Pseudo Dyna-Q: A Reinforcement Learning Framework for Interactive Recommendation

Lixin Zou[1], Long Xia[2], Pan Du[3], Zhuo Zhang[4],
Ting Bai[5], Weidong Liu[1], Jian-Yun Nie[3], Dawei Yin[6,*]
[1]Tsinghua University, China, [2]York University, Canada
[3] University of Montreal, Canada, [4] The University of Melbourne, Australia
[5] Beijing University of Posts and Telecommunications, China, [6]JD Data Science Lab, China
{zoulx15,liuwd}@mails.tsinghua.edu.cn,longxia@yorku.ca,zhuo.zhang.cn@gmail.com
baiting@bupt.edu.cn,{nie,pandu}@iro.umontreal.ca,yindawei@acm.org

ABSTRACT

Applying reinforcement learning (RL) in recommender systems is attractive but costly due to the constraint of the interaction with real customers, where performing online policy learning through interacting with real customers usually harms customer experiences. A practical alternative is to build a recommender agent offline from logged data, whereas directly using logged data offline leads to the problem of selection bias between logging policy and the recommendation policy. The existing direct offline learning algorithms, such as Monte Carlo methods and temporal difference methods are either computationally expensive or unstable on convergence.

To address these issues, we propose **Pseudo Dyna-Q (PDQ)**. In PDQ, instead of interacting with real customers, we resort to a customer simulator, referred to as the World Model, which is designed to simulate the environment and handle the selection bias of logged data. During policy improvement, the World Model is constantly updated and optimized adaptively, according to the current recommendation policy. This way, the proposed PDQ not only avoids the instability of convergence and high computation cost of existing approaches but also provides unlimited interactions without involving real customers. Moreover, a proved upper bound of empirical error of reward function guarantees that the learned offline policy has lower bias and variance. Extensive experiments demonstrated the advantages of PDQ on two real-world datasets against state-of-the-arts methods.

CCS CONCEPTS

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

KEYWORDS

Pseudo Dyna-Q, Customer Simulator, Model-Based Reinforcement Learning, Offline Policy Learning, Recommender Systems

* Corresponding author.

WSDM '20, February 3–7, 2020, Houston, TX, USA

ACM ISBN 978-1-4503-6822-3/20/02…$15.00
https://doi.org/10.1145/3336191.3371801

ACM Reference Format:
Lixin Zou[1], Long Xia[2], Pan Du[3], Zhuo Zhang[4], Ting Bai[5], Weidong Liu[1], Jian-Yun Nie[3], Dawei Yin[6,*]. 2020. Pseudo Dyna-Q: A Reinforcement Learning Framework for Interactive Recommendation. In *The Thirteenth ACM International Conference on Web Search and Data Mining (WSDM '20), February 3–7, 2020, Houston, TX, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3336191.3371801

1 INTRODUCTION

Recommender systems have shown its effectiveness and become more popular for past decades. Recommendation by nature is an interactive process: a recommendation agent suggests items, based on customers' preferences; customers provide feedback on the suggested items; and the agent updates customers' preferences and makes further recommendations. Applying reinforcement learning (RL) to interactive recommendation (e.g., personalized music streams in Spotify[1], product feeds in Amazon[2], image feeds in Pinterests[3]) has attracted a lot of interests from the research community [30, 47, 49]. However, employing RL in real-world recommender systems still remains challenging. In general, the RL agents are learned through trial-and-error search, such as Atari games [22] and AlphaGo [31, 32], where the agents improve policy via numerous failures before achieving greatest strides. In realistic recommender systems, directly building a recommender agent from scratch, which requires the agent to interact with real customers numerous times, will hurt customer experiences and no customer would be willing to collaborate for a long time. An alternative is to make use of the logged data and build a recommender agent in offline manner before deploying online.

The logged data in offline policy learning are generally used in two different ways: it can be used in Direct Reinforcement Learning, which trains the recommendation policy directly using the logged data (refers to as *learning*); or it can be used in Indirect Reinforcement Learning, which first builds a simulator to imitate customers' behaviors and then learns the policy via querying the simulator (refers to as *planning*). Current methods of direct reinforcement learning include Monte Carlo (MC) and Temporal Difference (TD). Offline MC estimation with importance sampling guarantees an unbiased estimation, but it suffers from the problem of high variance. Especially in realistic recommender systems, there are often millions of candidate items for the recommendation, leading to an

[1]https://www.spotify.com/
[2]https://www.amazon.com/
[3]https://www.pinterest.com/

extremely large action space and an unbounded importance weight of training samples. As a result, it requires both large training samples and computation resources to achieve statistical efficiency in training. TD-based methods improve efficiency by using the bootstrapping technique in estimation. However, it is confronted with another notorious problem called *Deadly Triad*, that is, the problem of instability and divergence arises whenever combining function approximation, bootstrapping and offline training [35] (see the example of training divergence in Sec 5.3.1 Figure 4). Unfortunately, in recommender systems, due to the complexity of modeling customer behaviors, most state-of-the-art methods [47, 48] that are designed with neural architectures, will encounter inevitably the *Deadly Triad* problem in offline policy learning.

The indirect reinforcement learning approach using a simulator, theoretically, does not incur the real-world cost and can provide unlimited simulated experiences to learn the recommendation policy. However, building an effective recommendation simulator is in its own non-trivial problem, which has not been well explored up to now. Preliminary works on recommendation simulators [28, 47] typically ignore the selection bias of logged data [29], resulting in a biased simulator. Moreover, in those existing methods, a simulator was built before performing policy learning and is kept unchanged during policy learning, that is, a fixed simulator serves all intermediate policies when performing policy improvement. We believe that the simulator should also be updated constantly, in accordance with the improved target policies, to obtain the customized optimal accuracy of the simulation.

To address these issues, inspired by Dyna-Q [23, 34], we integrated learning (direct reinforcement learning) and planning (indirect reinforcement learning) in a unified framework, named ***P**seudo **D**yna-**Q*** (PDQ). Different from Dyna-Q, the recommendation policy is trained without the requirement of real customer interactions. Specifically, we introduce an environment model, referred to as *world model*, to simulate the environments and generate simulated customer experiences in offline policy learning. The policy learning is then decomposed into two iterative steps: in the first step, the world model is constantly updated in accordance to current recommendation policy by de-biasing the selection bias with importance sampling; the second step improves the recommendation policy with Q-learning, via both the logged data and the world model (referred to as direct reinforcement learning and planning respectively). Compared with existing approaches[28, 47], the advantages of PDQ lie in two aspects: **1)** PDQ breaks the Deadly Triad by employing a world model for planning; **2)** the bias induced by simulator is minimized by constantly updating the world model and by a direct off-policy learning.

To these ends, our main contributions in this work are as follows:

- We present Pseudo Dyna-Q (PDQ) for interactive recommendation, which provides a general framework that can be instantiated in different neural architectures, and tailored for specific recommendation tasks.
- We conduct a general error analysis for the world model and show the connection of the error and dispersity between recommendation policy and logging policy.
- We implement a simple instantiation of PDQ, and demonstrate its effectiveness on two real-world large scale datasets, showing superior performance over the state-of-the-art methods in interactive recommendation.

