NEURAL MODEL-BASED REINFORCEMENT LEARNING FOR RECOMMENDATION

Anonymous authors
Paper under double-blind review

ABSTRACT

There is a great interest in applying reinforcement learning (RL) to recommendation systems. However, in this setting, an online user is the environment; neither the reward function nor the environment dynamics is clearly defined, making the application of RL challenging. In this paper, we propose a novel model-based reinforcement learning framework for recommendation systems, where we developed a generative adversarial network to imitate user behavior dynamics and learn her reward function. Using this user model as the simulation environment, we develop a novel DQN algorithm to obtain a combinatorial recommendation policy which can handle a large number of candidate items efficiently. In our experiments with real data, we show this generative adversarial user model can better explain user behavior than alternatives, and the RL policy based on this model can lead to better long turn reward for the user and higher click rate for the system.

1 INTRODUCTION

Recommendation systems have become a crucial part of almost all online service platforms. A typical interaction between the system and its users is — users are recommended a page of items and provide feedback; and then the system recommends a new page of items. A common way of building recommendation systems is to estimate a model which minimizes the discrepancy between the model prediction and the *immediate* user response according to some loss function. In other words, these models do not explicitly take into account the long term user interest. However, user's interest can evolve over time based on what she observes, and the recommender's action may significantly influence such evolution. In some sense the recommender is guiding users' interest by displaying particular items and hidding the rest. Thus, a recommendation strategy which takes user's long term interest into account is more favorable.

Reinforcement learning (RL) is a learning paradigm where a policy will be obtained to take actions in an environment so as to maximize the expected long term reward. In the recommendation system setting, the environment will correspond to the logged online user, the reward function will be the user's interest, and the policy will correspond to the recommendation policy. Although RL framework has been successfully applied to many game settings, such as Atari games (Mnih et al., 2015) and GO (Silver et al., 2016), it met a few challenges in the recommendation system setting.

First, a user's interest (reward function) driving her behavior is typically unknown, yet it is critically important for the use of reinforcement learning algorithms. Essentially, without the reward function, neither model-based nor model-free reinforcement learning algorithms can be used. Thus, in existing use of RL for recommendation systems, the reward functions are typically manually designed which may or may not reflect user's true interests (Zhao et al., 2018a; Zheng et al., 2018).

Second, model-free reinforcement learning typically requires lots of interactions with the environment in order to learn a good policy. This is impractical in the recommendation system setting. A logged online user will quickly abandon the service if the recommendation looks random and do not meet user's interests. Thus, to avoid the large sample complexity of the model-free approach, a model-based RL approach is more preferable. In a related but a different setting where one wants to train a robot policy, recent work showed that model-based reinforcement learning is much more sample efficient (Nagabandi et al., 2017; Deisenroth et al., 2015; Clavera et al., 2018). However, these approaches can not be used in the recommendation setting, since a user behavior model typically consists of sequences of discrete choices under complex session context.

To address the above challenges, we propose a novel model-based reinforcement learning framework for recommendation systems, where we first recover a user behavior model from historical data, and then optimize a policy in response to this model. Our technical innovations are:

1. We develop a generative adversarial network formulation to imitate user behavior dynamics and recover her reward function. These two components are estimated simultaneously via a joint mini-max optimization algorithm;
2. Using this model as the simulation environment, we develop a cascaded DQN algorithm to obtain a recommendation policy. The cascaded design of action-value function allows us to handle a large number of items with time complexity only linear in the number.

In our experiments with real data, we showed that this generative adversarial model has a better fit to user behavior in terms of held-out likelihood and click prediction. Since our model can take historical items clicked by the user into account, these results show the benefits of explicitly modeling the influence of history on user's interests, and imply the potential usefulness of reinforcement learning. Based on the learned user model and reward, we show that the estimated recommendation policy leads to better cumulative long term reward for the user. Furthermore, in the case of model mismatch, our model-based policy can also quickly adapt to the new dynamics with much few number of user interactions compared to model-free approaches.

2 Related Work

Commonly used recommendation algorithms typically use simple user model. For instance, logistic regression based wide&deep networks (Cheng et al., 2016) basically assume an user choose each item independently; collaborative competitive filtering (Yang et al., 2011) takes into account the context of an user's choice, but assume each pageview is independent from each other. Session-based RNN (Hidasi et al., 2016) improves upon previous approaches by modeling user's history, but this model does not recover user's reward function and can not be used subsequently for reinforcement learning. Contextual bandit based approaches, such as LinUCB (Li et al., 2010), can deal with adversarial user behaviors, but the reward is used and updated in a Bayesian framework, not directly usable by a reinforcement learning framework.

Zhao et al. (2018b;a); Zheng et al. (2018) used model-free RL for recommender systems, which may require many user interactions and the reward function is manually designed. Model-based reinforcement learning has been commonly used in robotics applications, and resulted in much reduced (interaction) sample complexity to obtain good policy (Deisenroth et al., 2015; Nagabandi et al., 2017; Clavera et al., 2018). The advantage of model-based approach is that potentially large amount of off-policy data can be pooled and used to learn a good environment dynamics model, whereas model-free approach can only use expensive on-policy data for learning. However, previous model-based approach are typically designed based on physics or Gaussian processes, and not tailored for complex sequences of user behaviors.

