# Generative Inverse Deep Reinforcement Learning for Online Recommendation

Xiaocong Chen
University of New South Wales
Sydney, Australia
xiaocong.chen@unsw.edu.au

Lina Yao
University of New South Wales
Sydney, Australia
lina.yao@unsw.edu.au

Aixin Sun
Nanyang Technological University
Singapore
axsun@ntu.edu.sg

Xianzhi Wang
University of Technology Sydney
Sydney, Australia
xianzhi.wang@uts.edu.au

Xiwei Xu
Data 61, CSIRO
Sydney, Australia
xiwei.xu@data61.csiro.au

Liming Zhu
Data 61, CSIRO
Sydney, Australia
liming.zhu@data61.csiro.au

## ABSTRACT

Deep reinforcement learning enables an agent to capture users' interest through dynamic interactions with the environment. It uses a reward function to learn user's interest and to control the learning process, attracting great interest in recommendation research. However, most reward functions are manually designed; they are either too unrealistic or imprecise to reflect the variety, dimensionality, and non-linearity of the recommendation problem. This impedes the agent from learning an optimal policy in highly dynamic online recommendation scenarios. To address the above issue, we propose a generative inverse reinforcement learning approach that avoids the need of defining an elaborative reward function. In particular, we model the recommendation problem as an automatic policy learning problem. We first generate policies based on observed users' preferences and then evaluate the learned policy by a measurement based on a discriminative actor-critic network. We conduct experiments on an online platform, VirtualTB, and demonstrate the feasibility and effectiveness of our proposed approach via comparisons with several state-of-the-art methods.

## CCS CONCEPTS

**•Information systems → Recommender systems; •Computing methodologies → Reinforcement learning; Neural networks;**

## KEYWORDS

Deep Reinforcement Learning; Deep Learning; Recommender System

**ACM Reference format:**
Xiaocong Chen, Lina Yao, Aixin Sun, Xianzhi Wang, Xiwei Xu, and Liming Zhu. 2016. Generative Inverse Deep Reinforcement Learning for Online Recommendation. In *Proceedings of ACM Conference, Washington, DC, USA, July 2017 (Conference'17),* 10 pages.
DOI: 10.1145/nnnnnnn.nnnnnnn

## 1 INTRODUCTION

Online recommendation involves real-time interaction between user and system. It differs from traditional recommendation problems in facing a dynamic environment where user's preference may change rapidly. In recent decades, deep reinforcement learning (DRL) has become a promising solution to online recommendation, given its ability to learn optimal strategies from interactions. This is in contrast to traditional methods like multi-armed bandit, which assume users' preference to be stable and thus cannot meet the requirements for online recommendation. DRL-based recommendation systems cover three categories: deep Q-learning based, policy gradient based, and hybrid methods. Deep Q-learning aims to find the best step by maximizing a Q-value over all possible actions. [31] first introduced DRL into recommendation systems for news recommendation; then, [5] introduced a robust Q-learning to handle dynamic environments for online recommendation. However, Q-learning based methods suffer the *agent stuck problem*, i.e., it requires the maximise operation over the action space, which becomes infeasible when the action space is extremely large. Policy gradient based methods can mitigate the agent stuck problem [4]. They use the average reward as a guideline, yet they may treat bad actions as good actions, making the algorithm hard to converge [20]. Hybrid methods combine the advantages of Q-learning and policy gradient. A popular hybrid method is actor-critic network [16], which adopts policy gradient on an actor network and Q-learning on a critic network to achieve nash equilibrium on both networks. Until now, actor-critic network has been widely applied to DRL-based recommendation systems [6, 19].

Despite differences, all existing DRL-based methods rely on well-designed context-dependent reward functions, as shown in Fig 1 (a). They commonly use 'User's click or not' is the main metric to measure recommendation quality and to define the reward function. They suffer limited generalization ability because, in many cases, the reward function cannot be easily defined, due to dynamic environments and various factors that affect user's interest [26]. Besides, they generally consider both user's preference and user's actions (e.g., click-through, rating or implicit feedback) in the reward function, assuming that the reward is determined by the current chosen item and user's action is unaffected by the recommended items. Those assumptions may not hold for online recommendation [26]. Another challenge comes from the time-consuming recommendation policy search using reinforcement learning [8]. The state space (i.e., all candidate items in which users might be interested) and action space (i.e., all the actions for candidate items) in a recommendation problem might be huge, and a traditional reinforcement learning based method needs to iterate all the possible combinations

*Conference'17, Washington, DC, USA*
2016. 978-x-xxxx-xxxx-x/YY/MM…$15.00
DOI: 10.1145/nnnnnnn.nnnnnnn

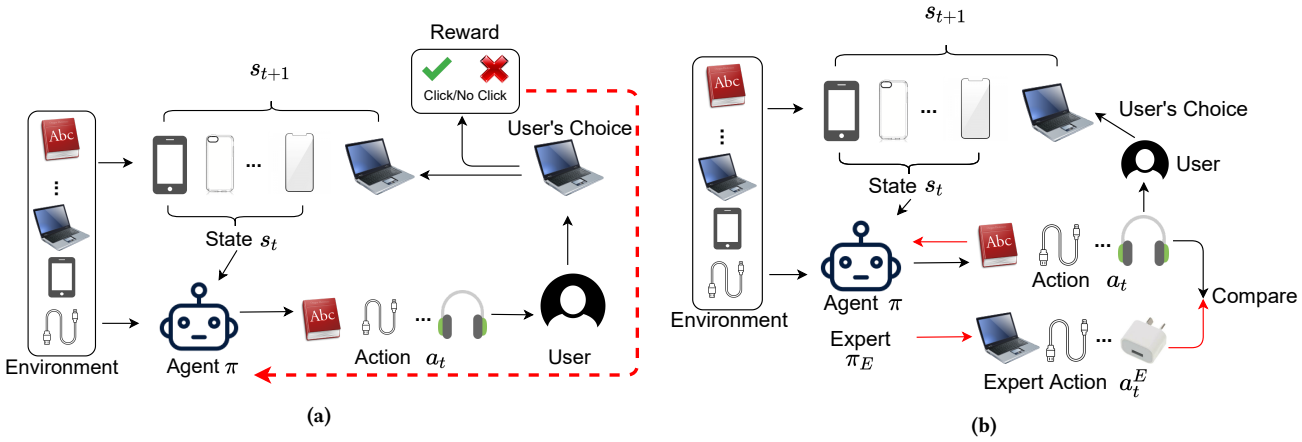


**Figure 1: General workflow of current reinforcement learning based recommendation system (a) which relies on reward function to guide the agent. The inverse reinforcement learning based approach (b) which does not require the reward function guiding. We use red color to represent how those two approaches update the policy.**

to figure out the best policy. Recent achievements on generative adversarial imitation learning (GAIL) [12] provide a solution to learning from expert without needing to know the exact reward function. However, GAIL is designed for traditional reinforcement learning environments that involve only static and countable states. Besides, GAIL only works well on survival or explore tasks, which have different goals from recommendation tasks.