2 PRELIMINARIES

Online interactive recommendation. In general, we assume a typical interactive recommendation setting between the customer and the recommender system – in each interaction, the customer is recommended an item $i_t \in \mathcal{I}$ and provides a feedback $f_t \in \mathcal{F}$ (i.e.,skipping, clicking or purchase) at the t-th interaction; then the system recommends new items i_{t+1} for the customer until the customer leaves the platform. Here, $\mathcal{I}$ is the set of candidate items for recommendation, $\mathcal{F}$ is the set of possible feedback. Such an interactive process can be formally formulated as a Markov Decision Process (MDP).

Given the observation on past interactions $s_t = \{u, i_1, f_1, \ldots, i_{t-1}, f_{t-1}\}$ (i.e., the state in MDP with $s_t \in \mathcal{S}$), the recommender is modeled by a conditional distribution $\pi : \mathcal{S} \times \mathcal{I} \rightarrow \mathbb{R}$ with $\pi(i|s_t)$ (i.e., the policy in RL community) being probability of recommending item i at the t-th interaction. The interaction between the customer and the recommender will generate a recommendation trajectory as $\xi = (s_0, i_0, r_0, \ldots, s_t, i_t, r_t, ..., s_T)$, where $r_t \in \mathbb{R}$ is a reward associated with customer's feedback, e.g., a click or a purchase. The aim of a recommender agent is to learn a policy π for maximizing the reward values of a trajectory ξ, formulated as:

$$\pi^* = \arg\max_{\pi \in \Pi} \eta(\pi). \tag{1}$$

Here, $\eta(\pi)$ is the expected discounted reward by following the policy π:

$$\eta(\pi) = \mathbb{E}_{\xi \sim P^{\pi}_{\xi}} \left[\sum_{t=0}^{T} \gamma^t r(s_t, i_t) \right], \tag{2}$$

where $r(s_t, i_t) \in [0, r_{\max}]$ is the reward associated with customer's feedback; $\xi \sim P^{\pi}_{\xi}$ means that the trajectory is generated by following the recommendation policy π; γ is hyper-parameter for discounting the long-term rewards.

Offline learning task. As mentioned earlier, due to the high cost and risk of deploying an immature recommendation policy, the offline learning task aims to learn a recommendation policy using a large logged trajectories to avoid interactions with real customers online.

Given the logged trajectory data $\mathcal{D} = \{\xi^{(k)}\}_{k=1}^{N}$, where $\xi^{(k)} \sim P^{\pi_b}_{\xi}$, N is the total number of trajectories in the logged data, the aim of the offline learning task is the same as that of the online recommender, as expressed in Equation 1. The difference is that the trajectories are supposed to be generated independently by a logging policy π_b, i.e. $\xi^{(k)} \sim P^{\pi_b}_{\xi}$, instead of being generated through interactions with real customers, i.e., $\xi^{(k)} \sim P^{\pi}_{\xi}$ as in Equation 2. The offline learning task needs to handle the policy bias to learn an optimal policy π^* without interactions with real customers online.

3 POLICY LEARNING FOR RECOMMENDER VIA PSEUDO DYNA-Q

The proposed PDQ recommender agent is shown in Figure 1. It consists of two modules:

- *A world model* for generating simulated customers' feedback, which should be similar to those generated by a real customer according to the historical logged data.
- *A recommendation policy* which selects the next item to recommend based on the current state. It is learned to maximize the cumulative reward, such as total clicks in a session.

The recommender policy and the world model are co-trained in an iterative way in PDQ. In each iteration, once the current recommender policy is set, the world model will be updated accordingly to support it. In turn, the new information gained from the updated world model will further improve the recommendation policy through planning. This way, the recommendation policy is iteratively improved with an evolving world model.

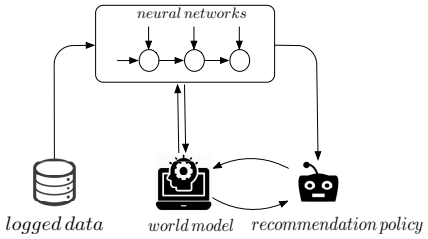

Figure 1: An illustration of Pseudo Dyna-Q Framework.

3.1 World Model Learning

3.1.1 The Error Function. The goal of the world model is to imitate the customer's feedback and generate the pseudo experiences as real as possible. As the reward function is associated with a customer's feedback, e.g., a click or a purchase, learning the reward function is equivalent to imitate customers' feedback. Formally, the world model can be learned effectively by minimizing the errors between online and offline rewards:

$$\begin{aligned} \ell(\pi;\theta_M) &= Err\left(\eta(\pi),\eta(\pi;\theta_M)\right) \\ &\triangleq \mathbb{E}_{\xi\sim P_\xi^\pi}\left[\sum_{t=0}^{T-1}\gamma^t\Delta(r_t,r_t(\theta_M))\right], \end{aligned} \tag{3}$$

where $\eta(\pi)$ is the expected discounted reward following policy in real world, while $\eta(\pi;\theta_M)$ is the expected reward of following π with θ_M as the parameter in the world model. $\Delta(\cdot) = \frac{\delta(\cdot)}{K} - 1 \in [-1,0]$ measures the difference between the real reward $r(\cdot)$ and learned reward $r(\cdot;\theta_M)$ with $\delta(\cdot;\theta_M) \in [0,K]$ as a scalar loss function.

Since ξ in the dataset $\mathcal{D}$ is generated by logging policy π_b, the objective function can be rewritten as:

$$\begin{aligned} \theta_M^* &= \arg\min_{\theta_M\in\Theta}\mathbb{E}_{\xi\sim P_\xi^\pi}\left[\sum_{t=0}^{T-1}\gamma^t\Delta_t(\theta_M)\right] \\ &= \arg\min_{\theta_M\in\Theta}\mathbb{E}_{\xi\sim P_\xi^{\pi_b}}\left[\sum_{t=0}^{T-1}\gamma^t\omega_{0:t}\Delta_t(\theta_M)\right]. \end{aligned} \tag{4}$$

Here, $\Delta_t(\theta_M)$ is the shorthand for $\Delta(r_t,r_t(\theta_M))$, ξ is generated by following logging policy π_b, $\omega_{0:t} \triangleq \prod_{j=0}^{t}\frac{\pi(i_j|s_j)}{\pi_b(i_j|s_j)}$ is the importance ratio to correct the discrepancy between recommendation policy and logging policy. Accordingly, $\hat{\theta}_M^*$ can be acquired by solving the sample average approximation:

$$\hat{\ell}(\pi;\theta_M) = \sum_{t=0}^{T-1}\gamma^t\cdot\frac{1}{N}\sum_{k=1}^{N}\omega_{0:t}^{(k)}\Delta_t(\theta_M). \tag{5}$$

However, this estimator has unbounded variance, since $\omega_{0:t}^{(k)}$ can be arbitrarily big when $\pi_b \approx 0$, which causes $\hat{\ell}(\pi;\theta_M)$ to be far away from the true risk $\ell(\pi;\theta_M)$. This problem can be fixed by "clipping" the importance sampling weights [16] as:

$$\hat{\ell}^c(\pi;\theta_M) = \sum_{t=0}^{T-1}\gamma^t\cdot\frac{1}{N}\sum_{k=1}^{N}\min\left\{\omega_{0:t}^{(k)},c\right\}\Delta_t(\theta_M), \tag{6}$$

where $c>0$ is a hyper-parameter chosen to balance the bias and variance in the estimator, i.e. a smaller value of c means toleration of a larger bias in the estimator.