3 Setting and Overview of the Approach

We assume a typical interaction setting between the recommendation system and its users be – **in each pageview, users are recommended a page of k items and provide feedback by clicking on only one of these items; and then the system recommends a new page of k items.**

Since reinforcement learning can explicitly take into account longer term rewards, it holds the promise to improve user's long term engagement with an online platform. In the reinforcement learning language, a recommendation system wants to learn a recommendation policy $\pi(\boldsymbol{s})$ such that the long term expected reward is maximized, i.e.

$$\pi^* = \arg\max_{\pi(\boldsymbol{s}^t, \mathcal{I}^t)} \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(\boldsymbol{s}^t, \mathcal{A}^t, a^t)\right], \text{ where } \boldsymbol{s}^0 \sim p^0,\ \mathcal{A}^t \sim \pi,\ \boldsymbol{s}^{t+1} \sim P(\cdot|\boldsymbol{s}^t, \mathcal{A}^t). \tag{1}$$

Below we will elaborate on several key aspects of the above RL framework as instantiated in recommendation system setting:

1. **Environment**: will correspond to a logged online user who can click on one of the k items displayed by the recommendation system in each pageview (or interaction).
2. **State** $\boldsymbol{s}^t \in \mathcal{S}$: will correspond to an ordered sequence of user's historical clicks.
3. **Action** $\mathcal{A}^t \in \binom{\mathcal{I}}{k}$: will correspond to a subset of k items chosen by the recommendation system from $\mathcal{I}$ to display to the user. The notation $\binom{\mathcal{I}}{k}$ means all subset of size k of $\mathcal{I}$.

4. **State Transition** $P(\cdot|\boldsymbol{s}^t, \mathcal{A}^t) : \mathcal{S} \times \binom{\mathcal{I}}{k} \mapsto \mathcal{P}(\mathcal{S})$: will correspond to a user behavior model which returns the transition probability for $\boldsymbol{s}^{t+1}$ given previous state $\boldsymbol{s}^t$ and displayed items $\mathcal{A}^t$. It is equivalent to the distribution $\phi(\boldsymbol{s}^t, \mathcal{A}^t)$ over user's action defined in our user model, which will be introduced in section 4.1.
5. **Reward Function** $r(\boldsymbol{s}^t, \mathcal{A}^t, a^t) : \mathcal{S} \times \binom{\mathcal{I}}{k} \times \mathcal{I} \mapsto \mathbb{R}$: will correspond to user's reward or satisfaction in state $\boldsymbol{s}^t$ and after making her choice $a^t \in \mathcal{A}^t$ from the set displayed by the recommendation system. Here we assume that the reward to the recommendation system is the same as user's interests. Thus, a recommendation algorithm which optimizes its long term reward is designed to satisfy the user in a long run. One can also include the online platform's benefit to the reward, but in this paper we will focus on user's satisfaction.
6. **Policy** $\pi(\boldsymbol{s}^t, \mathcal{I}^t) : \mathcal{S} \times 2^{\mathcal{I}} \mapsto \mathcal{P}(\binom{\mathcal{I}}{k})$: will correspond to a recommendation algorithm which takes user's state $\boldsymbol{s}^t$ and returns the probability distribution of a subset of $\mathcal{I}^t$ to be displayed. Here we introduce the notation $\mathcal{I}^t \subset \mathcal{I}$ to represent available items at time t.

Remark. We note that in the above mapping, **Environment, State and State Transition** are associated with the user, the **Action and Policy** are associated with the recommendation system, and the **Reward Function** is associated with both the recommendation system and the user. Here we use the notation $r(\boldsymbol{s}^t, \mathcal{A}^t, a^t)$ to emphasize the dependency of the reward on the recommendation action, since user can only choose from the display set. However, the value of the reward is actually determined by user's state and the clicked item once the item is in the display set. Thus, in section 4.1 when we discuss on the user model, we simply denote $r(\boldsymbol{s}^t, a^t) = r(\boldsymbol{s}^t, \mathcal{A}^t, a^t)$ and assume $a^t \in \mathcal{A}^t$ is true. In fact, $r(\boldsymbol{s}^t, \mathcal{A}^t, a^t) = r(\boldsymbol{s}^t, a^t) \cdot \mathbf{1}(a^t \in \mathcal{A}^t)$.

When both the reward function and the state transition model are given, the optimal policy π^* in Eq. (1) can be estimated by repeated query to the model using algorithms such as Q-learning (Watkins, 1989). Thus, our first goal is to learn user's reward function. Furthermore, since model-based reinforcement learning is more sample efficient, we will also estimate a user behavior model associate with the reward function. Given both the reward function and state transition model, we can then use an RL algorithm to obtain a recommendation policy.

