : N$ **do**
11   # **Sampling training samples from logged data.**
12   Sampling (s, i, r, s') from $\mathcal{D}$, and storing in buffer $\mathcal{M}$;
13   # **Sampling training samples by interacting with the S-Network.**
14   l = False;
15   sample a initial user u from user set;
16   initial $s = \{u\}$;
17   **while** l *is False* **do**
18   sample a recommendation i *w.r.t* ϵ-greedy Q-value;
19   execute i;
20   S-Network responds with f, d, l, v^r;
21   set r according to f, d, l, v^r;
22   set $s' = s \oplus \{i, r, d\}$;
23   store (s, i, r, s') in buffer $\mathcal{M}$;
24   update $s \leftarrow s'$;
25   **end**
26   # **Updating the Q-Network.**
27   **for** $j = 1 : L$ **do**
28   Sample random mini-batches of training (s_t, i_t, r_t, s_{t+1}) from $\mathcal{M}$;
29   Update θ_q via mini-batch SGD w.r.t. Equation (5);
30   **end**
31   # **Updating the S-Network.**
32   **for** $j = 1 : K$ **do**
33   Sample mini-batches of (s_t, i_t, r_t, s_{t+1}) from $\mathcal{D}$;
34   Set f, d, l, v^r according to r_t, s_{t+1};
35   Update θ_s via mini-batch SGD *w.r.t.* the loss in Equation (7);
36   **end**
37   **end**
38   **until** *convergence*;

π_b [22], an importance weighted loss is minimized as follow:

$$
\begin{aligned}
\ell(\theta_s) &= \sum_{t=0}^{T-1} \gamma^t \frac{1}{n} \sum_{k=1}^{n} \{\omega_{0:t}, c\} \delta_t(\theta_s) \quad (7) \\
\delta_t(\theta_s) &= \lambda_f \cdot \Psi(f_t, \hat{f}_t) + \lambda_d \cdot (d_t - \hat{d}_t)^2 + \\
&\quad \lambda_l \cdot \Psi(l_t, \hat{l}_t) + \lambda_v \cdot (v^r - \hat{v}^r)^2,
\end{aligned}
$$

where n is the total number of trajectories in the logged data. $\omega_{0:t} = \prod_{k=0}^{t} \frac{\pi(i_k|s_k)}{\pi_b(i_k|s_k)}$ is the importance ratio to reduce the disparity between π (the policy derived from Q-Network, *e.g.,* ϵ-greedy) and π_b, $\Psi(\cdot, \cdot)$ is the cross-entropy loss function, and c is a hyperparameter to avoid too large importance ratio. $\delta_t(\theta_s)$ is a multi-task loss function that combines two classification loss and two regression loss, λ_* is the hyper-parameter controlling the importance of different task.

As π derived from Q-Network is constantly changed with the update of θ_q, to keep adaptive to the intermediate policies, the S-Network also keep updated in accordance with π to obtain the customized optimal accuracy. Finally, we implement an interactive

training procedure, as shown in Algorithm 1, where we specify the order in which they occur within each iteration.

5 SIMULATION STUDY

We demonstrate the ability of FeedRec to fit the user's interests by directly optimizing user engagement metrics through simulation. We use the synthetic datasets so that we know the "ground truth" mechanism to maximize user engagement, and we can easily check whether the algorithm can indeed learn the optimal policy to maximize delayed user engagement.

5.1 Setting

Formally, we generate M users and N items, each of which is associated with a d-dimensional topic parameter vector $\boldsymbol{\vartheta} \in \mathbb{R}^d$. For M users ($\mathcal{U} = \{\boldsymbol{\vartheta}_u^{(1)}, \ldots, \boldsymbol{\vartheta}_u^{(M)}\}$) and N items ($\mathcal{I} = \{\boldsymbol{\vartheta}_i^{(1)}, \ldots, \boldsymbol{\vartheta}_i^{(N)}\}$), the topic vectors are initialized as

$$\boldsymbol{\vartheta} = \frac{\tilde{\boldsymbol{\vartheta}}}{||\tilde{\boldsymbol{\vartheta}}||}, \text{ where } \tilde{\boldsymbol{\vartheta}} = \begin{cases} \tilde{\vartheta}_k = 1 - \kappa, & \text{the primary topic } k, \\ \tilde{\vartheta}_{k'} \sim U(0, \kappa), & k' \neq k, \end{cases} \quad (8)$$