3.1.2 The Error Bound and Its Induced Regularizer. The variance of $\hat{\ell}^c(\pi;\theta_M)$ in Equation (6) varies very differently across different hypothesis. Consider two policies π_1 and π_2, where π_1 is similar to π_b, but π_2 is not. Importance sampling gives us lower variance estimates for $\hat{\ell}^M(\pi_1;\theta_M)$, but higher variance estimates for $\hat{\ell}^M(\pi_2;\theta_M)$. Following the intuition above, we can get the upper bound of the error function in Equation (7) (the proof is provided in Appendix).

Theorem 3.1. *Let $\rho_\pi^t(s,i)$ be the probability of arriving at state s and taking the action i at timestep t when following policy π. Define the divergence between the policy π and π_b as $D_f(\pi\|\pi_b) = \sum_t\gamma^t d_f\left(\rho_\pi^t\|\rho_{\pi_b}^t\right) = \sum_t\gamma^t\left[\sum_{(s_t,i_t)} f\left(\frac{\rho_\pi^t}{\rho_{\pi_b}^t}\right)\rho_{\pi_b}^t\right]$, where $d_f(\cdot\|\cdot)$ is the f-divergence with $f(x)=x^2-x$. With probability at least $1-\zeta$, for all θ_M, we have $\ell(\pi;\theta_M)\le\mathcal{B}$, where*

$$\mathcal{B} = \hat{\ell}^c(\pi;\theta_M) + \sqrt{\frac{18((1-\gamma)D_f(\pi\|\pi_b)+1)\mathcal{Q}_{\theta_M}(n,\zeta)}{nT}} + \frac{c\cdot 45\mathcal{Q}_{\theta_M}(n,\zeta)}{n-1}, \tag{7}$$

and $\mathcal{Q}_{\theta_M}(n,\zeta)$ measures the capacity of the reward function family.

The second part of the error bound in Equation (7) indicates that, in order to tighten the upper error bound, we can force ρ_π^t and $\rho_{\pi_b}^t$ to be as close as possible. We are here unable to intervene ρ_π^t directly (which is typically obtained through maximizing $\eta(\pi;\theta_M)$ in policy learning phase, see in Sec 3.2). Alternatively, we can intervene the reward function $r(\theta_M)$ to force ρ_π^t to approach $\rho_{\pi_b}^t$, as long as we can find out the relationship among $r(\theta_M)$, ρ_π^t and $\rho_{\pi_b}^t$. The relationship is stated in Lemma 3.2 (the proof is provided in Appendix).

Lemma 3.2. *Assuming that r' is the reward function which satisfies $r'(s_t,i_t)\propto\rho_{\pi_b}^t(s_t,i_t)$, π^* is the optimal policy maximizing $\eta(\pi)$ under r'. For any indicator policy π, we have $D_f(\pi\|\pi_b)\ge D_f(\pi^*\|\pi_b)$.*

As stated in Lemma 3.2, the optimal policy π^* for $r(s_t,i_t;\theta_M)$ has minimal divergences $D_f(\pi^*\|\pi_b)$ when the reward $r(s_t,i_t;\theta_M)$ is proportional to $\rho_{\pi_b}^t(s_t,i_t)$. In other words, the state action pair (s_t,i_t) with high visiting frequency $\rho_{\pi_b}^t$ should be assigned higher reward by $r(s_t,i_t;\theta_M)$. This inspires us to add a regularizer intervening the reward function r to force ρ_π^t to approach $\rho_{\pi_b}^t$.

The offline (s_t, i_t) are generated by following π_b, which follows the probability of $\rho^t_{\pi_b}(s_t, i_t)$. The regularized error function is:

$$\ell^{cr}(\pi;\theta_M) = \mathbb{E}_{\xi\sim P^{\pi_b}_{\xi}}\left[\sum_{t=0}^{T-1}\min\{\omega_{0:t}, c\}\,\gamma^t\Delta_t(\theta_M)\right] + \lambda\sum_{(s_t,i_t)}\rho^t_{\pi_b}\Delta(r_{\max}, r_t(\theta_M)), \quad (8)$$

where $\Delta(r_{\max}, r_t(\theta_M))$ is the regularizer for encouraging π to visit (s_t, i_t) (heuristically reducing the divergence $D_f(\pi\|\pi_b)$ in Theorem 3.1). λ is the hyper-parameter that controls the influence of the regularizer term. Accordingly, $\hat{\theta}^*_M$ can be obtained by solving the sample average approximation

$$\hat{\ell}^{cr}(\pi;\theta_M) = \sum_{t=0}^{T-1}\gamma^t\cdot\frac{1}{N}\sum_{k=1}^{N}\left(\min\left\{\omega^{(k)}_{0:t}, c\right\}\Delta_t(\theta_M) + \lambda\Delta_t(r_{\max}, r(\theta_M))\right). \quad (9)$$

3.2 Policy Learning

We use Q-Learning [22] to improve the recommendation policy via using the experiences from the world model and via directly using the logged experiences. In each time-step t of recommendation, the recommender agent observes the state of customer s_t, and chooses the item i_t to recommend using an ϵ-greedy policy (i.e.,with probability $1-\epsilon$ selecting the max Q-value action, with probability ϵ randomly choosing an action) w.r.t. the approximated value function $Q(s, i; \theta_Q)$, which can be customized for specific recommendation tasks. The agent then receives the response $r(s_t, i_t; \theta_M)$ from world model and updates the state to s_{t+1}. Finally, we store the experience (s_t, i_t, r_t, s_{t+1}) in a large replay buffer $\mathcal{M}$ from which samples are taken in mini-batch training. The cycle continues until the customer leaves the platform.

We improve the value function $Q(s, i; \theta_Q)$ by adjusting θ_Q to minimize the mean-square loss function, defined as follows:

$$\ell(\theta_Q) = \mathbb{E}_{(s_t,i_t,r_t,s_{t+1})\sim\mathcal{M}}\left[(y_t - Q(s_t, i_t; \theta_Q))^2\right] \quad (10)$$
$$y_t = r_t + \gamma\max_{i_{t+1}\in I} Q(s_{t+1}, i_{t+1}; \theta_Q),$$

where y_t is the target value based on the optimal *Bellman Equation* [35]. By differentiating the loss function w.r.t. θ_Q, we arrive at the following gradient:

$$\nabla_{\theta_Q}\ell(\theta_Q) = \mathbb{E}_{(s_t,i_t,r_t,s_{t+1})\sim\mathcal{M}}\left[(r + \gamma\max_{i_{t+1}} Q(s_{t+1}, i_{t+1}; \theta_Q) - Q(s_t, i_t; \theta_Q))\nabla_{\theta_Q}Q(s_t, i_t; \theta_Q)\right] \quad (11)$$

In fact, the policy learning maximizes the $\eta(\pi;\theta_M)$ of the world model, where the regularization in Equation (9) is actually conveyed in θ_Q.