More specifically, our solution consists of two major innovations. The first component is a generative adversarial network for the user behavior model and reward function. We will design a position weighted sequence model to embed the user's historical sequence of clicks into the user state $\boldsymbol{s}^t$, and then use this state to define the reward function of the user and the probability of user clicking on a specific item in a pageview. Our formulation unifies the two estimation problems in a single mini-max optimization, and learn a consistent pair of user behavior model and reward function based on the sequence of user clicking history. The second component is a cascaded design of action-value function which allows us to learn a policy efficiently given a large action space faced by the recommendation system. In the following two sections, we will explain these two components of our framework (as shown in figure 1) in more details.

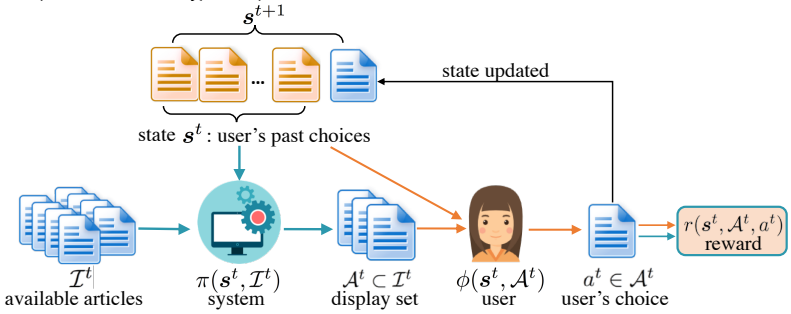

Figure 1: An illustration of the interaction between user and recommendation system. Green arrows represent the recommendation framework and orange arrows represent user's framework.

4 Generative Adversarial Network for User Behavior

In this section, we will propose a model for the sequence of user clicking behavior, discussion its parametrization and parameter estimation.

4.1 User Behavior As Reward Maximization

We assume that during each pageview, the user is presented with k items, out of which she clicks one. The sequence of items selected by the user before session t is an ordered list of observed features,

and we view it or embedding of it as the state of the user, i.e. $\boldsymbol{s}^t := h\left(\boldsymbol{F}_*^{1:t-1} = [\boldsymbol{f}_*^1, \cdots, \boldsymbol{f}_*^{t-1}]\right)$, where $\boldsymbol{f}_*^{\cdot} \in \mathbb{R}^d$ is the feature vector of the selected items, and $h(\cdot)$ is an embedding function. One can also view $\boldsymbol{F}_*^{1:t-1}$ as a matrix of historical features of clicked items, and similarly one can also define M step historical features of click items as $\boldsymbol{F}_*^{t-M:t-1} := [\boldsymbol{f}_*^{t-M}, \cdots, \boldsymbol{f}_*^{t-1}]$.

Suppose in pageview t, the user is presented with a set of k items $\mathcal{A}^t = \{a_1, \cdots, a_k\}$ and their associated features $\boldsymbol{F}^t = \{\boldsymbol{f}_1^t, \cdots, \boldsymbol{f}_k^t\}$ by the recommendation system, and a selection $a^t \in \mathcal{A}^t$ is taken by the user according to her preference function $a^t \sim \phi^*(\boldsymbol{s}^t, \mathcal{A}^t) \in \Delta^{k-1}$ where Δ^{k-1} is the probability simplex. More specifically, we assume that user clicks items to maximize its own interests. That is we assume that user's clicking probability is the results of the following optimization

$$\textbf{User Model:} \quad \phi^*(\boldsymbol{s}^t, \mathcal{A}^t) = \arg\max_{\phi \in \Delta^{k-1}} \mathbb{E}_\phi\left[r(\boldsymbol{s}^t, a^t)\right] - \frac{1}{\eta}\mathbb{E}_\phi[R(\phi)], \tag{2}$$

where $R(\phi)$ is a concave regularization function and η controls the strength of the regularization. After clicking an item a^t, the user state will be updated to $\boldsymbol{s}^{t+1} = h\left(\boldsymbol{F}_*^{1:t} = \{\boldsymbol{f}_*^1, \cdots, \boldsymbol{f}_*^{t-1}, \boldsymbol{f}_{a^t}^t\}\right)$.

Remark. We assume the case when the user does not click any item to be a special item and consider its reward as zero or a constant to be learned. Alternatively, it can also be included as an item of zero feature vector, so equation Eq. (2) can generate this case with simple modification. Besides, when no item is clicked, the state of user will not be updated (i.e., $\boldsymbol{s}^{t+1} = \boldsymbol{s}^t$).

Furthermore, if we allow $\phi \in \Delta^{k-1}$ to be arbitrary mappings (i.e., not limited in a specific parameter space), the optimization problem in Eq. (2) has a closed form solution, where user clicks on an item according to an exponential distribution in the follow lemma (See Appendix A for a proof).

Lemma 1. *Let the regularization be $R(\phi) = \log(\phi)$ and $\phi \in \Delta^{k-1}$ include arbitrary mappings. Then the optimal solution ϕ^* for the problem in Eq. (2) has a closed form*

$$\phi^*(\boldsymbol{s}^t, \mathcal{A}^t)_i = \frac{\exp(\eta r(\boldsymbol{s}^t, a_i))}{\sum_{a_j \in \mathcal{A}^t} \exp(\eta r(\boldsymbol{s}^t, a_j))}. \tag{3}$$

Furthermore, at each time t, the user's decision according to her optimal policy ϕ^* is equivalent to he following discrete choice model

$$a^t = \arg\max_{a \in \mathcal{A}^t} \eta\, r(\boldsymbol{s}^t, a) + \varepsilon^t, \tag{4}$$

where ε^t follows a Gumbel distribution.