Targeting at the above challenges, we aim to enable the agent to infer a reward function from user's behaviors via inverse reinforcement learning and learn the recommendation policy directly from user behaviors efficiently and adaptively, as shown in Fig 1 (b). We propose a generative inverse reinforcement learning approach for inferring an implicit reward function from user behaviors adaptively. The approach can measure the performance of the current recommendation policy and update the current policy with the expert policy in the discriminator, thus avoiding needing to define a reward function for online recommendation. In particular, we transform inverse deep reinforcement learning into a generator to augment a diverse set of state-action pairs. Under this generative strategy, our method can achieve better generalization ability under complex online recommendation conditions. In a nutshell, we make the following contributions:

- We propose generative inverse reinforcement learning to automatically learn reward function for online recommendation. To the best of our knowledge, this is the first work to decouple the reward function and the agent for online recommendation.
- We design a novel actor-discriminator network module, which takes a discriminator as the critic network and a novel actor-critic network as the actor network, to implement the proposed framework. The module is model-free and can be easily generalized to a variety of scenarios.
- We conduct experiments on a virtual online platform, VirtualTB, to demonstrate the feasibility and effectiveness of the proposed approach. Our proposed method achieves a higher click-through-rate than several state-of-the-art methods.

## 2 PROBLEM FORMULATION

**Online Recommendation** Online recommendation differs from offline recommendation in dealing with real-time interactions between users and the recommendation system. The system needs to analyse users' behaviors and updates the recommend policy dynamically. The objective is to find a solution that best reflects those interactions and apply it to the recommendation policy.

**Reinforcement Learning-based Recommendation** learns from interactions through a Markov Decision Process (MDP). Given a recommendation problem consisting of a set of users $\mathcal{U} = \{u_0, u_1, \cdots u_n\}$, a set of items $\mathcal{I} = \{i_0, i_1, \cdots i_m\}$ and user's demographic information $\mathcal{D} = \{d_0, d_1, \cdots, d_n\}$, MDP can be represented as a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$, where $\mathcal{S}$ denotes the state space, which is the combination of the subsets of $\mathcal{I}$; $\mathcal{A}$ is the action space, which represents agent's selection during recommendation based on the state space $\mathcal{S}$; $\mathcal{P}$ is the set of transition probabilities for state transfer based on the action received; $\mathcal{R}$ is a set of rewards received from users, which are used to evaluate the action taken by the recommendation system, with each reward being a binary value to indicate user's click; $\gamma$ is a discount factor $\gamma \in [0, 1]$ for balancing the future reward and current reward.

Given a user $u_0$ and the state $s_0$ observed by the agent (or the recommendation system), which includes a subset of item set $\mathcal{I}$ and user's demographic information $d_0$, a typical recommendation iteration for user $u_0$ goes as follows. First, the agent makes an action $a_0$ based on the recommend policy $\pi_0$ under the observed state $s_0$ and receives the corresponding reward $r_0$. Then, the agent generates a new policy $\pi_1$ based on the received reward $r_0$ and determines the new state $s_1$ based on the probability distribution $p(s_{new}|s_0, a_0) \in \mathcal{P}$. The cumulative reward after $k$ iterations is as

follows:

$$r_c = \sum_{k=0} \gamma^k r_k$$

**Inverse Reinforcement Learning-based Online Recommendation** We propose inverse reinforcement learning without predefining a reward function for online recommendation. We aim to optimize the current policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$ to make recommendations that are most suitable for the user. Specifically, we model online recommendation as an MDP with a finite state set $\mathcal{S}$, a set of actions $\mathcal{A} = \{a_0, a_1, \cdots, a_t\}$, transition probabilities $\mathcal{P}(\mathcal{S}, \mathcal{A})$, and a discount factor $\gamma$. Suppose there exist an expert policy $\pi_E$ that can master any state $s \in \mathcal{S}$. The recommendation turns into the optimization problem of finding the policy $\pi$ that best approximates the expert policy $\pi_E$ across the cost function class $C$, by the following objective function [1]:

$$\underset{\pi}{\text{minimize}} \max_{c \in C} \mathbb{E}_\pi[c(s,a)] - \mathbb{E}_{\pi_E}[c(s,a)] \tag{1}$$

The cost function class $C$ is restricted to convex sets defined by the linear combination of a few basis functions $\{f_1, f_2, \cdots, f_k\}$. Hence, the corresponding feature vector for the state-action pair $(s,a)$ can be represented as $f(s,a) = [f_1(s,a), f_2(s,a), \cdots, f_k(s,a)]$. The expectation $\mathbb{E}_\pi[c(s,a)]$ is defined as (on $\gamma$-discounted infinite horizon):

$$\mathbb{E}_\pi[c(s,a)] = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t c(s_t, a_t)] \tag{2}$$

## 3 METHODOLOGY

The overall structure of our proposed approach is illustrated in Fig 2. It consists of two main components: recommendation policy learning, and discriminative actor-critic network for recommendation. *Recommendation policy learning* provides a theoretical analysis about how the optimal recommendation policy can be learned from the given expert policy $\pi_E$. Idealised recommendation policy $\pi$ is able to recommend items to users based on their dynamic preference. *discriminative actor-critic network for recommendation* constitutes the main structure of our approach. Besides the above components, we will present the optimization method and the corresponding training algorithm, which aims to limit the update step in optimizing the loss function to ensure that a new policy achieves better performance than the old one.

### 3.1 Recommendation Policy Learning

Recommendation policy learning is a crucial part of the proposed method. It aims to make the learned recommendation policy $\pi$ and expert policy $\pi_E$ as similar as possible by training a generative adversarial network (GAN) [10].

Learn a recommendation policy $\pi$ from expert policy $\pi_E$ can be formulate as the problem of matching two occupancy measures [1]. The occupancy measure $\rho$ can be defined as:

$$\rho_\pi(s,a) = \pi(s|a) \sum_{t=0}^{\infty} \gamma^t P(s_t = s|\pi) \tag{3}$$

Since the generator aims to generate the policy as similar to the expert policy as possible, we use the same way as described on GAIL [12] to bridge inverse reinforcement learning and GAN by making an analogy from the occupancy matching to distribution matching. To be specific, GA regularizer $\psi_{GA}$ is introduced to regularize the loss function. we can directly measure the difference between the policy and expert policy without needing to know the reward function. However, this method may suffer the ambiguity problem that many candidate policies $\pi_c$ can approximate the expert $\pi_E$. $\gamma$-discounted causal entropy $H(\pi)$ [3] is defined to resolve the ambiguity. The objective function can be expressed as:

$$\underset{\pi}{\text{minimize}} -\lambda H(\pi) + \underbrace{\psi_{GA}(\rho_\pi - \rho_{\pi_E})}_{D_{JS}(\rho_\pi, \rho_{\pi_E})} \tag{4}$$

where $\lambda$ is a factor with $\lambda \geq 0$. Note that Eq.(4) has the same goal as the GAN, i.e., finding the squared metric between distributions. More specifically, we have the following equivalence for Eq.(4):

$$\begin{aligned} &\underset{\pi}{\text{minimize}} -\lambda H(\pi) + \psi_{GA}(\rho_\pi - \rho_{\pi_E}) \equiv \min_\pi \max_D \mathcal{L}_\mathcal{D} \\ &\mathcal{L}_\mathcal{D} = \mathbb{E}_\pi[\log D(s,a)] + \mathbb{E}_{\pi_E}[\log(1 - D(s,a))] - \lambda H(\pi) \end{aligned} \tag{5}$$