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

Algorithm 1: The training of Pseudo Dyna-Q.

Input: $\mathcal{D}$, ϵ,L,K
Output: $M(s, i; \theta_M)$, $Q(s, i; \theta_Q)$
Randomly initialize parameters θ_Q, $\theta_M \leftarrow$ Uniform(−0.1, 0.1);
# **Pretraining the world model.**
for $j = 1 : K$ **do**
  Sample random mini-batches of (s_t, i_t, r_t, s_{t+1}) from $\mathcal{D}$;
  Set f_t according to r_t;
  Set e_t according to s_{t+1};
  Update θ_M via mini-batch SGD w.r.t. the loss in Equation (9);
end
# **Iterative training of world model and Q-value network.**;
repeat
  for $j = 1 : N$ **do**
    # **Sampling training data by querying the world model.**
    e = False;
    sample a initial customer u from customer set;
    initialize $s = \{u\}$;
    while e *is False* **do**
      sample a recommendation i by ϵ-greedy w.r.t Q-value;
      execute i;
      world model responds with f, e;
      set r according to f;
      set $s' = s \oplus \{i, r\}$;
      store (s, i, r, s') in buffer $\mathcal{M}$;
      update $s \leftarrow s'$;
    end
    # **Adding logged data to the training samples.**
    Sampling (s, i, r, s') from $\mathcal{D}$, and storing in buffer $\mathcal{M}$;
    # **Updating the Q-value network.**
    for $j = 1 : L$ **do**
      Sample random mini-batches of training (s_t, i_t, r_t, s_{t+1}) from $\mathcal{M}$;
      Update θ_Q via mini-batch SGD w.r.t. Equation (11);
    end
    # **Updating the world model.**
    for $j = 1 : K$ **do**
      Sample mini-batches of (s_t, i_t, r_t, s_{t+1}) from $\mathcal{D}$;
      Set f_t, e_t according to r_t, s_{t+1};
      Update θ_M via mini-batch SGD w.r.t. the loss in Equation (9);
    end
  end
until *convergence*;

4 AN INSTANTIATION OF PSEUDO DYNA-Q

We have described a general framework of PDQ. In this section, we present a simple instantiation of the framework. Note that the proposed framework is not limited to the instantiation we describe here. More sophisticated designs of state representation, world model and Q-value network could be used, according to the specific recommendation tasks. As shown in Figure 2, the instantiation of Pseudo Dyna-Q contains three parts: (a)*The state tracker* for tracking current customer's preferences, e.g. encoding both the long-term and temporary interests into a dense state representation s_t; (b) *The Q-value Network* for predicting the Q-value of the policy; (c) *The world model* for generating pseudo customer's feedback.

4.1 State Tracker

RNN is often used to keep track of the states. In reality, customers' current interests are often related to the earlier items in addition to the recent ones, and RNN-based methods are unable to cope with such long term dependency. Recent works on memory network and self attention show effectiveness on this issue [20, 33, 41], and we here adopt such a design. Given the observation $s_t = \{u, i_1, f_1, \ldots, i_{t-1}, f_{t-1}\}$, the entire set of $\{i_j\}$ are converted into embedding vectors $\{\boldsymbol{i}_j\}$ of dimension H by embedding each i_j in a continuous space, which, in the simplest case, is an embedding matrix A (of size $I \times H$). To represent the feedback information

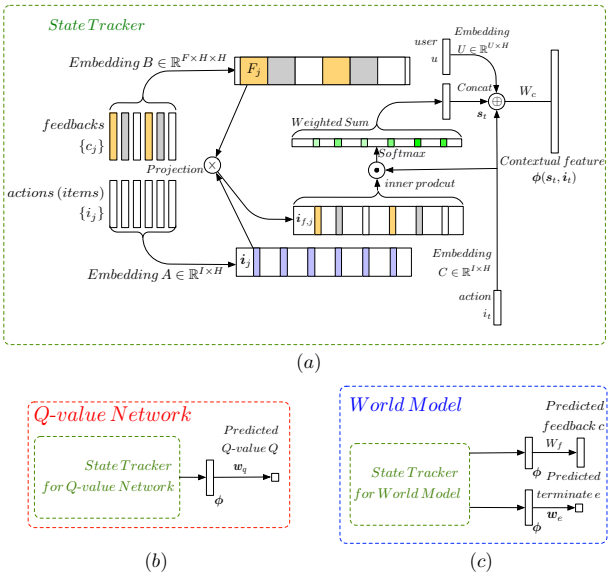

Figure 2: The neural architecture of PDQ. (a)*The state tracker* maintains customer's preferences with a memory network and self-attention mechanism; (b)*The Q-value Network* predicts the Q-value by the inner product between tracked customer's preferences and a weight vector; (c)*The world model* generates customer's feedback with a multi-head MLP.

into item embedding, we project $\{\boldsymbol{i}_j\}$ into a feedback-dependent space by multiplying the embedding with a feedback-dependent projection matrix as follows:

$$\boldsymbol{i}_{f_j,j} = F_{f_j} \boldsymbol{i}_j, \tag{12}$$

where $F_{f_j} \in \mathbb{R}^{H \times H}$ is a projection matrix for a specific feedback f_j. In the embedding space, we compute the match score between the recommendation embedding $\boldsymbol{i}_t$ and each memory cell $\boldsymbol{i}_{f_j,j}$ by taking the inner product followed by a softmax:

$$\alpha_j = \text{Softmax}(\boldsymbol{i}_t^\top \boldsymbol{i}_{f_j,j}), \tag{13}$$

where $\text{Softmax}(z_j) = \frac{\exp(z_j)}{\sum_l \exp(z_l)}$, $\boldsymbol{i}_t$ is the embedding of recommendation. Defined in this way, α is a probability vector over the inputs. Finally, the state $\boldsymbol{s}_t$ is formed by concatenating customers' embedding $\boldsymbol{u} \in \mathbb{R}^U$ and α weighted sum of inputs as:

$$\boldsymbol{s}_t = \left[\boldsymbol{u}, \sum_{j=0}^{t-1} \alpha_j \boldsymbol{i}_{f_j,j} \right], \tag{14}$$

where $[\cdot,\cdot]$ means concatenation operation. Given it, the contextual feature for decision making is formulated as:

$$\boldsymbol{\phi}(s_t, i_t) = \boldsymbol{W}_c[\boldsymbol{s}_t, \boldsymbol{i}_t] + \boldsymbol{b}_c, \tag{15}$$

where $\boldsymbol{W}_c$ and $\boldsymbol{b}_c$ are the weight and bias terms.

4.2 The Q-value Network

The approximation of Q-value is accomplished by the inner products of the dense state embedding with a weight vector as follows:

$$Q(s_t, i_t; \theta_Q) = \boldsymbol{w}_q^\top \boldsymbol{\phi}(s_t, i_t) + b_q. \tag{16}$$

Table 1: Statistics of the datasets.