This lemma makes it clear that the user essentially greedily picks an item according to its reward function (exploitation), and yet the Gumbel noise ε^t allows the user to deviate and explore other less rewarding items. Similar model has also appeared in the econometric choice model (Manski, 1975; McFadden, 1973), but previous econometric models did not take into account diverse features and user state evolution. Furthermore, to the best of our knowledge, such models have not been considered in model-based RL. Furthermore, other regularization $R(\phi)$ can also be used in our framework, which may induce different user behaviors. In these cases, the relations between ϕ^* and r are also different, and may not appear in closed form.

4.2 Model Parameterization

In the following, we will first define the state embedding $\boldsymbol{s}^t = h(\boldsymbol{F}_*^{1:t-1})$, and then define the reward function $r(\boldsymbol{s}^t, a^t)$ based on the state and the embedding of click a^t.

More specifically, we propose to use a simple and effective position weighting scheme for the state embedding. Let $\boldsymbol{W} \in \mathbb{R}^{m \times n}$ be a matrix where the number of rows m correspond to the number of history steps to take into account, and each of the n columns will correspond to one set of importance weights on positions. Then the embedding map h can be designed as

$$\boldsymbol{s}^t = h(\boldsymbol{F}_*^{1:t-1}) := vec\left[\,\sigma\left(\boldsymbol{F}_*^{t-m:t-1}\boldsymbol{W} + \boldsymbol{B}\right)\right] \quad \in \quad \mathbb{R}^{dn \times 1}, \tag{5}$$

where $\boldsymbol{B} \in \mathbb{R}^{d \times n}$ is a bias matrix, $\sigma(\cdot)$ is a nonlinear activation function such as the ReLU and ELU unit applied elementwisely to its arguments, and $vec[\cdot]$ turns the input matrix into a long vector by concatenating the matrix columns. Alternatively, one can also use an LSTM to capture the history. However, the advantage of our position weighting parametrization is that the history embedding is obtained by a shallow network which is more efficient for forward computation and gradient backpropagation than recurrent networks.

The user clicks a^t corresponds to an item with feature $\boldsymbol{f}_{a^t}^t$. Thus we will use $\boldsymbol{f}_{a^t}^t$ as the surrogate for user click and parametrize the reward function and user behavior model as

$$r(\boldsymbol{s}^t, a^t) := \boldsymbol{r}^\top \sigma\left(\boldsymbol{V}\begin{bmatrix}\boldsymbol{s}^t\\ \boldsymbol{f}_{a^t}^t\end{bmatrix} + \boldsymbol{b}\right), \quad \phi(\boldsymbol{s}, \mathcal{A}^t) \propto \exp\left(\boldsymbol{r}'^\top \sigma\left(\boldsymbol{V}'\begin{bmatrix}\boldsymbol{s}^t\\ \boldsymbol{f}_{a^t}^t\end{bmatrix} + \boldsymbol{b}'\right)\right) \tag{6}$$

where $\boldsymbol{V}, \boldsymbol{V}' \in \mathbb{R}^{\ell\times(dn+d)}$ are weight matrices, $\boldsymbol{b}, \boldsymbol{b}' \in \mathbb{R}^{1\times(dn+d)}$ are bias vectors and $\boldsymbol{r}, \boldsymbol{r}' \in \mathbb{R}^\ell$ are the final regression parameter. See Figure 2 for an illustration of the overall parametrization.

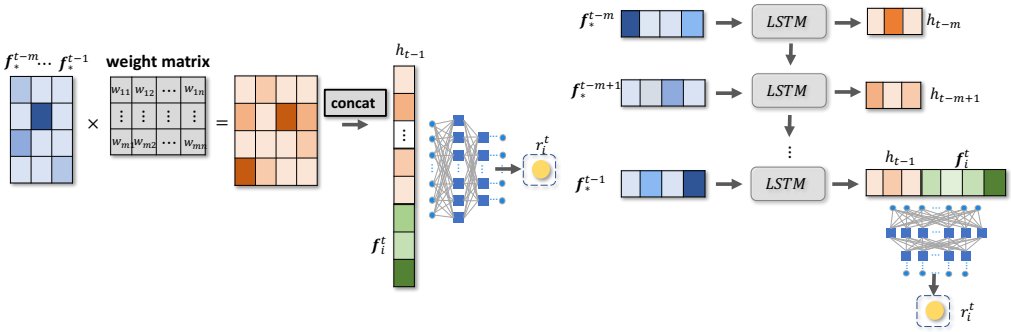

Figure 2: Architecture of our GAN model with reward function and user behavior model parameterized similarly by either position weight (PW) or LSTM.