Such a loss function would encourage agent to learn a policy $\pi$ towards expert-like regions of state-action space guided by $\log D(s,a)$. However, GAIL is only suitable for survival or exploration task. That is, for survival task, the reward used on GAIL is $\log D(s,a)$. It is always negative since $D(s,a) \in [0,1]$ which encourages agent to end current episode to stop more negative reward. For exploration task, the reward function is $-\log(1 - D(s,a))$, it is always positive and may lead to agent loop in the environment to collect more rewards. In recommendation system, agent is supposed to explore the environment and make recommendation based on observation. Hence, fully negative or positive is not suitable for recommendation. Indeed, we design a new reward function which can relief this problem and it is more suitable in recommendation task to evaluate recommendation policy:

$$R(s,a) = \log D(s,a) - \log(\max(\epsilon, 1 - D(s,a))) \tag{6}$$

Note that, in recommendation task, the agent only need to ensure the target item will be recommended; it is not essential to behave exactly same as the expert policy. Hence, we include user's feedback $r \in [0,1]$ as part of the reward function to ensure agent will not take extra step to update itself when user click one of the recommendations.

$$R(s,a) = \log D(s,a) - \log(\max(\epsilon, 1 - D(s,a))) + r \tag{7}$$

It is worth the mention that such reward function is used to evaluate the learned policy $\pi$ from the expert policy. It is distant from the reward function which used to reflect user's preference.

### 3.2 Discriminative Actor-Critic Network

The discriminative actor-critic network aims to make the recommendation based on the policy learned from the inverse reinforcement learning framework. Specifically, we take *advantage actor-critic network*, a variant of the actor-critic to constitute the main structure of our approach. Within this network, the actor uses the policy gradient to update the policy, and the critic uses Q-learning to evaluate the policy and provides feedback [16].

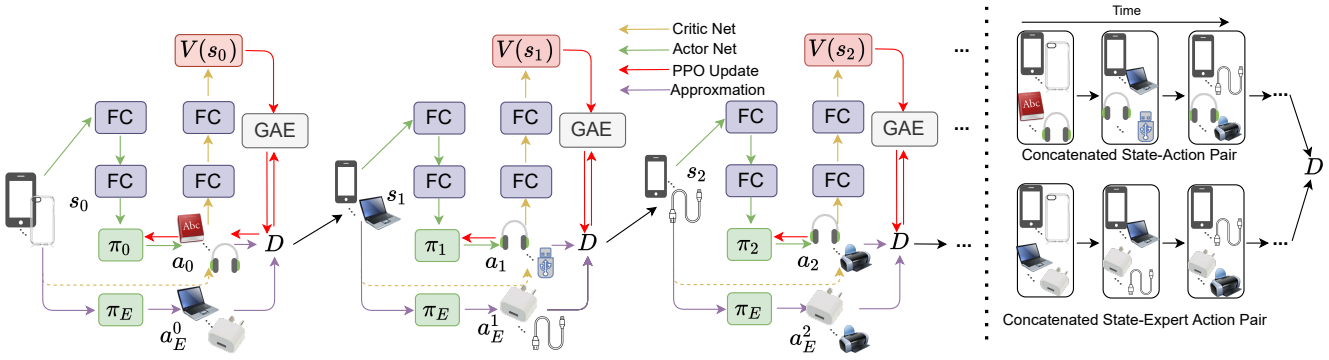


**Figure 2: The proposed framework. The actor-critic network generates the policy $\pi$ based on current state $s_t$. The discriminator takes state-action pair $(s_t, a_t)$ and expert state-action pair $(s_t, a_E^t)$ as input. The goal of the actor-critic network is to generate $\pi$ which can let discriminator classify it as an expert policy. The discriminator aims to distinguish the policy that is generated by actor-critic network. The input of GAE is the Value from the critic network and $c(s, a)$ from discriminator (we will present its details later). We circled one update episode which happened on agent.**

Given user's profile at timestamp $t$ (i.e., item list $\{i_0, \cdots, i_{t-2}, i_{t-1}\}$) and optional demographic information $\mathcal{D}$ (used to generate state $s_t$), the environment embeds user's recent interest and features into the latent space via neural network [6, 19]. Once the actor network gets the state $s_t$ from the environment, it feeds it to a network with two fully-connected layers with ReLU as the activation function. The final layer outputs the target policy function $\pi$ parameterized by $\theta$ to be updated together with discriminator $D$. Then, the critic network takes the input from the actor network with current policy $\pi_{\theta_t}$, which can be used for sampling to get the trajectory $(s_t, a_t)$. We concatenate and feed the state-action pair to two fully-connected layers with ReLU as the activation function. The critic network outputs a value $V(s_t, a_t) \in \mathbb{R}$ to be used to calculate the advantage, i.e., a value used for optimization (to be discussed later).

The discriminator $D$ is the key component of our approach. To build an end-to-end model and better approximate the expert policy $\pi_E$, we parameterize the policy as $\pi_\theta$ and clip the output of the discriminator so that $D : \mathcal{S} \times \mathcal{A} \rightarrow (0, 1)$ with weight $w$. The loss function of $D$ is $\mathcal{L}_D$. Besides, we use Adam [15] to optimize weight $w$ (the optimization for $\theta$ will be introduced later). Here, the discriminator $D$ can be treated as a local cost function to guide the policy update. Specifically, the policy will move toward expect-like regions (divided by $D$) in the latent space by minimizing the loss function $\mathcal{L}_D$, i.e., finding a point $(\pi, D)$ for Eq.(5) such that the equation output is minimal.

### 3.3 Training and Optimization

We use the actor-critic network as a policy network to be trained jointly with the discriminator. Therefore, the actor-critic network needs to update the policy parameter $\theta$ based on the discriminator. During this process, we limit the agent's step size to ensure the new policy is better than the old one. Specifically, we use trust region policy optimization (TRPO) [23] to update the policy parameter $\theta$ and formulate the TRPO problem as follows:

$$\max_\theta \frac{1}{T}\sum_{t=0}^{T}\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}A_t\right]$$
$$\text{subject to } D_{KL}^{\theta_{old}}(\pi_{\theta_{old}}, \pi_\theta) \leq \eta \tag{8}$$

where $A_t$ is the advantage function calculated by Generalized Advantage Estimation (GAE) [24] below:

$$A_t = \sum_{l=0}^{\infty}(\gamma\lambda_g)^l\Big[-V(s_t) + \sum_{l=0}^{\infty}\gamma^l r_{t+l}\Big] \tag{9}$$

where the reward $r_{t+l}$ is the $l$-step's test reward at timestamp $t$. The reward $r_{t+l}$ have two components which are reward returned by environment and the bonus reward calculated by Discriminator $D$ by using $\log(D(s_t, a_t))$. Considering the massive computation load of updating the TRPO via optimizing Eq.(8), we use Proximal Policy Optimization (PPO) [25] with the objective function below, to update the policy:

$$\mathbb{E}_t\Big[\min\Big(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}A_t, \text{clip}\Big(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}, 1-\epsilon, 1+\epsilon\Big)A_t\Big)\Big] \tag{10}$$

where $\epsilon$ is the clipping parameter, which represents the maximum percentage of change that can be updated at a time.