where κ controls how much attention would be paid on non-primary topics. Specifically, we set the dimension of user vectors and item vectors to 10. Once the item vector $\boldsymbol{\vartheta}_i$ is initialized, it will keep the same for the simulation. At each time step t, the agent feeds one item $\boldsymbol{\vartheta}_i$ from $\mathcal{I}$ to one user $\boldsymbol{\vartheta}_u$. The user checks the feed and gives feedback, *e.g.,* click/skip, leave/stay (depth metric), and revisit (return time metric), based on the "satisfaction". Specifically, the probability of click is determined by the cosine similarity as $p(\text{click}|\boldsymbol{\vartheta}_u, \boldsymbol{\vartheta}_i) = \frac{\boldsymbol{\vartheta}_i^\top \boldsymbol{\vartheta}_u}{\|\boldsymbol{\vartheta}_i\|\|\boldsymbol{\vartheta}_u\|}$. For leave/stay or revisit, these feedback are related to all the feeds. In the simulation, we assume these feedback are determined by the mean entropy of recommendation list because many existing works[1, 2, 4] assume the diversity is able to improve the user's satisfactory on the recommendation results. Also, diversity is also delayed metrics [29, 39], which can verify whether FeedRec could optimize the delayed metrics or not.

5.2 Simulation Results

Some existing works [1, 2, 4] assume the diversity is able to improve the user's satisfactory on the recommendation results. Actually, it is an indirect method to optimize user engagement, and diversity here play an instrumental role to achieve this goal. We now verify that the proposed FeedRec framework has the ability to directly optimize user engagement through different forms of diversity. To generate the simulation data, we follow the popular diversity assumption[1, 2, 4]. These works tried to enhance diversity to achieve better user engagement. However, it is unclear that to what extent the diversity will lead to the best user engagement. Therefore, the pursuit of diversity may not lead to the improvement of user satisfaction.

We assume that there are two types of relationship between user engagement and diversity of recommendation list.

1) **Linear style**. In the linear relationship, higher entropy brings more satisfaction, that is, higher entropy attracts user to browse more items and use the system more often. The probability of user
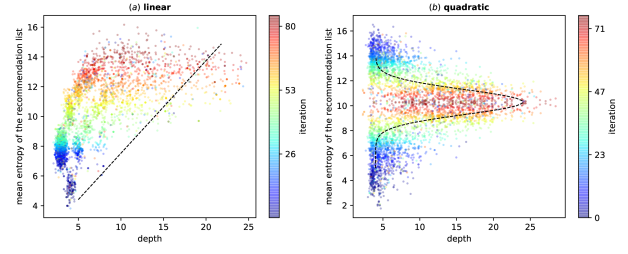

Figure 3: Different distributions of user's interests and browsing depth. The dashed line represents the distribution of scrolling down and entropy (linear in (a) and quadratic in (b)). The color bar shows the interaction iteration in training phrase, from blue to red. The average browsing depth over all users are shown as dots.
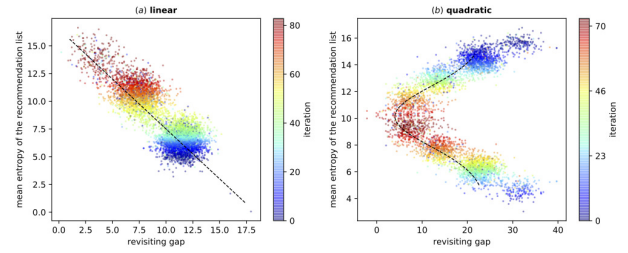

Figure 4: Different distributions of user's interests and interval days between two visits. The dashed line represents the distribution of return time and entropy (linear in (a) and quadratic in (b)). The color bar shows the interaction iteration in training phrase, from blue to red. The average return time over all users are shown as dots.