4.3 Learning User Behavior Model

In practice, both the user reward function $r(\boldsymbol{s}^t, a^t)$ and the behavior model $\phi(\boldsymbol{s}^t, \mathcal{A}^t)$ are unknown and need to be estimated from data. Given a sequence of T observed user clicks, $\{a_{true}^1, a_{true}^2, \ldots, a_{true}^T\}$ and the corresponding item features $\{\boldsymbol{f}_*^1, \boldsymbol{f}_*^2, \ldots, \boldsymbol{f}_*^T\}$, we can estimate the user utility and behavior model by solving the following mini-max optimization

$$\min_{\theta\in\Theta}\left(\max_{\phi\in\Phi}\mathbb{E}_\phi\left[\sum_{t=1}^T r_\theta(\boldsymbol{s}_{true}^t, a^t) - \frac{1}{\eta}R(\phi)\right]\right) - \sum_{t=1}^T r_\theta(\boldsymbol{s}_{true}^t, a_{true}^t) \tag{7}$$

where θ denotes all parameters in the r function, $\Phi \subset \{\phi : \mathcal{S} \times \binom{\mathcal{I}}{k} \mapsto \Delta^{k-1}\}$ denotes a family of mappings, and we use $\boldsymbol{s}_{true}^t = \{\boldsymbol{f}_*^1, \ldots, \boldsymbol{f}_*^{t-1}\}$ to emphasize that this is observed in the data. The meaning of the optimization problem is that a good reward function should give the observed user behavior higher reward than the user behavior model, and thus the user model can approach the real user behavior by maximizing this reward function.

The mini-max optimization problem in Eq. (7) can be solved by iteratively updating the ϕ and r_θ, but the process may be unstable due to the non-convex problems. We will explain how we can make use of a special closed form to help initialize the mini-max optimization after the statement of the following lemma. See Appendix A for a proof.

Lemma 2. *Consider the case when regularization is defined as $R(\phi) = \log(\phi)$ and Φ includes all mappings from $\mathcal{S} \times \binom{\mathcal{I}}{k}$ to Δ^{k-1}. Then the optimization problem in Eq. (7) is equivalent to the following maximum likelihood estimation*

$$\max_{\theta\in\Theta}\prod_{t=1}^T \frac{\exp(\eta r_\theta(\boldsymbol{s}_{true}^t, a_{true}^t))}{\sum_{a^t\in\mathcal{A}^t}\exp(\eta r_\theta(\boldsymbol{s}_{true}^t, a^t))}. \tag{8}$$

This lemma enables us to learn a θ^* in a more stable and efficient way by solving a single maximization problem. We can simply apply standard algorithms such as SGD or ADAM to find an approximate solution $\hat{\theta}^*$. Then we can use this $\hat{\theta}^*$ as the initialization for the optimization problem Eq. (7) with other $R(\phi)$.

5 Cascading Q-networks for Recommendation Policy

Using the estimated user behavior model and the corresponding reward function as the simulation environment, we can then use reinforcement learning to obtain a recommendation policy. Note that the recommendation policy needs to deal with a *combinatorial action space* $\binom{\mathcal{I}}{k}$, where each action is a set of k items chosen from a larger set $\mathcal{I}$ of K candidates. Two challenges associated with this problem include the potentially high computational complexity of the combinatorial action space and the development of a framework for estimating the long-term reward (the Q function) from a

combination of items. Our contribution is designing a novel cascade of Q-networks consistent with each other for handling the combinatorial action space. Based on this cascade of Q-networks, we can also design an algorithm to estimate them from interaction with the environment.

5.1 Cascading Q-Networks

We assume that each time when a user visits the online platform, the recommendation system need to choose a subset $\mathcal{A}$ of k items, i.e., $\mathcal{A} \subset \mathcal{I}$. We will use the Q-learning framework where an optimal action-value function $Q^*(\boldsymbol{s}, \mathcal{A})$ will be learned and satisfies

$$Q^*(\boldsymbol{s}^t, \mathcal{A}^t) = \mathbb{E}\left[r(\boldsymbol{s}^t, \mathcal{A}^t, a^t) + \gamma \max_{\mathcal{A}' \subset \mathcal{I}} Q^*(\boldsymbol{s}^{t+1}, \mathcal{A}')\right], \ \ a^t \in \mathcal{A}^t. \tag{9}$$

Once the action-value function is learned, an optimal policy for recommendation can be obtained as

$$\pi^*(\boldsymbol{s}^t, \mathcal{I}^t) = \arg\max_{\mathcal{A}^t \subset \mathcal{I}^t} Q^*(\boldsymbol{s}^t, \mathcal{A}^t), \tag{10}$$

where $\mathcal{I}^t \subset \mathcal{I}$ is the set of items available at time t. Furthermore, we assume that once an item is clicked by an user, it becomes unavailable and will be excluded from being recommended again. The challenge is that the action space of the recommendation system contains $\binom{K}{k}$ many choices, which can be very large even for moderate K (e.g. 1000) and k (e.g. 5). Furthermore, an item put in different combinations can have different probability of being clicked, which is indicated by the user model and is in line with reality. For instance, interesting items may compete with each other for user attention. Thus, the policy in Eq. (10) will be very expensive to compute. To address this challenge, we will design not just one but a set of k related Q-functions which will be used in a cascading fashion for finding the maximum in Eq. (10).