Dataset	#Customers	#Items	# Total Behaviors	# Behaviors per Customer	# Behaviors per Item
Taobao	986,240	4,161,799	100,144,665	101.5419	24.0628
Retailrocket	81,620	103,873	948,537	11.6214	9.1317

Here, $\boldsymbol{w}_q$ and b_q are the weight vector and bias terms. The update of Q-value network follows the Equation (11).

4.3 The World Model

As mentioned in Section 3.1, the world model imitates the customer's behaviors and provides the reward function for policy learning, where the reward function is determined by two parts: 1) the customer's feedback (i.e.,clicked or not); 2) customer's leaving (i.e.,not leaving means more future reward). Therefore, the world model uses the s_t and i_t as input and generates customer's response f_t and a binary variable e_t, which indicates whether the session terminates. This generation is accomplished using the world model $M(s, i; \theta_M)$ (shown in Figure 2(c)) as follows:

$$\boldsymbol{f}_t = \text{Softmax}(\boldsymbol{W}_f \boldsymbol{\phi}(s_t, i_t) + \boldsymbol{b}_f), \tag{17}$$

$$e_t = \text{Sigmoid}(\boldsymbol{w}_e^\top \boldsymbol{\phi}(s_t, i_t) + b_e), \tag{18}$$

where $\boldsymbol{W}_f$, $\boldsymbol{b}_f$, $\boldsymbol{w}_e$ and b_e are the weights and bias. Figure 2 is a multi-task neural network that combines two classification tasks of simulating f_t and e_t, respectively. The parameter θ_M is thus updated by setting δ_t in Equation (9) with cross-entropy as:

$$\delta_t(\theta_M) = \boldsymbol{f}_t \log \boldsymbol{f}_t(\theta_M) + e_t \log e_t(\theta_M) + (1 - e_t) \log(1 - e_t(\theta_M)). \tag{19}$$

5 EXPERIMENTS

In this section, we perform empirical evaluations of our proposed PDQ on two large collections of real-world customer logs extracted from e-commerce platforms. The source code can be found at Github: https://github.com/zoulixin93/pseudo_dyna_q.

5.1 Experimental Settings

Dataset. We adopt the following two public datasets in our experiments.

- **Taobao**[4]: Taobao is the largest E-commerce platform in China. Taobao dataset contains a subset of customer behaviors including click, purchase, adding item to shopping cart and item favoring from November 25, 2017 to December 03, 2017.
- **Retailrocket**[5]: Retailrocket is a dataset collected from a real-world ecommerce website over a period of 4.5 months, which contains customers behaviour data (i.e. events like "clicks", "add to carts" and "transactions").

Detailed statistic information, including the number of customers, items and behaviors, of these datasets is given in Table 1.

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

[4]https://tianchi.aliyun.com/datalab/dataSet.html?dataId=649
[5]https://www.kaggle.com/retailrocket/ecommerce-dataset/home

- BPR [26]: It optimizes the matrix factorization model with a pairwise ranking loss. This is a popular method for item recommendation. However, it ignores the sequential information of recommendation and cannot optimize the long-term reward in recommendation.
- FPMC [27]: It learns a transition matrix based on underlying Markov chains. Sequential behaviors are modeled only between the adjacent transactions.
- GRU4Rec [17]: This is a representative approach that utilizes RNN to learn the dynamic representation of customers and items in recommender systems.
- NARM [20]: This is a state-of-the-art approach in personalized trajectory-based recommendation with RNN models. It uses attention mechanisms to determine the relatedness of the past purchases in the trajectory for the next purchase.
- DQN-R [48]: It is an elegant and concise off-policy reinforcement learning method, which has been employed for sequential e-commerce recommendation in [48].
- DDPG-KNN [11]: DDPG is an actor-critic, model-free framework. In [11], it has been adapted for discrete recommendation by combining DDPG with an approximate KNN method.
- PDQ: Our PDQ model that utilizes a world model to imitate customer's feedback and learns an offline policy by combining planning and direct RL. To verify the effect of different components, we also test the following degenerated PDQ models:
  - PDQ(N): The naive PDQ, which separately optimizes the world model and recommendation policy and does not handle the distribution mismatch between the logging policy and recommendation policy.
  - PDQ(IM): It iteratively trains the world model and offline policy, and employs clipping importance sampling to deal with the mismatch between logging policy and recommendation policy.
  - PDQ(IM+R): Our integrated PDQ(IM+R) model, which regularizes the generalization error by minimizing the distribution divergence between the logging policy and recommendation policy.

Parameter Setting. The state tracker has one hidden layer and 200, 100 hidden units for Taobao and Retailrocket respectively. All the baseline models share the same layer and hidden nodes configuration for the neural networks. ϵ-greedy is always applied for exploration 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 buffer size of $\mathcal{M}$ is set as 10000. The target value function is updated at the end of each epoch. In each epoch, the mini-batch size is 256. The networks are trained with SGD [3] with a learning rate of 0.005. Unless otherwise specified, the hyper-parameter λ for regularization is 0.01. We used TensorFlow to implement the pipelines and trained networks with an Nvidia GTX 1080 ti GPU cards. All the experiments are obtained by an average of 5 repeat runs.

5.2 Online Testing Experiments

5.2.1 Simulation Setting. To perform evaluation of RL methods on ground-truth, a straightforward way is to collect a large logged dataset and evaluate the learned policy through online A/B test, which, however, could be too expensive and commercially risky for the platform. Similar to [5, 11, 28], we demonstrate how the proposed method would perform on a real world recommender system by constructing a simulated customer model utilizing data from Taobao and Retailrocket. Without loss of generality, we regard the *"clicks", "add to carts", "transactions"* as positive feedback (clicks) and assume a standard rank-H-restricted matrix factorization model [26] $Pr(\text{click}|u,i) = \text{Sigmoid}\left(\boldsymbol{u}_u^\top \boldsymbol{i}_i\right)$ for customers' clicks, where $\boldsymbol{u}_u, \boldsymbol{i}_i \in \mathbb{R}^H$ are the latent factors learned by fitting Taobao and Retailrocket dataset through BPR-MF. The ranking H for Taobao and Retailrocket are set as $[100, 50]$, learning rate is $[0.1, 0.1]$, and the maximum iteration is 2000 before convergence.

Apart from the feedback, we need to simulate customers' patience on the platform – when to end the trajectory after losing patience. Similar to [11], we assume that the ending probability is correlated with customers' feedback. In other words, if the presented item is accepted then the trajectory has a small ending probability; if the item is not accepted then the trajectory has a higher ending probability. However, this assumption is not good enough for building a real customer simulator, since under this assumption, the learned policy could repeatedly recommend similar items, which may annoy customers. For this reason, we further assume that the ending probability is also related to the diversity of recommended items (i.e. diverse recommendations are more attractive to customers [9]). Formally, based on the intuitions, the ending probability for a trajectory is set as:

$$Pr(\text{ending}|u, i_0, i_1, \ldots, i_t) = 1 - Pr(\text{click}|u, i_t)\text{Sigmoid}\left(\frac{2}{t \cdot (t-1)} \sum_{m,k \in \{1,2,\ldots,t\}} \text{entropy}(\boldsymbol{i}_m, \boldsymbol{i}_k)\right),$$

where $\text{entropy}(\boldsymbol{i}_m, \boldsymbol{i}_k) = \sum_{j=1}^{H} \boldsymbol{i}_{m,j} \log \frac{\boldsymbol{i}_{m,j}}{\boldsymbol{i}_{k,j}}$ measures the distance between item m and item k.