staying with the system after checking the fed items is set as:

$$p(stay|\boldsymbol{\vartheta}_1, \ldots, \boldsymbol{\vartheta}_t) = a\mathfrak{E}(\boldsymbol{\vartheta}_1, \ldots, \boldsymbol{\vartheta}_t) + b, \, a > 0$$

$$\mathfrak{E}(\boldsymbol{\vartheta}_1, \ldots, \boldsymbol{\vartheta}_t) = \frac{1}{t \times (t-1)} \sum_{\substack{m, n \in \{1, \ldots, t\} \\ m \neq n}} \boldsymbol{\vartheta}_m \log \frac{\boldsymbol{\vartheta}_m}{\boldsymbol{\vartheta}_n}$$

where $\{\boldsymbol{\vartheta}_1, \ldots, \boldsymbol{\vartheta}_t\}$ is the list of recommended items, $\mathfrak{E}(\boldsymbol{\vartheta}_1, \ldots, \boldsymbol{\vartheta}_t)$ is the mean entropy of the items. a and b are used to scale into range (0,1). The interval days of two visit is set as:

$$v^r = V - d * \mathfrak{E}(\boldsymbol{\vartheta}_{i,1}, \ldots, \boldsymbol{\vartheta}_{i,t}), \, V > 0, \, d > 0,$$

where V and d are constants to make v^r positive.

2) **Quadratic style**. In the quadratic relationship, moderate entropy makes the best satisfaction. The probability of user staying with the system after checking the fed items is set as:

$$p(stay|\boldsymbol{\vartheta}_1, \ldots, \boldsymbol{\vartheta}_t) = \exp\{-\frac{(\mathfrak{E}(\boldsymbol{\vartheta}_1, \ldots, \boldsymbol{\vartheta}_t) - \mu)^2}{\sigma}\},$$

where μ and σ are constants. Similarly, the interval days of two visit is set as:

$$v^r = V(1 - \exp\{-\frac{(\mathfrak{E}(\boldsymbol{\vartheta}_{i,1}, \ldots, \boldsymbol{\vartheta}_{i,t}) - \mu)^2}{\sigma}\}), \, V > 0.$$

Following the above process of interaction between "user" and system agent, we generate 1,000 users, 5,000 items, and 2M episodes for training.

We here report the average browsing depth and return time w.r.t. different relationship—linear or quadratic—of each training step, where the blue points are at earlier training steps and the red point is at the later training steps. From the results shown in

Table 1: Summary statistics of dataset.

Statistics	Numerical Value
The number of trajectories	633,345
The number of items	456,805
The number of users	471,822
Average/max/min clicks	2.04/99/0
Average/max/min dwell time(minutes)	2.4/5.3/0.5
Average/max/min browsing depth	13.34/149/1
Average/max/min return time (days)	5.18/17/0

Figure 3 and Figure 4, we can see that no matter what diversity assumptions are, FeedRec is able to converge to the best diversity by directly optimizing delay metrics. In (a) of Figure 3 (and also Figure 4), FeedRec discloses that the distribution of entropy of recommendation list and browsing depth (and also return time) is linear. As the number of rounds of interaction increases, the user's satisfaction gradually increases, therefore more items are browsed (and internal time between two visits is shorter). In (b), user engagement is highest in a certain entropy, higher or lower entropy can cause user's dissatisfaction. Therefore, moderate entropy of recommendation list will attract the user to browse more items and use the recommender system more often. The results indicate that FeedRec has the ability to fit different types of distribution between user engagement and the entropy of the recommendation list.

6 EXPERIMENTS ON REAL-WORLD E-COMMERCE DATASET

6.1 Dataset

We collected 17 days users' accessing logs in July 2018 from an e-commerce platform. Each accessing log contains: timestamp, user id u, user's profile ($u_p \in \mathbb{R}^{20}$), recommended item's id i_t, behavior policy's ranking score for the item $\pi_b(i_t|s_t)$, and user's feedback f_t, dwell time d_t. Due to the sparsity of the dataset, we initialized the items' embedding ($i \in \mathbb{R}^{20}$) with pretrained vectors, which is learned through modeling users' clicking streams with skip-gram [16]. The user's embedding u is initialized with user's profile u_p. The returning gap was computed as the interval days of the consecutive user visits. Table 1 shows the statistics of the dataset.

6.2 Evaluation Setting

off-line A/B testing. To perform evaluation of RL methods on ground-truth, a straightforward way is to evaluate the learned policy through online A/B test, which, however, could be prohibitively expensive and may hurt user experiences. Suggested by [6, 7], a specific off-policy estimator NCIS [25] is employed to evaluate the performance of different recommender agents. The step-wise variant of NCIS is defined as

$$\hat{R}^{\pi}_{step-NCIS} = \sum_{\xi_k \in \mathcal{T}} \sum_{t=0}^{T-1} \frac{\bar{\rho}^i_{0:t} r^k_t}{\sum_{j=1}^{K} \bar{\rho}^j_{0:t}} \quad (9)$$