Let an recommendation be $\mathcal{A} = \{a_1, a_2, \cdots, a_k\}$ and the optimal action given state $\boldsymbol{s}$ be $\mathcal{A}^* = \{a_1^*, a_2^*, \cdots, a_k^*\} = \arg\max_{\mathcal{A}} Q^*(\boldsymbol{s}, \mathcal{A})$. The key idea of our cascading Q-function framework comes from the following fact:

$$\max_{a_1, a_2, \cdots, a_k} Q^*(\boldsymbol{s}, a_1, a_2, \cdots, a_k) = \max_{a_1}\left(\max_{a_2, \ldots, a_k} Q^*(\boldsymbol{s}, a_1, a_2, \cdots, a_k)\right), \tag{11}$$

which also implies that there is a cascade of mutually consistent $Q^{1*}, Q^{2*}, \ldots, Q^{k*}$ such that:

$$\begin{aligned}
a_1^* &= \arg\max_{a_1} Q^{1*}(\boldsymbol{s}, a_1) \quad \text{with} \quad Q^{1*}(\boldsymbol{s}, a_1) := \max_{a_2, \cdots, a_k} Q^*(\boldsymbol{s}, a_1, \cdots, a_k), \\
a_2^* &= \arg\max_{a_2} Q^{2*}(\boldsymbol{s}, a_1^*, a_2) \quad \text{with} \quad Q^{2*}(\boldsymbol{s}, a_1, a_2) := \max_{a_3, \cdots, a_k} Q^*(\boldsymbol{s}, a_1, \cdots, a_k), \\
&\cdots\cdots \\
a_k^* &= \arg\max_{a_k} Q^{k*}(\boldsymbol{s}, a_1^*, \cdots, a_{k-1}^*, a_k) \quad \text{with} \quad Q^{k*}(\boldsymbol{s}, a_1, \cdots, a_k) := Q^*(\boldsymbol{s}, a_1, \cdots, a_k).
\end{aligned}$$

Thus, if we have access to all these Q^{j*} functions, then we can obtain an optimal action in $O(k|\mathcal{I}|)$ computations by applying these function in a cascading manner. See algorithm 1 and Figure 3 for a summary. However, these cascade of Q^{j*} functions are usually not available and need to be estimated from data.

5.2 Parametrization and Estimation of Cascading Q-Networks

Note that the set of Q^{j*} functions need to satisfy a large set of constraints which may not be easy to strictly enforce. That is at the optimal point, the value of Q^{j*} is the same as Q^* for all j, i.e.,

$$Q^{j*}(\boldsymbol{s}, a_1^*, \cdots, a_j^*) = Q^*(\boldsymbol{s}, a_1^*, \cdots, a_k^*), \quad \forall j = 1, \ldots, k. \tag{12}$$

Instead of coding these constraints in our neural network parametrization, we will take them into account in a soft and approximate way in our model fitting process.

More specially, we will use the same embedding for the state $\boldsymbol{s}$ as in Eq. (5), and parametrize each Q^{j*} function as

$$\widehat{Q^j}(\boldsymbol{s}, a_{1:j-1}^*, a_j; \Theta_j) = \boldsymbol{q}_j^\top \sigma\left(\boldsymbol{L}_j \left[\boldsymbol{s}^\top, \boldsymbol{f}_{a_1^*}^\top, \ldots, \boldsymbol{f}_{a_{j-1}^*}^\top, \boldsymbol{f}_{a_j}^\top\right]^\top + \boldsymbol{c}_j\right), \ \forall j = 1, \ldots, k \tag{13}$$

where $\boldsymbol{L}_j \in \mathbb{R}^{\ell \times (dn+dj)}$, $\boldsymbol{c}_j \in \mathbb{R}^\ell$, and $\boldsymbol{q}_j \in \mathbb{R}^\ell$ are the set Θ_j of parameters. Now the problem left is how we can estimate these functions $\widehat{Q^j}$ and take into account constraints in Eq. (12).

Different from standard Q-learning, our cascading Q-learning process will then be learning k parametrized functions $\widehat{Q^j}(\boldsymbol{s}^t, a_{1:j-1}^*, a_j; \Theta_j)$ as approximations of Q^{j*}. To enforce the constraints

in Eq. (12) in a soft and approximate way, we can define the loss as

$$\left(y - \widehat{Q^j}\right)^2, \text{ where } y = r(\boldsymbol{s}^t, \mathcal{A}^t, a^t) + \gamma \widehat{Q^k}(\boldsymbol{s}^{t+1}, a_1^*, \cdots, a_k^*; \Theta_k), \ \forall j = 1, \ldots, k. \tag{14}$$

That is all $\widehat{Q^j}$ is fitting against the same target. Then the parameters Θ_k can be updated by performing gradient steps over the above loss.

The overall cascading Q-learning algorithm is summarized in Algorithm 2 in Appendix B, where we employ the cascading Q functions to search the optimal action efficiently (line 9). Besides, both the experience replay (Mnih et al., 2013) and ϵ-exploration techniques are applied. The system's experiences at each time-step are stored in a replay memory set $\mathcal{M}$ (line 11) and then a minibatch of data will be sampled from the replay memory to update $\widehat{Q^j}$ (line 13 and 14). An exploration to the action space is executed with probability ϵ (line 8).