5.2.2 Evaluation Setting. The principle of our evaluation is to recover the real offline learning and recommendation scenario. To this end, we first sample $9.8M$ and $891k$ trajectories as the training dataset from the simulators built using Taobao and Retailrocket datasets. Here, the logging policy π_b is set as $\pi_b(i|s) = \text{Softmax}(\frac{\boldsymbol{u}_u^\top \boldsymbol{i}_i}{\tau})$, where τ is the temperature to control the performance of logging policy. In real learning tasks, π_b should not be too random because online platforms usually spend a lot of effort in designing their recommender systems to satisfy customers' demands. Therefore, we set the $\tau = 5$, to ensure a medium level logging policy. After that, we train the model using the offline trajectories and evaluate the learned policies on the simulator.

5.2.3 Main Results. Results by an average of 5 repetitive experiment runs are obtained and we report the three metrics in Table 2 and Figure 3. For each recommendation agent, we report its results in terms of the average clicks per trajectory (Clicks), average diversity of recommendations (Diversity) and the average number of interactions (Horizons) [11, 23], which measure the goodness of the whole trajectory recommendations. Figure 3 shows the learning curves on average clicks per session of different kinds of recommendation agents on these two datasets. From Table 2 and Figure 3, we have following observations:

Table 2: Performance comparison of different recommendation agents on offline learning tasks.

Agents	Taobao									Retailrocket								
	Epoch=1000			Epoch=10000			Epoch=20000			Epoch=1000			Epoch=10000			Epoch=20000		
	Clicks	Diversity	Horizon	Clicks	Diversity	Horizon	Clicks	Diversity	Horizon	Clicks	Diversity	Horizon	Clicks	Diversity	Horizon	Clicks	Diversity	Horizon
BPR	1.3344	0.0002	1.8328	1.8664	0.0003	1.9072	1.8433	0.0004	1.8948	1.4949	0.0007	1.8293	1.8055	0.0006	1.8937	1.8013	0.0008	1.8967
FPMC	1.8227	0.0424	2.0910	1.8428	0.0443	2.1170	1.8394	0.0441	2.0903	1.8361	0.0782	2.3712	1.8327	0.0501	2.2040	1.8127	0.0405	2.1505
GRU4Rec	1.4264	0.0866	2.2773	1.8852	0.0475	2.1290	1.8421	0.0435	2.0985	1.7380	0.1168	2.4046	1.9460	0.0867	2.3776	1.9272	0.0831	2.3629
NARM	1.2532	0.1107	2.3587	1.9921	0.0884	2.4386	1.9926	0.0747	2.3483	1.8785	0.0854	2.3901	2.0716	0.1001	2.5663	1.9466	0.0867	2.4410
DQN-R	1.9598	**0.2583**	**3.3379**	0.8777	0.0008	1.7068	0.8810	0.0006	1.7117	1.8861	0.1344	2.7174	1.5397	0.0727	2.2252	0.6040	0.0005	1.6368
DDPG-KNN(k=1)	0.8695	0.0004	1.6852	0.8662	0.0006	1.6830	0.8678	0.0004	1.6997	0.5924	0.0009	1.6262	0.6251	0.0006	1.6449	0.5957	0.0008	1.6274
DDPG-KNN(k=0.1N)	**2.0815**	0.0314	2.1111	0.8722	0.0519	1.7013	0.8745	0.0007	1.7079	2.0502	0.0876	2.3329	1.4256	0.0433	2.0099	0.6050	0.0017	1.6375
DDPG-KNN(k=N)	1.9928	0.0269	2.0815	0.8645	0.0219	1.6869	0.8624	0.0157	1.6899	1.5953	0.0457	2.0977	1.0083	0.0223	1.8384	0.8942	0.0008	1.7740
PDQ (N)	1.7909	0.2405	3.1165	2.1217	0.3111	3.5264	1.7830	0.2491	3.1420	1.3420	0.1002	2.3881	2.0193	0.1164	2.7436	2.2471	0.1373	2.9560
PDQ (IM)	1.8862	0.2558	3.1255	2.4326	0.3733	3.9725	2.5801	0.3285	3.8945	2.2351	0.1056	2.7528	2.2478	0.1431	3.0465	2.6020	0.1442	3.2629
PDQ (IM+R)	1.8855	0.2232	2.9952	**2.5885***	**0.4206***	**4.2816***	**2.7998***	**0.4182***	**4.3168***	**2.4424***	**0.1358***	**2.9854***	**2.9920***	**0.2117***	**3.5191***	**2.8131***	**0.1988***	**3.5036***

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

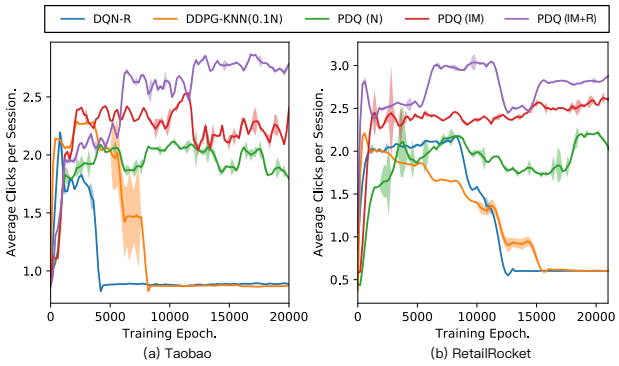

Figure 3: Learning curves of PDQ agents and baseline models.

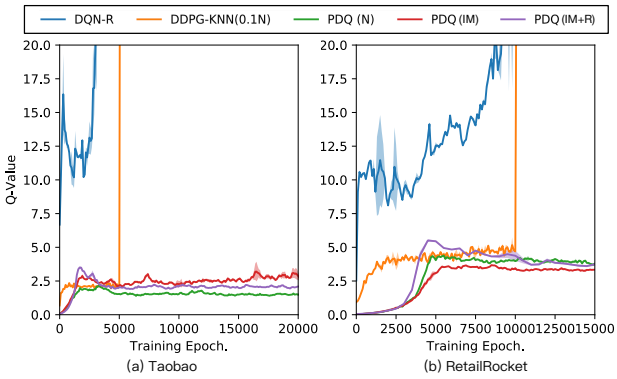

Figure 4: The average Q-value of different recommendation agents.

(1) We observed that non-RL methods (i.e. BPR), compared with RL methods, are very stable but at a lower performance level in the offline learning task. This is because they mainly focus on the item-level performance, and are unable to improve the overall performance on trajectory level. Intuitively, diversity is an implicit metric that can improve the horizon of interactions, which is helpful to improve the overall clicks. However, compared with PDQ, non-RL methods' performances on diversity are bad because they can not optimize the overall clicks in a long-term view and repeatedly recommend similar items for customers.