The training procedure involves two components: the discriminator and the actor-critic network. The training algorithm is illustrated in Algorithm 1. Specifically, for the discriminator, We use Adma as the optimizer to find the gradient for Eq.(5) for weight $w$:

$$\mathbb{E}_\pi[\nabla_w \log(D_w(s, a))] + \mathbb{E}_{\pi_E}[\nabla_w \log(1 - D_w(s, a))] \tag{11}$$

## 4 EXPERIMENTS

Experiments are conducted on several different simulation environments includes Virtual TaoBbao [27], RecoGym [22] and RecSim [13]. To best our knowledge, there are only three gym-based platforms available for online recommendation. Different from the

offline experiments which the environment is built upon an existing datasets, those platforms will generate users-items interaction dynamically.

## 4.1 Simulation Platforms

*4.1.1 VirtualTB.* VirtualTB is a dynamic environment to test the feasibility of the recommendation agent. It enables a customized agent to interact with it and achieve the corresponding rewards. On VirtualTB, each customer has 11 static attributes as the demographic information and are encoded into an 88-dimensional space with binary values. The customers have multiple dynamic interests that are encoded into a 3-dimensional space and may change over the interaction process. Each item has several attributes, e.g., price and sales volume, and are encoded into a 27-dimensional space. The environment will random generate those demographic information and dynamic interests at the initialization stage, and those dynamic interests will be update episodically based on the items be recommended.

*4.1.2 RecoGym.* RecoGym is a simulation environment for recommender system [22]. Different from VirtualTB, RecoGym does not involve user's long-term interest. Moreover, RecoGym only contains 100 users and 10 items implies to a smaller state and action space. RecoGym is focusing on product recommendation and online advertising based on data comes from e-commerce which provides both organic and the bandit user-system interaction.

*4.1.3 RecSim.* RecSim is another open-sourced simulation environment for recommender system [13]. RecSim is a configurable platforms allows customized environments to fulfil unique requirements of particular aspects of user behavior and item information in sequential interactive recommendation problems. There two built-in tasks which are long-term stratification and user interest evolution. In this paper, we do no use any customized environment. those two built-in tasks are mainly used in our experiment.

## 4.2 Baseline methods

The proposed method is focusing on online recommendation and reinforcement learning. Hence, we select the following existing methods as baselines.

- IRecGAN [2]: An online recommendation method that employs reinforcement learning and GAN.
- PGCR [20]: A policy Gradient based method for contextual recommendation.
- GAUM [7]: A deep Q-learning based method that employs GAN and cascade Q-learning for recommendation.
- KGRL [6]: Actor-Critic based method for interactive recommendation (i.e., a variant of online recommendation).

Note that GAUM and PGCR are not designed for online recommendation, and KGRL requires knowledge graph as side information, which is unavailable to the gym environment. Hence, we only keep the network structure and put those network into the VirtualTB platform for testing.

## 4.3 Evaluation Metric and Experimental Setup

The experiments are conducted in the OpenAI gym environment where the reward can be readily obtained for each episode. Since each episode may have differed numbers of steps, it leads to the difficulty in determining when users will end the session. For this reason, we choose click-through-rate to represent the performance which is defined as:

$$CTR = \frac{r_{episode}}{10 * N_{step}}$$

where 10 means that the user is interested in all the ten items that are recommended in a single page, $r_{episode}$ is the reward received per episode and $N_{step}$ is the number of steps included in one episode.

For RecSim and RecoGym, the quality score is used to determine the performance. Quality score is pre-defined on the environment which is similar as the reward.

**Algorithm 1:** Training algorithm for our model

**input :** Expert Policy $\pi_E$, current state $s$
Sampling expert trajectories $(s, a_E) \sim \pi_E$;
Initialize discriminator parameter $w_0$ ;
Initialize policy parameter $\theta_0$ ;
Initialize clipping parameter $\epsilon$;
**for** $i = 0, 1, \cdots$ **do**
  Sampling trajectories $(s, a) \sim \pi_{\theta_i}$;
  Update the parameter $w_i$ by gradient on Eq.(11) ;
  **for** $k = 0, 1, \cdots$ **do**
    Get the trajectories $(s, a)$ on policy $\pi_\theta = \pi(\theta_k)$ ;
    Estimate advantage $A_t$ using Eq.(9);
    Compute the Policy Update

$$\theta_{k+1} = \arg\max_{\theta} \text{Eq.(10)}$$

    Optimize these $K$ step by Adma
  **end**
  $\theta_i \leftarrow \theta_K$;
**end**

The model is implemented by using PyTorch, and the experiments are carried out on a server which consists of two 12-core/24-thread Intel (R) Xeon (R) CPU E5-2697 v2 CPUs, 6 NVIDIA TITAN X Pascal GPUs, 2 NVIDIA TITAN RTX, with a total 768 GiB memory.

## 4.4 Expert Policy Acquisition

In this part, we introduce the strategy on acquiring the expert policy for VirtualTB. There is no official expert policy in VirtualTB. Obviously, it is unrealistic to manually create the expert policy from this virtual environment where the source data is not available. Hence, we follow a similar strategy as in [9] to generate a set of expert policy from a pre-trained expert policy network. We design an actor-critic network $\mathcal{M}$ with the same actor and critic network structure as our model, but without advantage. The critic network from $\mathcal{M}$ is used to calculate the $Q$-value by adopting deep Q-learning. We adopts the Deep Deterministic Policy Gradients (DDPG) [18] to train $\mathcal{M}$. The detailed training procedure can be found on Algorithm 2.