Algorithm 1 Search using $\widehat{Q^j}$ Cascades

function ARGMAX_Q($\boldsymbol{s}, \mathcal{A}, \Theta_1, \cdots, \Theta_k$)
  Let $\mathcal{A}^*$ be empty.
  $\mathcal{I} = \mathcal{A} \setminus \boldsymbol{s}$ ▷ remove clicked items.
  for $j = 1$ to k **do**
    $a_j^* = \arg\max_{a_j \in \mathcal{I} \setminus \mathcal{A}^*} \widehat{Q^j}(\boldsymbol{s}, a_{1:j-1}^*, a_j; \Theta_j)$
    Update $\mathcal{A}^* = \mathcal{A}^* \cup \{a_j^*\}$
  end for
  return $\mathcal{A}^* = (a_1^*, \cdots, a_k^*)$
end function

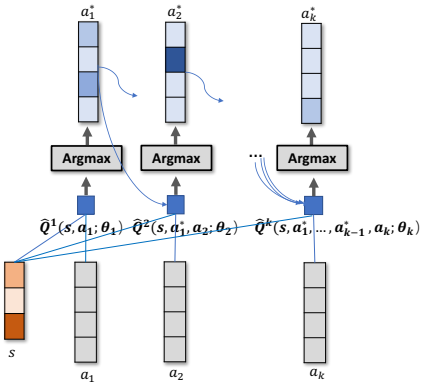

Figure 3: Cascading Q-networks

6 EXPERIMENTS

We conduct three sets of experiments to evaluate our user model (called **G**AN model) and the resulting RL recommendation policy. **(1)** The first set of experiments in section 6.2 show that GAN is a better predictive user model. By taking both the context where the item is displayed in and the evolution of user's interest into account, GAN outperforms other baseline models in terms of both likelihood and precision. **(2)** In section 6.3 we compare the recommendation policies developed from different user models. It is revealed from experiments that recommendation policies generated from GAN model can not only help users achieve higher long term reward but also help the system achieve higher click rate. **(3)** In section 6.4 we test how recommendation policies can benefit from GAN model to better capture users' interest with fewer interactions. A comparison is made among RL policy with two versions of our GAN model (GAN(PW) and GAN(LSTM)), RL policy without GAN model and multi-arm bandit algorithm which does not make an assumption on user behavior.

6.1 DATASET AND FEATURE DESCRIPTION

We will use two real world datasets for our experiments: **(1) MovieLens public dataset** and **(2) an online news article recommendation dataset from Ant Financial**. The datasets are either preprocessed or collected to follow our recommendation setting in Figure 1. More details can be found in Appendix C.

6.2 PREDICTIVE PERFORMANCE OF USER MODEL

A series of models are designed to predict user behavior by utilizing the interaction data with users, among which logistic regression (LR) and collaborative filtering (CF) type models are popular. Besides, collaborative competitive filtering (CCF) model significantly outperforms standard CF approaches as reported by Yang et al. (2011). Thus, we choose the most widely used LR model and the advanced CF model (CCF) as our baseline: **(1) Logistic regression (LR)**. The click-through rate(CTR) likelihood function can be defined using logistic function as $p_i = \frac{1}{1+\exp(-r(\boldsymbol{f}_i))}$, where $r(\cdot)$ is a utility/reward function learned by optimizing $\sum_{i \in \mathcal{C}} \log\left(1+\exp(-r(\boldsymbol{f}_i))\right) + \sum_{i \notin \mathcal{C}} \log\left(1+\exp(r(\boldsymbol{f}_i))\right)$ where $\mathcal{C}$ is the set of clicked items; **(2) Collaborative competitive filtering (CCF)**. CCF predicts users' action by comparing the items in the current display set $\boldsymbol{F}$. More precisely, the likelihood formulation of CCF is $p_i = \exp(r(\boldsymbol{f}_i)) / \sum_{\boldsymbol{f}_j \in \boldsymbol{F}} \exp(r(\boldsymbol{f}_j))$.

GAN, CCF, and LR formulate the probability distribution of user behavior in different ways. For both datasets, we randomly split the users into train(50%), validation(12.5%) and test(37.5%) subsets. The predictive performances are only accessed on users in test set and all results shown in this section are averaged over 5 random splits. We consider the following metrics to evaluate the predictive performance of each user model and report results in table 1.

- **NLL** is the negative log-likelihood loss value averaged over all users in the test set, $\frac{1}{N}\sum_{u=1}^{N}\frac{1}{T_u}\sum_{t=1}^{T_u} -\log p^t_{u,i^*}$, where i^* is the item clicked by the user observed in the data, T_u is number of events associated with user u and p^t_{u,i^*} is the probability that u clicks item i^* at time t. Different user models formulate the probability p^t_{u,i^*} in different ways.
- **Prec@k** is top-k precision. When a user clicks on one of the displayed items, we count it as an event. For each event, the user model can rank the displayed items by likelihood. Prec@k is the proportion of top-k likelihood items that are truly clicked by the user. An average is taken over all events for each user first before average over all users.