(2) Directly applying off-policy RL methods on offline training will result in the failure of the learned policy. Figure 3 shows that DQN-R and DDPG-KNN can improve the clicks at the beginning but rapidly degrade after several rounds of training. Additionally, based on the results in Table 2, the actor of DDPG-KNN(1) (i.e. purely depending on the actor-network for the recommendation) does not work at all. This phenomenon has been referred to as the well-known Deadly Triad Problem in the community, which indicates the intrinsic failure caused by the explosion of Q-value during the training process.

(3) By addressing the selection bias problem through learning a simulator, the proposed PDQ(IM+R) agents consistently outperform baselines with a statistically significant margin.

5.3 Analysis

In this section, we further analyze the effectiveness of the proposed framework for the offline recommendation task based on the two datasets.

5.3.1 The Deadly Triad in Recommendation. As previously mentioned, directly deploying TD based methods may not be safe in offline training tasks (i.e. the headache of Deadly Triad[3]). To study the problem of Deadly Triad in the recommendation, the average Q-value over training epochs (averaged over 5 repeat experiments) has been shown in Figure 4. We can see that: **(1)** the Q-value of DQN-R and DDPG-KNN (0.1N) are not stable in offline training. The Q-value of DQN-R quickly blows up after several iterations. DDPG-KNN's Q-value seems stable in the beginning but it also changes to infinity at a certain point. This phenomenon may be caused by the difference in estimating the next state value between these two methods. In DQN-R, the maximal next Q-value $Q^\pi(s_t, a_t)$ is chosen for TD estimation. However, in DDPG-KNN, the sampling action value $Q^\pi(s_t, a_t)$ is used. The max operation accelerates the blow-up of Q-value. **(2)** Compared with directly learning an offline policy, PDQ can effectively solve this problem by building a world model bridging the gap between offline and online training. In Figure 4, the Q-value of PDQ converges to a stable value at the end.

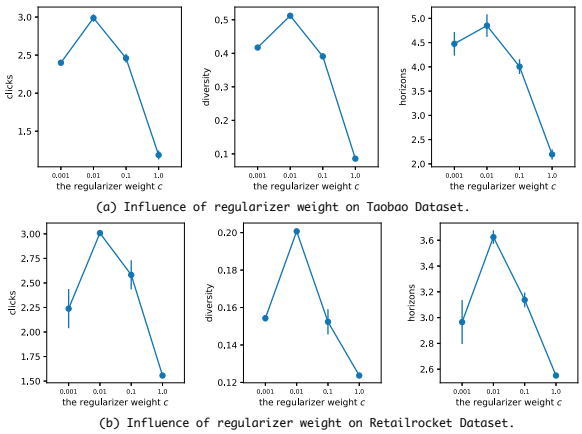

Figure 5: The influence of regularize weight on performance.

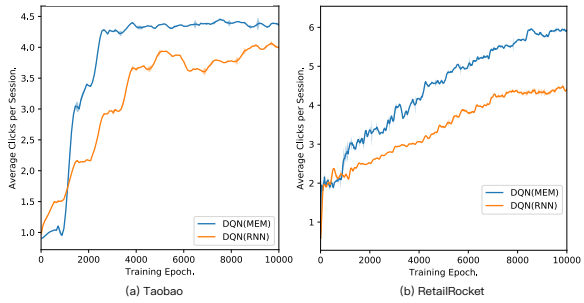

Figure 6: Learning curves of DQN with Memory Network and GRU Network.

5.3.2 Influence of Regularizer Weight. We investigate how the performance varies w.r.t. the regularizer weight λ in Equation (8). The Clicks, Diversity, and Horizons are shown in Figure 5 with different λ ranging from 0.001 to 1. The experimental results demonstrate that if the regularizer is too large, it will hurt performance. On the contrary, a too-small regularizer will have a limited influence on the variance of the learned world model, which is consistent with our intuition about the regularizer.

5.3.3 The Effectiveness of State Tracker. We propose a memory-based neural architecture to track customers' interests in the interactive recommendation. To verify its effectiveness, we train two online DQN agents on the simulator with different state tracker: one with our proposed memory-based state tracker, named DQN(MEM); The other DQN(GRU) employs GRU as function approximation, which has been widely used in recommendation tasks [17, 20]. Figure 6 presents the learning curve on two datasets. DQN(MEM) performs better than DQN(GRU), which suggests that the proposed memory and self-attention based state tracker is more capable of modeling complex interactions than GRU as function approximation.

6 RELATED WORK

Recommender systems have attracted a lot of attentions from the research community and industry. Being supervised by the history records is the common practice in majority models, including *traditional factorization methods* [4, 14, 19, 25], *deep neural models*, such as multilayer perceptron [8], denoising auto-encoders [44], convolutional neural network (CNN) [2, 38], recurrent neural network (RNN) [13, 15, 20, 43], memory network [7] and attention architectures [1, 6]. Based on the partial observed history dataset, these existing models usually predict a customer's feedback by a learning function to maximize some well-defined evaluation metrics in ranking, such as Recall, Precision and NDCG [10]. However, most of them are myopic because the learned policies are greedy with estimating customers' feedback and unable to optimize customers' feedback in the long run.

Recently, *reinforcement learning-based approaches* have attracted a lot of attention in recommender systems. The core idea of RL models is learning an effective policy to maximize the expected reward in the long run. The most common approach is learning the policy by learning empirical rewards from interaction with real customers, e.g. contextual bandit (i.e. 1-horizon MDP) based recommender methods [21, 24, 42, 45], Markov Decision Process (MDP) based recommendation methods [5, 11, 46–51]. Contextual bandit models handle the notorious explore/exploit dilemma in online environment for the cold start problem; while MDP based methods design different neural network architectures to extract interactive information from customer status. Due to the fact that learning a policy online by interacting with real customers may lead to poor customer experiences [37]. Hence the most common way is to utilize the history data to train an offline policy, which enables the recommender system to get past its blundering stage in an offline environment without putting anyone in an unfriendly experience.

The *offline policy learning*, which is a tempting challenge, has attracted great interest in RL community to design stable and efficient learning algorithms. Algorithms in existing models can be classified into *Monte Carlo* (MC) and *temporal-difference* (TD) methods. Due to the efficiency problem of MC, its usage has been limited to off-policy evaluation [12, 18, 39]. As for TD, it has a black cloud, i.e. deadly triad[3], hanging over its head. Current solutions for Deadly Triad are limited to linear function approximation, such as GTD2 [36] and GRetrace(λ) [40]. However, none of them can be applied to complex function approximation, i.e. neural networks.

7 CONCLUSION

In this work, we investigated offline policy learning in recommender systems, which is usually more practical than online policy learning in practice. An offline policy learning strategy—Pesudo Dyna-Q (PDQ) was proposed for interactive recommendation. PDQ performs offline policy learning through both model-based indirect and direct offline learning, where the world model is introduced to simulate the environments and assist TD-based policy improvement. We also provided a general error analysis of the world model's risk function, and based on the analysis, the world model is designed to keep adaptively optimized for specific recommendation policies,

during policy learning. The TD based Q-Learning in offline setting is hence able to prevent from instability of convergence, and perform policy improvement effectively, via both logged experiences and querying the simulator. Extensive experiments on two real world large scale datasets showed that the instantiated PDQ based on neural networks outperforms state-of-the-art methods noticeably.