For RecSim and RecoGym, the expert policy are acquired via a similar way. Different from VirtualTB, RecSim and RecoGym

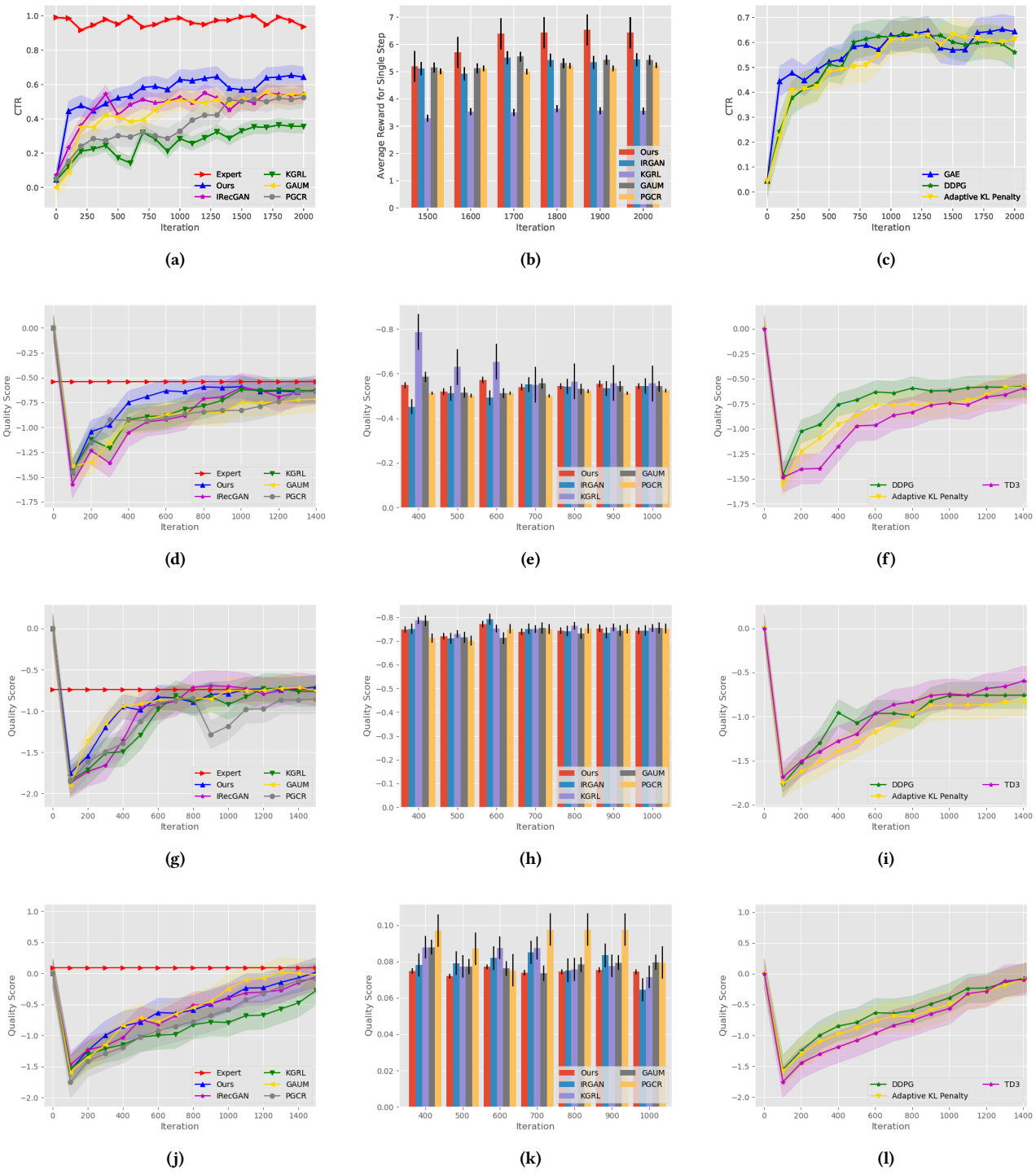


**Figure 3: Experimental results where (a) is CTR with 95% confidence interval, (b) is the average reward received each step after 1500 iterations with 95% confidence interval and (c) comes from the ablation study of different algorithm in VirtualTB. For comparison, we also provide the performance from RecSim - Interest Evolution, RecSim - Long-term Engagement and RecoGym with the same order.**

**Table 1: CTR for Different Parameter Settings for GAE and PPO with 95% Confidence Interval in VirtualTB, RecSim - Interest Evolution, RecSim - Long-term Engagement and RecoGym.**

| | | GAE: $\lambda_g$ | | | | | |
|---|---|---|---|---|---|---|---|
| | | 0.94 | 0.95 | 0.96 | 0.97 | 0.98 | 0.99 |
| PPO: $\epsilon$ | 0.05 | 0.630 ± 0.063 | 0.632 ± 0.064 | 0.633 ± 0.062 | 0.630 ± 0.059 | 0.626 ± 0.060 | 0.629 ± 0.059 |
| | 0.10 | 0.632 ± 0.062 | 0.635 ± 0.060 | 0.636 ± 0.061 | 0.636 ± 0.058 | 0.634 ± 0.061 | 0.633 ± 0.060 |
| | 0.15 | 0.633 ± 0.060 | 0.635 ± 0.061 | 0.639 ± 0.061 | 0.640 ± 0.057 | 0.639 ± 0.059 | 0.638 ± 0.061 |
| | 0.20 | 0.634 ± 0.060 | 0.636 ± 0.060 | 0.641 ± 0.063 | **0.643 ± 0.061** | 0.643 ± 0.063 | 0.641 ± 0.058 |
| | 0.25 | 0.631 ± 0.061 | 0.635 ± 0.059 | 0.636 ± 0.060 | 0.637 ± 0.060 | 0.636 ± 0.061 | 0.634 ± 0.059 |
| | 0.30 | 0.630 ± 0.059 | 0.631 ± 0.061 | 0.632 ± 0.060 | 0.630 ± 0.059 | 0.630 ± 0.058 | 0.629 ± 0.050 |

| | | GAE: $\lambda_g$ | | | | | |
|---|---|---|---|---|---|---|---|
| | | 0.94 | 0.95 | 0.96 | 0.97 | 0.98 | 0.99 |
| PPO: $\epsilon$ | 0.05 | -0.550 ± 0.023 | -0.55 ± 0.011 | -0.533 ± 0.009 | -0.560 ± 0.021 | -0.529 ± 0.032 | -0.531 ± 0.034 |
| | 0.10 | -0.563 ± 0.031 | -0.564 ± 0.020 | -0.536 ± 0.022 | -0.564 ± 0.034 | -0.534 ± 0.036 | -0.533 ± 0.031 |
| | 0.15 | -0.533 ± 0.030 | -0.535 ± 0.021 | -0.532 ± 0.021 | -0.540 ± 0.027 | -0.539 ± 0.029 | -0.532 ± 0.031 |
| | 0.20 | -0.530 ± 0.019 | -0.532 ± 0.030 | -0.531 ± 0.019 | -0.529 ± 0.021 | -0.530 ± 0.033 | -0.540 ± 0.018 |
| | 0.25 | -0.530 ± 0.020 | -0.532 ± 0.029 | -0.532 ± 0.020 | -0.537 ± 0.018 | -0.536 ± 0.011 | -0.541 ± 0.029 |
| | 0.30 | -0.532 ± 0.019 | -0.532 ± 0.021 | -0.532 ± 0.019 | -0.530 ± 0.019 | **-0.529 ± 0.018** | -0.526 ± 0.019 |