$$\bar{\rho}_{t_1:t_2} = \min\{c, \prod_{t=t_1}^{t_2} \frac{\pi(a_t|s_t)}{\pi_b(a_t|s_t)}\},$$

where $\bar{\rho}_{t_1:t_2}$ is the max capping of importance ratio, $\mathcal{T} = \{\xi_k\}$ is the set of trajectory ξ_k for evaluation, K is the total testing trajectory. The numerator of Equation (9) is the capped importance weighted reward and the denominator is the normalized factor. Setting the r_t with different metrics, we can evaluate the policy from different perspective. To make the experimental results trustful and solid, we use the first 15 days logging as training samples, the last 2 days as testing data, the test data is kept isolated. The training samples are used for policy learning. The testing data are used for policy evaluation. To ensure small variance and control the bias, we set c as 5 in experiment.

The metrics. Setting the reward in Equation (9) with different user engagement metrics, we could estimate a comprehensive set of evaluation metrics. Formally, these metrics are defined as follows,

- Average Clicks per Session: the average cumulative number of clicks over a user visit.
- Average Depth per Session: the average browsing depth that the users interact with the recommender agent.
- Average Return Time: the average revisiting days between a user's consecutive visits up till a particular time point.

The baselines. We compare our model with state-of-the-art baselines, including both supervised learning based methods and reinforcement learning based methods.

- **FM**: Factorization Machines [21] is a strong factoring model, which can be easily implemented with factorization machine library (libFM)[4].
- **NCF**: Neural network-based Collaborative Filtering [9] replaces the inner product in factoring model with a neural architecture to support arbitrary function from data.
- **GRU4Rec**: This is a representative approach that utilizes RNN to learn the dynamic representation of customers and items in recommender systems [10].
- **NARM**: This is a state-of-the-art approach in personalized trajectory-based recommendation with RNN models [12]. It uses the attention mechanism to determine the relatedness of the past purchases in the trajectory for the next purchase.
- **DQN**: Deep Q-Networks [18] combined Q-learning with Deep Neural Networks. We use the same function approximation as FeedRec and train the neural network with naive Q-learning using the logged dataset.
- **DEERs**: DEERs [34] is a DQN based approach for maximizing users' clicks with pairwise training, which considers user's negative behaviors.
- **DDPG-KNN**: Deep Deterministic Policy Gradient with KNN [5] is a discrete version of DDPG for dealing with large action space, which has been deployed for Pagewise recommendation in [33].
- **FeedRec**: To verify the effect of different components, experiments are conducted on the degenerated models as follow: 1) **S-Network** is purely based on our proposed S-Network, which makes recommendations based on the ranking of next possible clicking item. 2) **FeedRec(C)**, **FeedRec(D)**, **FeedRec(R)** and **FeedRec(All)** are our proposed methods with different metrics as reward. Specifically, they use the clicks, depth, return time and

---

[4]http://www.libfm.org

Table 2: Performance comparison of different agents on JD dataset.

Agents	Average Clicks per Session	Average Depth per Session	Average Return Time
FM	1.9829	11.2977	16.5349
NCF	1.9425	11.1973	18.2746
GRU4Rec	2.1154	13.8060	14.0268
NARM	2.3030	15.3913	11.0332
DQN	1.8211	15.2508	6.2307
DEER	2.2773	18.0602	5.7363
DDPG-KNN(k=1)	0.6659	9.8127	15.4012
DDPG-KNN(k=0.1N)	2.5569	16.0936	7.3918
DDPG-KNN(k=N)	2.5090	14.6689	14.1648
S-Network	2.5124	16.1745	10.1846
FeedRec(C)	2.6194	18.1204	6.9640
FeedRec(D)	2.8217	21.8328	4.8756
FeedRec(R)	3.7194	23.4582	3.9280
FeedRec(All)	**4.0321***	**25.5652***	**3.9010***

" $*$ " indicates the statistically significant improvements (*i.e.,* two-sided
t-test with $p < 0.01$) over the best baseline.

the weighted sum of instant and delayed metrics as the reward function respectively.