REFERENCES

[1] Ting Bai, Jian-Yun Nie, Wayne Xin Zhao, Yutao Zhu, Pan Du, and Ji-Rong Wen. 2018. An attribute-aware neural attentive model for next basket recommendation. In *SIGIR'18*. ACM, 1201–1204.

[2] Ting Bai, Lixin Zou, Wayne Xin Zhao, Pan Du, Weidong Liu, Jian-Yun Nie, and Ji-Rong Wen. 2019. CTRec: A Long-Short Demands Evolution Model for Continuous-Time Recommendation. In *SIGIR'19*. ACM, 675–684.

[3] Léon Bottou. 2012. Stochastic gradient descent tricks. In *Neural networks: Tricks of the trade*. Springer, 421–436.

[4] 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*. 381–389.

[5] Haokun Chen, Xinyi Dai, Han Cai, Weinan Zhang, Xuejian Wang, Ruiming Tang, Yuzhou Zhang, and Yong Yu. 2018. Large-scale Interactive Recommendation with Tree-structured Policy Gradient. *arXiv preprint arXiv:1811.05869* (2018).

[6] Weijian Chen, Yulong Gu, Zhaochun Ren, Xiangnan He, Hongtao Xie, Tong Guo, Dawei Yin, and Yongdong Zhang. 2019. Semi-supervised user profiling with heterogeneous graph attention networks. In *IJCAI'19*. AAAI Press, 2116–2122.

[7] Xu Chen, Hongteng Xu, Yongfeng Zhang, Jiaxi Tang, Yixin Cao, Zheng Qin, and Hongyuan Zha. 2018. Sequential recommendation with user memory networks. In *WSDM'18*. ACM, 108–116.

[8] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. 2016. Wide & deep learning for recommender systems. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*. ACM, 7–10.

[9] Peizhe Cheng, Shuaiqiang Wang, Jun Ma, Jiankai Sun, and Hui Xiong. 2017. Learning to recommend accurate and diverse items. In *WWW'17*. 183–192.

[10] Charles LA Clarke, Maheedhar Kolla, Gordon V Cormack, Olga Vechtomova, Azin Ashkan, Stefan Büttcher, and Ian MacKinnon. 2008. Novelty and diversity in information retrieval evaluation. In *SIGIR'08*. ACM, 659–666.

[11] 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).

[12] Mehrdad Farajtabar, Yinlam Chow, and Mohammad Ghavamzadeh. 2018. More Robust Doubly Robust Off-policy Evaluation. *ICML'18* (2018).

[13] Yulong Gu, Zhuoye Ding, Shuaiqiang Wang, and Dawei Yin. 2020. Hierarchical User Profiling for E-commerce RecommenderSystems. In *WSDM'20*. ACM.

[14] Patrik O Hoyer. 2004. Non-negative matrix factorization with sparseness constraints. *Journal of machine learning research* 5, Nov (2004), 1457–1469.

[15] Chao Huang, Xian Wu, Xuchao Zhang, Chuxu Zhang, Jiashu Zhao, Dawei Yin, and Nitesh V Chawla. 2019. Online Purchase Prediction via Multi-Scale Modeling of Behavior Dynamics. In *SIGKDD'19*. ACM, 2613–2622.

[16] Edward L Ionides. 2008. Truncated importance sampling. *Journal of Computational and Graphical Statistics* 17, 2 (2008), 295–311.

[17] Dietmar Jannach and Malte Ludewig. 2017. When recurrent neural networks meet the neighborhood for session-based recommendation. In *RecSys'17*. ACM, 306–310.

[18] Nan Jiang and Lihong Li. 2015. Doubly robust off-policy value evaluation for reinforcement learning. *ICML'15* (2015).

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

[20] 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.

[21] 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.

[22] 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).

[23] Baolin Peng, Xiujun Li, Jianfeng Gao, Jingjing Liu, Kam-Fai Wong, and Shang-Yu Su. 2018. Deep Dyna-Q: Integrating Planning for Task-Completion Dialogue Policy Learning. *ACL'18* (2018).

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

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

[26] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian personalized ranking from implicit feedback. In *UAI'09*. AUAI Press, 452–461.

[27] Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. 2010. Factorizing personalized markov chains for next-basket recommendation. In *WWW'10*. ACM, 811–820.

[28] David Rohde, Stephen Bonner, Travis Dunlop, Flavian Vasile, and Alexandros Karatzoglou. 2018. RecoGym: A Reinforcement Learning Environment for the problem of Product Recommendation in Online Advertising. *RecSys'18* (2018).

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

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

[31] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. Mastering the game of Go with deep neural networks and tree search. *nature* 529, 7587 (2016), 484.

[32] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. 2017. Mastering the game of Go without human knowledge. *Nature* 550, 7676 (2017), 354.

[33] Sainbayar Sukhbaatar, Jason Weston, Rob Fergus, et al. 2015. End-to-end memory networks. In *NIPS'15*. 2440–2448.

[34] Richard S Sutton. 1991. Dyna, an integrated architecture for learning, planning, and reacting. *ACM SIGART Bulletin* 2, 4 (1991), 160–163.

[35] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.

[36] Richard S Sutton, Hamid R. Maei, and Csaba Szepesvári. 2009. A Convergent O(n) Temporal-difference Algorithm for Off-policy Learning with Linear Function Approximation. In *NIPS'09*. 1609–1616.

[37] Adith Swaminathan and Thorsten Joachims. 2015. Counterfactual risk minimization: Learning from logged bandit feedback. In *ICML'15*. 814–823.

[38] Jiaxi Tang and Ke Wang. 2018. Personalized top-n sequential recommendation via convolutional sequence embedding. In *WSDM'18*. ACM, 565–573.

[39] Philip S Thomas, Georgios Theocharous, and Mohammad Ghavamzadeh. 2015. High-Confidence Off-Policy Evaluation.. In *AAAI'15*. 3000–3006.

[40] Ahmed Touati, Pierre-Luc Bacon, Doina Precup, and Pascal Vincent. 2017. Convergent tree-backup and retrace with function approximation. *ICML'17* (2017).

[41] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *NIPS'17*. 5998–6008.

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

[43] 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.

[44] Yao Wu, Christopher DuBois, Alice X Zheng, and Martin Ester. 2016. Collaborative denoising auto-encoders for top-n recommender systems. In *WSDM'16*. ACM, 153–162.

[45] 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.

[46] Xiangyu Zhao, Long Xia, Jiliang Tang, and Dawei Yin. 2019. Deep reinforcement learning for search, recommendation, and online advertising: a survey by Xiangyu Zhao, Long Xia, Jiliang Tang, and Dawei Yin with Martin Vesely as coordinator. *ACM SIGWEB Newsletter* Spring (2019), 4.

[47] 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.

[48] 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.

[49] 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. In *WWW'18*. 167–176.

[50] Lixin Zou, Long Xia, Zhuoye Ding, Jiaxing Song, Weidong Liu, and Dawei Yin. 2019. Reinforcement Learning to Optimize Long-term User Engagement in Recommender Systems. In *SIGKDD'19*. ACM, 2810–2818.

[51] 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.