| | | GAE: $\lambda_g$ | | | | | |
|---|---|---|---|---|---|---|---|
| | | 0.94 | 0.95 | 0.96 | 0.97 | 0.98 | 0.99 |
| PPO: $\epsilon$ | 0.05 | -0.730 ± 0.023 | -0.742 ± 0.024 | -0.733 ± 0.020 | -0.750 ± 0.019 | -0.728 ± 0.019 | -0.729 ± 0.019 |
| | 0.10 | -0.732 ± 0.012 | -0.738 ± 0.020 | -0.735 ± 0.021 | -0.746 ± 0.018 | -0.733 ± 0.021 | -0.734 ± 0.020 |
| | 0.15 | -0.733 ± 0.020 | -0.737 ± 0.019 | -0.734 ± 0.021 | -0.740 ± 0.017 | -0.735 ± 0.019 | -0.738 ± 0.021 |
| | 0.20 | -0.734 ± 0.020 | -0.736 ± 0.020 | -0.741 ± 0.023 | -0.741 ± 0.021 | -0.743 ± 0.023 | -0.741 ± 0.018 |
| | 0.25 | -0.731 ± 0.021 | -0.736 ± 0.019 | -0.736 ± 0.020 | -0.737 ± 0.020 | -0.736 ± 0.021 | -0.734 ± 0.019 |
| | 0.30 | **-0.730 ± 0.019** | -0.734 ± 0.021 | -0.732 ± 0.018 | -0.734 ± 0.019 | -0.731 ± 0.018 | -0.729 ± 0.020 |

| | | GAE: $\lambda_g$ | | | | | |
|---|---|---|---|---|---|---|---|
| | | 0.94 | 0.95 | 0.96 | 0.97 | 0.98 | 0.99 |
| PPO: $\epsilon$ | 0.05 | 0.0532 ± 0.008 | 0.0531 ± 0.007 | 0.0533 ± 0.008 | 0.0531 ± 0.009 | 0.0533 ± 0.008 | 0.0529 ± 0.009 |
| | 0.10 | 0.0532 ± 0.006 | 0.0535 ± 0.008 | 0.0536 ± 0.006 | 0.0536 ± 0.007 | 0.0534 ± 0.006 | 0.0533 ± 0.006 |
| | 0.15 | 0.0533 ± 0.009 | 0.0535 ± 0.011 | 0.0539 ± 0.061 | 0.0540 ± 0.007 | 0.0539 ± 0.009 | 0.0538 ± 0.008 |
| | 0.20 | 0.0534 ± 0.006 | 0.0536 ± 0.012 | 0.0541 ± 0.005 | **0.0543 ± 0.004** | 0.0543 ± 0.007 | 0.0541 ± 0.008 |
| | 0.25 | 0.0531 ± 0.007 | 0.0535 ± 0.009 | 0.0536 ± 0.005 | 0.0537 ± 0.006 | 0.0536 ± 0.005 | 0.0534 ± 0.009 |
| | 0.30 | 0.0530 ± 0.009 | 0.0531 ± 0.005 | 0.0532 ± 0.006 | 0.0530 ± 0.005 | 0.0530 ± 0.008 | 0.0529 ± 0.050 |

are relatively small which do not requires extra step to process. DDPG is directly applied to learn the policy and expert policies are collected when it converged.

## 4.5 Hyperparameters Setting

For the policy network $\mathcal{M}$, we set the DDPG parameters as: $\gamma = 0.95, \tau = 0.001$, size of hidden layer is 128, the size of reply buffer is $1,000$ and the number of episode is set to $200,000$. For Ornstein-Uhlenbeck Noise, scale is 0.1, $\mu = 0, \theta = 0.15, \sigma = 0.2$. For our approach, number of episode is set to $100,000$, hidden size of the advantage actor-critic network is 256, hidden size for discriminator is 128, learning rate is 0.003, factor $\lambda$ is $10^{-3}$, mini batch size is 5 and the epoch of PPO is 4. For the generalized advantage estimation, we set the discount factor $\gamma$ to 0.995, $\lambda_g = 0.97$ and $\epsilon = 0.2$ for VirtualTB, $\lambda_g = 0.98$ and $\epsilon = 0.3$ for RecSim - Interest Evolution, $\lambda_g = 0.94$ and $\epsilon = 0.3$ for RecSim - Long-term Engagement, $\lambda_g = 0.97$ and $\epsilon = 0.2$ for RecoGym. For fair comparison, all those baseline methods are training under the same condition. For easy recognition, we set one iteration as 100 episodes in VirtualTB and one iteration as 1 episode in RecSim and RecoGym.

## 4.6 Results

Fully results are reported in Fig 3. Our approach generally outperforms all four existing methods. In RecSim and RecoGym, our method achieves the similar results as the baselines as those two environment are quite small. learning from expert can not significantly improve the performance. Specifically, our method gets the best result among all the baseline methods after 5500 iterations in VirtualTB. It demonstrates the feasibility of the proposed approach. A possible reason for the poor performance of KGRL is that KGRL maintains a local knowledge graph inside the model and actively interacts with the environment. KGRL performs poorer than other baseline methods as the experiments are conducted on an online platform, which does not provide the side information for KGRL to generate its knowledge graph.

*4.6.1 Impact of Key Parameters.* We are interested in how the control parameter $\lambda$ on GAE and the clipping parameter $\epsilon$ on PPO

**Algorithm 2:** DDPG algorithm for expertise

Initialize actor network $\mu$ with parameter $\theta_\mu$ and critic network $Q$ with parameter $\theta_Q$ randomly;
Initialize target network $\mu'$ and $Q'$ with weight $\theta_{\mu'} \leftarrow \theta_\mu$, $\theta_{Q'} \leftarrow \theta_Q$ ;
Initialize Replay Buffer $\mathcal{B}$ ;
**for** $i = 0$ *to* $n$ **do**
  Receive the initial state $S_i$ ;
  **for** $t = 1$ *to* $T$ **do**
    Infer a action $a_t$ according to the $\mu(\cdot)$ ;
    Execute the action $a_t$ to receive a reward $r_t$ and observe a new state $S_{t+1}$;
    $\mathcal{B}$.store$(S_t, a_t, r_t, S_{t+1})$ ;
    Sample a random mini-batch of $\mathcal{N}$ transitions $(S_k, a_k, r_k, S_{k+1})$ from $\mathcal{B}$ ;
    Set $y_i = r_t + \gamma Q'(S_{t+1}, \mu'(S_{t+1}|Q_{\mu'})|\theta_{Q'})$ ;
    Update Critic by minimise the loss:
    $\frac{1}{N}\sum_{j=1}^{N}(y_i - Q(S_t, a_t|Q_\theta))^2$ ;
    Update the Actor policy by policy gradient:
    $\frac{1}{N}\sum_{j=1}^{N}\nabla_a Q(S_t, a)|_{a=\mu(S_t)}\nabla_{\theta_\mu}\mu(S_t|\theta_\mu)$ ;
    Update target network:
    $\theta_{\mu'} \leftarrow \tau\theta_\mu + (1-\tau)\theta_{\mu'}$;
    $\theta_{Q'} \leftarrow \tau\theta_Q + (1-\tau)\theta_{Q'}$;
  **end**
**end**

affect the performance. These two key parameters significantly affect the generalized advantage estimation and proximal policy optimization process. $\lambda$ is used to make a compromise between bias and variance and is normally selected from $[0, 95, 1)$ with a step size of 0.01. The clipping parameter $\epsilon$ is used to determine the number of percentage that needs to be clipped, normally smaller than 0.3 to control the optimization step size. For fairness, we report CTR at iteration 2000. Observe that when $\lambda = 0.97$ and $\epsilon = 0.2$, the model achieves the best result which is $0.643 \pm 0.061$. These two values are also used as our default setting. The results can be found on Table 1.