Experimental Setting. The weight ω for different metrics is set to $[1.0, 0.005, 0.005]^\top$. The hidden units for LSTM is set as 50 for both Q-Network and S-Network. All the baseline models share the same layer and hidden nodes configuration for the neural networks. The buffer size for Q-Learning is set 10,000, the batch size is set to 256. ϵ-greedy is always applied for exploration in learning, but discounted with increasing training epoch. The value c for clipping importance sampling is set 5. We set the discount factor $\gamma = 0.9$. The networks are trained with SGD with learning rate of 0.005. We used Tensorflow to implement the pipelines and trained networks with a Nvidia GTX 1080 ti GPU cards. All the experiments are obtained by an average of 5 repeat runs.

6.3 Experimental Results

Comparison against baselines. We compared FeedRec with state-of-the-art methods. The results of all methods over the real-world dataset in terms of three metrics are shown in Table 2. From the results, we can see that FeedRec outperformed all of the baseline methods on all metrics. We conducted significance testing (t-test) on the improvements of our approaches over all baselines. " $*$ " denotes strong significant divergence with p-value<0.01. The results indicate that the improvements are significant, in terms of all of the evaluation measures.

The influence of weight ω. The weight ω controls the relative importance of different user engagement metrics in reward function. We examined the effects of the weights ω. (a) and (b) of Figure 5 shows the parameter sensitivity of ω *w.r.t.* depth metric and return time metric respectively. In Figure 5, *w.r.t.* the increase of the weight of ω for depth and return time metrics, the user browses more items and revisit the application more often (the blue line). Meanwhile, in both (a) and (b), the model achieves best results in the cumulative clicks metric (the orange line) when the ω is set to 0.005. Too much
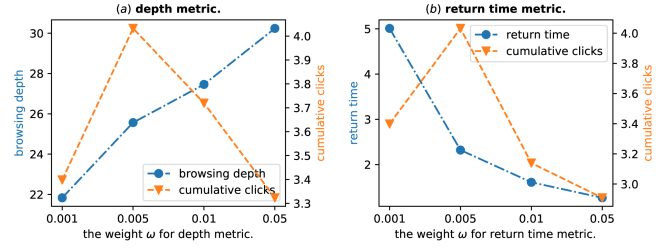
Figure 5: The influence of ω on performance.
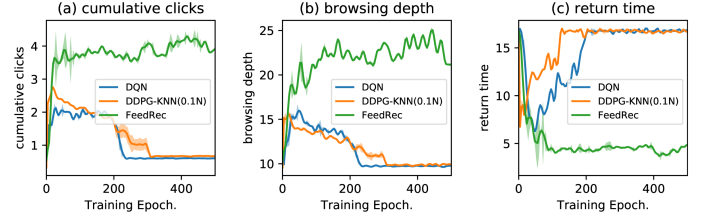
Figure 6: Comparison between FeedRec and baselines under offline learning.

weight on these metrics will overwhelm the importance of clicks on the rewards, which indicates that moderate value of weights on depth and return time can indeed improve the performance on cumulative clicks.

The effect of S-Network. The notorious *deadly triad* problem causes the danger of instability and divergence of most off-policy learning methods, even the robust Q-Learning. To examine the advantage of our proposed interactive training framework, we compared our proposed model FeedRec with DQN, DDPG-KNN under the same configuration. In Figure 6, we show different metrics vs the training iteration. We find that DQN, DDPG-KNN achieves a performance peak around 40 iterations and the performances are degraded rapidly with increasing iterations (the orange line and blue line). On the contrary, FeedRec achieves better performances on these three metrics and the performances are stable at the highest value (the green line). These observations indicate that FeedRec is stable and suitable through avoiding the *deadly triad* problem for off-policy learning of recommendation policies.

The relationship between user engagement and diversity. Some existing works [1, 2, 4] assume user engagement and diversity are related and intent to increase user engagement by increasing diversity. Actually, it is an indirect method to optimize the user engagement, and the assumption has not been verified. Here, we conducted experiments to see whether FeedRec, which direct optimize user engagement, has the ability to improve the recommendation diversity. For each policy, we sample 300 state-action pairs with importance ratio $\bar{\rho} > 0.01$ (the larger value of $\bar{\rho}$ in Equation (9) implies that the policy more favors such actions) and plot these state-action pairs, which are shown in Figure 7. The horizontal axis indicates the diversity between recommendation items, and the vertical axis indicates different types of user engagement (*e.g.,* browsing depth, return time). We can see that the FeedRec policy,
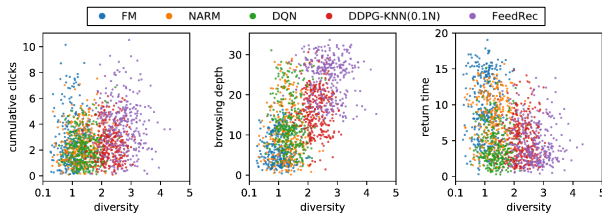
Figure 7: The relationship between user engagement and diversity.

learned by directly optimizing user engagement, favors for recommending more diverse items. The results verifies that optimization of user satisfaction can increase the recommendation diversity and enhancing diversity is also a means of improving user satisfaction.

7 CONCLUSION

It is critical to optimize long-term user engagement in the recommender system, especially in feed streaming setting. Though RL naturally fits the problem of maximizing the long-term rewards, there exist several challenges for applying RL in optimizing long-term user engagement: difficult to model the omnifarious user feedbacks (*e.g.,* clicks, dwell time, revisit, *etc*) and effective off-policy learning in recommender system. To address these issues, in this work, we introduce a RL-based framework — FeedRec to optimize the long-term user engagement. First, FeedRec leverage hierarchical RNNs to model complex user behaviors, refer to as Q-Network. Then to avoid the instability of convergence in policy learning, an S-Network is designed to simulate the environment and assist the Q-Network. Extensive experiments on both synthetic datasets and real-world e-commerce dataset have demonstrated effectiveness of FeedRec for feed streaming recommendation.

REFERENCES

[1] Gediminas Adomavicius and YoungOk Kwon. 2012. Improving aggregate recommendation diversity using ranking-based techniques. *TKDE* 24, 5 (2012), 896–911.

[2] Azin Ashkan, Branislav Kveton, Shlomo Berkovsky, and Zheng Wen. 2015. Optimal Greedy Diversity for Recommendation. In *IJCAI'15*. 1742–1748.

[3] Shiyu Chang, Yang Zhang, Jiliang Tang, Dawei Yin, Yi Chang, Mark A Hasegawa-Johnson, and Thomas S Huang. 2017. Streaming recommender systems. In *WWW'17*. ACM, 381–389.

[4] Peizhe Cheng, Shuaiqiang Wang, Jun Ma, Jiankai Sun, and Hui Xiong. 2017. Learning to Recommend Accurate and Diverse Items. In *WWW'17*. ACM, 183–192.

[5] 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).

[6] Mehrdad Farajtabar, Yinlam Chow, and Mohammad Ghavamzadeh. 2018. More Robust Doubly Robust Off-policy Evaluation. In *ICML'18*. 1446–1455.

[7] Alexandre Gilotte, Clément Calauzènes, Thomas Nedelec, Alexandre Abraham, and Simon Dollé. 2018. Offline A/B testing for Recommender Systems. In *WSDM'18*. ACM, 198–206.

[8] Li He, Long Xia, Wei Zeng, Zhiming Ma, Yihong Zhao, and Dawei Yin. 2019. Off-policy Learning for Multiple Loggers. In *SIGKDD'19*. ACM.

[9] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In *WWW'17*. ACM, 173–182.

[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] Mounia Lalmas, Heather O'Brien, and Elad Yom-Tov. 2014. Measuring user engagement. *Synthesis Lectures on Information Concepts, Retrieval, and Services* 6, 4 (2014), 1–132.

[12] Jing Li, Pengjie Ren, Zhumin Chen, Zhaochun Ren, Tao Lian, and Jun Ma. 2017. Neural Attentive Session-based Recommendation. In *CIKM'17*. ACM, 1419–1428.

[13] Lihong Li, Wei Chu, John Langford, and Robert E Schapire. 2010. A contextual-bandit approach to personalized news article recommendation. In *WWW'10*. ACM, 661–670.

[14] Zhongqi Lu and Qiang Yang. 2016. Partially Observable Markov Decision Process for Recommender Systems. *arXiv preprint arXiv:1608.07793* (2016).

[15] Tariq Mahmood and Francesco Ricci. 2009. Improving recommender systems with adaptive conversational strategies. In *HT'09*. ACM, 73–82.

[16] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. *arXiv preprint arXiv:1301.3781* (2013).

[17] Andriy Mnih and Ruslan R Salakhutdinov. 2008. Probabilistic matrix factorization. In *NIPS'08*. 1257–1264.

[18] 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).

[19] Bruno Pradel, Savaneary Sean, Julien Delporte, Sébastien Guérif, Céline Rouveirol, Nicolas Usunier, Françoise Fogelman-Soulié, and Frédéric Dufau-Joel. 2011. A case study in a recommender system based on purchase data. In *SIGKDD'11*. ACM, 377–385.