*4.6.2 Ablation Study.* In this part, we investigate the effect of the GAE. We use two different optimization strategies to optimize the proposed model which are DDPG and Adaptive KL Penalty Coefficient. The Adaptive KL Penalty Coefficient is the simplified version of the PPO which can be defined as:

$$L(\theta) = \mathbb{E}_t\Big[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}A_t - \beta\mathrm{KL}[\pi_{\theta_{old}}(\cdot|s_t), \pi_\theta(\cdot|s_t)]\Big]$$

The update rule for $\beta$ is:

$$\begin{cases}\beta \leftarrow \beta/b & d < d_{target} * a\\ \beta \leftarrow \beta * b & d >= d_{target} * a\end{cases}$$
$$\text{where } d = \mathbb{E}_t[\mathrm{KL}[\pi_{\theta_{old}}(\cdot|s_t), \pi_\theta(\cdot|s_t)]]$$

The parameter $a = 1.5$, $b = 2$ and $d_{target} = 0.01$ are determined empirically. The result of the ablation study can be found on Fig 3(c),(f),(i) and (l) for those four different tasks in three environments. Moreover, we also interest in the parameter impact of the $a$, $b$ and $d_{target}$. The results in VirtualTB can be found on Table 2.

**Table 2: Result when $a, b, d_{target}$ vary in VirtualTB**

| 0.001 | | a | | |
|---|---|---|---|---|
| | | 1 | 2 | 3 |
| b | 0.5 | 0.600 ± 0.041 | 0.600 ± 0.040 | 0.599 ± 0.040 |
| | 1.0 | 0.599 ± 0.041 | 0.601 ± 0.041 | 0.601 ± 0.042 |
| | 1.5 | 0.599 ± 0.040 | 0.601 ± 0.041 | 0.599 ± 0.041 |
| 0.003 | | a | | |
| | | 1 | 2 | 3 |
| b | 0.5 | 0.610 ± 0.044 | 0.611 ± 0.044 | 0.610 ± 0.050 |
| | 1.0 | 0.611 ± 0.042 | 0.612 ± 0.050 | 0.611 ± 0.049 |
| | 1.5 | 0.611 ± 0.040 | 0.612 ± 0.045 | 0.610 ± 0.046 |
| 0.01 | | a | | |
| | | 1 | 2 | 3 |
| b | 0.5 | 0.619 ± 0.041 | 0.618 ± 0.044 | 0.620 ± 0.040 |
| | 1.0 | 0.619 ± 0.042 | 0.620 ± 0.040 | 0.621 ± 0.042 |
| | 1.5 | 0.620 ± 0.043 | 0.621 ± 0.042 | 0.621 ± 0.043 |
| 0.03 | | a | | |
| | | 1 | 2 | 3 |
| b | 0.5 | 0.631 ± 0.044 | 0.629 ± 0.044 | 0.630 ± 0.040 |
| | 1.0 | 0.630 ± 0.042 | 0.631 ± 0.040 | 0.631 ± 0.045 |
| | 1.5 | 0.631 ± 0.041 | **0.632 ± 0.041** | 0.630 ± 0.042 |

Based on the conclusion from [25], the value of $a, b$ would not significantly affect the performance of the PPO process. Hence, we final decide to use $a = 2, b = 1.5, d_{target} = 0.03$ as the final parameter. Moreover, based on that, we only report the result on VirtualTB as it does not significantly affect the performane.

## 4.7 Discussion

This study provides a new approach for reinforcement learning based online recommendation, without needing to define the reward function. In this way, our work is feasible to be applied to various real-world recommendation scenarios where the reward function is highly domain-dependent or hard to be manually defined. The proposed method offers a fundamental support for inverse reinforcement learning based recommendation system. The proposed method can extract an adaptive reward function and automatically find the optimal strategies for generating recommendations that best fit users' interest based on a few user behaviors. Our empirical evaluation testifies its competitive performance against several state-of-the-art reinforcement learned based methods. Our model can potentially accelerate the application of reinforcement learning in complex environments.

# 5 RELATED WORK

In this section, we will briefly review the related works about the deep reinforcement learning in recommender systems which includes partial observable MDP-based methods and MDP-based methods.

Firstly, partial observable MDP-based (POMDP) relevant methods will be discussed. POMDP based methods can be further divided into three categories: value function estimation, policy optimization, and stochastic sampling. Hauskrecht [11] introduce the value function estimation to estimate the upper bound of the value by using an incremental value iteration method. Poupart and Boutilier [21] further improve the upper bound by using bounded policy iteration and value-directed compression to increase the scalability of the policy optimization. Kearns et al. [14] also deal with the scalability of the policy optimization in large-scaled POMDP. In a large-scaled POMDP, the reward is sparse and the traditional sampling strategy can not achieve a good performance. Due to the high computational and representational complexity of the POMDP based methods, MDP-based methods are relatively more popular in academia. However, there still exist a work [26] uses POMDP to construct the recommender system. The main idea is that, in recommender system task, there existing lots of co-founder factors will affect user's decision. Those co-founders are not visible which implies to POMDP would be a best choice.

MDP-based DRL methods cover three approaches to recommendation: deep Q-learning based, policy gradient based, and Actor-Critic based methods. [31] adopts deep Q-learning in news recommendation and use users' historical records as states. [32] improves the structure of Deep Q-learning to achieve robust results. [20] applies policy gradient to learn optimal recommendation strategies. [28] introduces the knowledge graph into the policy gradient for sequential recommendation. However, Q-learning may get stuck because of the max operation, and policy gradient requires large-scale data to boost convergence. It only updates once per episode. Hence, Actor-Critic uses Q-value to conduct the policy gradient per step instead of per episode. [30] adopts actor-critic methods in list-wise recommendation in a simulated environment. [6, 29] utilize the knowledge graph as the side information embedded into the state-action space to improve the model's capability on an actor-critic network. Besides, [19] produces recommendations via learning a state-action embedding within the DRL framework.

Furthermore, [7] integrates the generative adversarial network with reinforcement learning structure to generate user's attribute so that more side information would be available to boost the performance of reinforcement learning based recommendation. [26] proposes a multi-agent based DRL method for environment reconstruction by taking environmental co-founders into account.