[20] Lijing Qin, Shouyuan Chen, and Xiaoyan Zhu. 2014. Contextual combinatorial bandit and its application on diversified online recommendation. In *SDM'14*. SIAM, 461–469.

[21] Steffen Rendle. 2010. Factorization machines. In *ICDM'10*. IEEE, 995–1000.

[22] Tobias Schnabel, Adith Swaminathan, Ashudeep Singh, Navin Chandak, and Thorsten Joachims. 2016. Recommendations as Treatments: Debiasing Learning and Evaluation. In *ICML'16*. 1670–1679.

[23] Guy Shani, David Heckerman, and Ronen I Brafman. 2005. An MDP-based recommender system. *JMLR* 6, Sep (2005), 1265–1295.

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

[25] Adith Swaminathan and Thorsten Joachims. 2015. The self-normalized estimator for counterfactual learning. In *NIPS'15*. 3231–3239.

[26] Huazheng Wang, Qingyun Wu, and Hongning Wang. 2017. Factorization Bandits for Interactive Recommendation.. In *AAAI'17*. 2695–2702.

[27] Zihan Wang, Ziheng Jiang, Zhaochun Ren, Jiliang Tang, and Dawei Yin. 2018. A path-constrained framework for discriminating substitutable and complementary products in e-commerce. In *WSDM'18*. ACM, 619–627.

[28] Qingyun Wu, Hongning Wang, Liangjie Hong, and Yue Shi. 2017. Returning is Believing: Optimizing Long-term User Engagement in Recommender Systems. In *WWW'17*. ACM, 1927–1936.

[29] Long Xia, Jun Xu, Yanyan Lan, Jiafeng Guo, Wei Zeng, and Xueqi Cheng. 2017. Adapting Markov decision process for search result diversification. In *SIGIR'17*. ACM, 535–544.

[30] Xing Yi, Liangjie Hong, Erheng Zhong, Nanthan Nan Liu, and Suju Rajan. 2014. Beyond clicks: dwell time for personalization. In *RecSys'14*. ACM, 113–120.

[31] Chunqiu Zeng, Qing Wang, Shekoofeh Mokhtari, and Tao Li. 2016. Online context-aware recommendation with time varying multi-armed bandit. In *SIGKDD'16*. ACM, 2025–2034.

[32] Xiangyu Zhao, Xia Long, Tang Jiliang, and Yin Dawei. 2018. Deep Reinforcement Learning for Search, Recommendation, and Online Advertising: A Survey. *arXiv preprint arXiv:1812.07127* (2018).

[33] Xiangyu Zhao, Long Xia, Liang Zhang, Zhuoye Ding, Dawei Yin, and Jiliang Tang. 2018. Deep reinforcement learning for page-wise recommendations. In *RecSys'18*. ACM, 95–103.

[34] Xiangyu Zhao, Liang Zhang, Zhuoye Ding, Long Xia, Jiliang Tang, and Dawei Yin. 2018. Recommendations with negative feedback via pairwise deep reinforcement learning. In *SIGKDD'18*. ACM, 1040–1048.

[35] Xiangyu Zhao, Liang Zhang, Zhuoye Ding, Dawei Yin, Yihong Zhao, and Jiliang Tang. 2017. Deep reinforcement learning for list-wise recommendations. *arXiv preprint arXiv:1801.00209* (2017).

[36] Yin Zheng, Bangsheng Tang, Wenkui Ding, and Hanning Zhou. 2016. A neural autoregressive approach to collaborative filtering. In *ICML'16*. 764–773.

[37] Meizi Zhou, Zhuoye Ding, Jiliang Tang, and Dawei Yin. 2018. Micro behaviors: A new perspective in e-commerce recommender systems. In *WSDM'18*. ACM, 727–735.

[38] Yu Zhu, Hao Li, Yikang Liao, Beidou Wang, Ziyu Guan, Haifeng Liu, and Deng Cai. 2017. What to do next: Modeling user behaviors by time-lstm. In *IJCAI'17*. 3602–3608.

[39] Lixin Zou, Long Xia, Zhuoye Ding, Dawei Yin, Jiaxing Song, and Weidong Liu. 2019. Reinforcement Learning to Diversify Top-N Recommendation. In *DASFAA'19*. Springer, 104–120.