## 6 CONCLUSION AND FUTURE WORK

We propose a new approach InvRec for online recommendation. Our proposed approach is designed to overcome the challenges posed by an inaccurate reward function. Our proposed model is built on advantage actor-critic network with generate adversarial imitation learning. We evaluate our method on the online platform VirtualTB in comparison with three categories of state-of-the-art methods: Deep Q-Learning based, policy gradient based, and actor-critic network based methods. The results demonstrate the feasibility and superior performance of our approach.

This study is an preliminary attempt on applying deep inverse reinforcement learning to online recommendation, and a few challenges remain unaddressed, e.g., the sample inefficiency problem for the imitation learning [17], i.e., low sample inefficiency will lead to longer training time and require a larger dataset; random sampling also would affect performance. The possible solutions include using off-policy methods instead of on-policy methods and finding an optimal sampling strategy to enable the agent to obtain the same expert trajectories to handle the states that used to occur previously.

## REFERENCES


[1] Pieter Abbeel and Andrew Y Ng. 2004. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*. 1.

[2] Xueying Bai, Jian Guan, and Hongning Wang. 2019. A Model-Based Reinforcement Learning with Adversarial Training for Online Recommendation. In *Advances in Neural Information Processing Systems*. 10735–10746.

[3] Michael Bloem and Nicholas Bambos. 2014. Infinite time horizon maximum causal entropy inverse reinforcement learning. In *53rd IEEE Conference on Decision and Control*. IEEE, 4911–4916.

[4] Haokun Chen, Xinyi Dai, Han Cai, Weinan Zhang, Xuejian Wang, Ruiming Tang, Yuzhou Zhang, and Yong Yu. 2019. Large-scale interactive recommendation with tree-structured policy gradient. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 3312–3320.

[5] Shi-Yong Chen, Yang Yu, Qing Da, Jun Tan, Hai-Kuan Huang, and Hai-Hong Tang. 2018. Stabilizing reinforcement learning in dynamic environment with application to online recommendation. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1187–1196.

[6] Xiaocong Chen, Chaoran Huang, Lina Yao, Xianzhi Wang, Wei Liu, and Wenjie Zhang. 2020. Knowledge-guided Deep Reinforcement Learning for Interactive Recommendation. *arXiv preprint arXiv:2004.08068* (2020).

[7] Xinshi Chen, Shuang Li, Hui Li, Shaohua Jiang, Yuan Qi, and Le Song. 2019. Generative adversarial user model for reinforcement learning based recommendation system. In *International Conference on Machine Learning*. 1052–1061.

[8] Chelsea Finn, Sergey Levine, and Pieter Abbeel. 2016. Guided cost learning: Deep inverse optimal control via policy optimization. In *International conference on machine learning*. 49–58.

[9] Yang Gao, Huazhe(Harry) Xu, Ji Lin, Fisher Yu, Sergey Levine, and Trevor Darrell. 2018. Reinforcement Learning from Imperfect Demonstrations. (2018). https://openreview.net/forum?id=BJJ9bz-0-

[10] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In *Advances in neural information processing systems*. 2672–2680.

[11] Milos Hauskrecht. 1997. Incremental methods for computing bounds in partially observable Markov decision processes. In *AAAI/IAAI*. Citeseer, 734–739.

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

[13] Eugene Ie, Chih-wei Hsu, Martin Mladenov, Vihan Jain, Sanmit Narvekar, Jing Wang, Rui Wu, and Craig Boutilier. 2019. Recsim: A configurable simulation platform for recommender systems. *arXiv preprint arXiv:1909.04847* (2019).

[14] Michael Kearns, Yishay Mansour, and Andrew Y Ng. 2002. A sparse sampling algorithm for near-optimal planning in large Markov decision processes. *Machine learning* 49, 2-3 (2002), 193–208.

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

[16] Vijay R Konda and John N Tsitsiklis. 2000. Actor-critic algorithms. In *Advances in neural information processing systems*. 1008–1014.

[17] Ilya Kostrikov, Kumar Krishna Agrawal, Debidatta Dwibedi, Sergey Levine, and Jonathan Tompson. 2019. Discriminator-Actor-Critic: Addressing Sample Inefficiency and Reward Bias in Adversarial Imitation Learning. In *International Conference on Learning Representations*. https://openreview.net/forum?id=Hk4fpoA5Km

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

[19] Feng Liu, Huifeng Guo, Xutao Li, Ruiming Tang, Yunming Ye, and Xiuqiang He. 2020. End-to-End Deep Reinforcement Learning based Recommendation with Supervised Embedding. In *Proceedings of the 13th International Conference on Web Search and Data Mining*. 384–392.

[20] Feiyang Pan, Qingpeng Cai, Pingzhong Tang, Fuzhen Zhuang, and Qing He. 2019. Policy gradients for contextual recommendations. In *The World Wide Web Conference*. 1421–1431.

[21] Pascal Poupart and Craig Boutilier. 2005. VDCBPI: an approximate scalable algorithm for large POMDPs. In *Advances in Neural Information Processing Systems*. 1081–1088.

[22] David Rohde, Stephen Bonner, Travis Dunlop, Flavian Vasile, and Alexandros Karatzoglou. 2018. Recogym: A reinforcement learning environment for the

problem of product recommendation in online advertising. *arXiv preprint arXiv:1808.00720* (2018).

[23] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. 2015. Trust region policy optimization. In *International conference on machine learning*. 1889–1897.

[24] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. 2015. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438* (2015).

[25] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).

[26] Wenjie Shang, Yang Yu, Qingyang Li, Zhiwei Qin, Yiping Meng, and Jieping Ye. 2019. Environment Reconstruction with Hidden Confounders for Reinforcement Learning based Recommendation. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 566–576.

[27] Jing-Cheng Shi, Yang Yu, Qing Da, Shi-Yong Chen, and An-Xiang Zeng. 2019. Virtual-taobao: Virtualizing real-world online retail environment for reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 4902–4909.

[28] Pengfei Wang, Yu Fan, Long Xia, Wayne Xin Zhao, Shaozhang Niu, and Jimmy Huang. 2020. KERL: A Knowledge-Guided Reinforcement Learning Model for Sequential Recommendation. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 209–218.

[29] Kangzhi Zhao, Xiting Wang, Yuren Zhang, Li Zhao, Zheng Liu, Chunxiao Xing, and Xing Xie. 2020. Leveraging Demonstrations for Reinforcement Recommendation Reasoning over Knowledge Graphs. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 239–248.

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

[31] 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 *Proceedings of the 2018 World Wide Web Conference*. 167–176.

[32] Lixin Zou, Long Xia, Pan Du, Zhuo Zhang, Ting Bai, Weidong Liu, Jian-Yun Nie, and Dawei Yin. 2020. Pseudo Dyna-Q: A Reinforcement Learning Framework for Interactive Recommendation. In *Proceedings of the 13th International Conference on Web Search and Data Mining*. 816–824.