Table 1: Predictive performance evaluation: loss and precision of LR, CCF and GAN models assessed on a test set of 25,000 users for the news article dataset and 500 users for MovieLens dataset.

	(1) Ant Financial news dataset			(2) MovieLens dataset		
Model	NLL	prec@1	prec@2	NLL	prec@1	prec@2
LR	2.017(±0.007)	0.375(±0.002)	0.609(±0.001)	0.743(±0.001)	0.615(±0.007)	0.738(±0.012)
CCF	1.457(±0.003)	0.377(±0.001)	0.611(±0.001)	1.164(±0.009)	0.657(±0.008)	0.752(±0.011)
GAN(PW)	**1.368**(±0.001)	**0.419**(±0.001)	**0.658**(±0.001)	**1.141**(±0.013)	**0.666**(±0.007)	**0.754**(±0.013)
GAN(lstm)	**1.365**(±0.001)	**0.421**(±0.002)	**0.659**(±0.002)	**1.124**(±0.021)	**0.674**(±0.005)	**0.763**(±0.012)

As shown in table 1, our GAN user model significantly performs better than baseline models for click predictions. It is noted that the loss of LR is much lower than other models in the MovieLens dataset. This could be caused by the feature reduction process, since we learn the wide&deep layers by minizing the loss function of LR. The learned reduced features tend to achieve a lower loss for LR model. Besides, by comparing the results of GAN(PW) and GAN(lstm), we find that the position weighted(PW) scheme performs almost the same as the LSTM version, but PW is more favorable due to its simple architecture and efficiency. *Thus in the rest of the paper, we will use GAN(PW) for experiments and simply referring it as GAN.*

Another interesting comparison is shown in Figure 4 and more similar figures are provided in Appendix D.1. The blue curve is the trajectory of a user's actual choices of movies over time. The orange curves are simulated trajectories predicted by GAN and CCF. More specifically, at each time t we collect the item with highest likelihood estimated by each model to form a predicted trajectory. To capture the evolution of user's interest, we perform k-means clustering on movie feature vectors to separate the movies into 80 categories. Each data point (t, c) on the curve represents time step t and the category c of the item clicked by the user. The upper sub-figure reveals that GAN performs much better as time goes by. By contrast, the items predicted by CCF as shown in the lower sub-figure are concentrated consistently on several categories. This indicates an obvious drawback of static model - it fails to capture the evolution of user's interests.

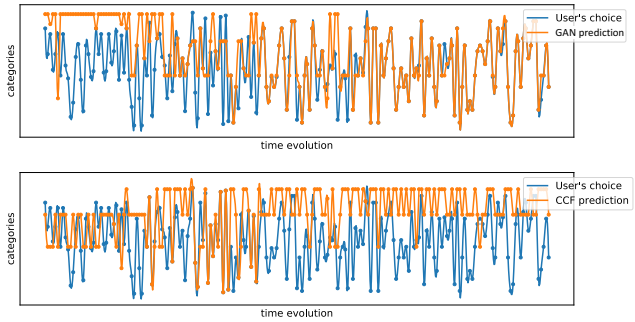

Figure 4: Comparison of the true trajectory(blue) of user's choices, the simulated trajectory predicted by GAN model (orange curve in upper sub-figure) and the simulated trajectory predicted by CCF (orange curve in the lower sub-figure) for the same user. Y-axis represents 80 categories of movies.

6.3 Recommendation Policies Generated from User Models

With a learned user model, one can immediately derive a greedy policy to recommend k items with highest estimated likelihood. In this section, we abuse the notation **LR, CCF and GAN** to represent their **greedy recommendation policy** in correspondence. Apart from that, we include the RL policy based on GAN model in the experiments. We employ a deep Q network to parametrize the RL policy, so we call it **GAN-DQN**.

Since we cannot make real interactions with online users at this moment, we use real data from the news article online platform to fit a user model, which we then use as a test environment. To make the experimental results trustful and solid, we fit test model on a randomly sampled set of 100 users and keep this set isolated. It is noteworthy that the 20 dimensional feature vectors are user-item cross features, so this user model is learned over user-specific features. Now we regard these 100 users as test users and the fitted model as test model. Four recommendation policies(LR, CCF, GAN, GAN-DQN) will then be evaluated among test users. All of these policies are learned from a set of 2,500 users without overlapping with the test users. The performance are evaluated by two metrics: (1) **Cumulative reward**: For each recommendation action, we can observe users' behavior and compute the her reward $r(\boldsymbol{s}^t, a^t)$ using the test model. It should be noted that we never use the reward of test users when we train the policy. The policy is only learned on training set. The numbers shown in table 2 is the cumulative reward averaged over time horizon first and then averaged over all users. It can be formulated as $\frac{1}{100}\sum_{u=1}^{100}\frac{1}{T}\sum_{t=1}^{T} r_u^t$, where r_u^t is the reward received by at time t. (2) **CTR(click through rate)**: it is the ratio of the number of clicks the system receives and the number of steps it is run. This information is not only accessible in our experiment but also in practice. The value displayed in table 2 is also averaged over 100 test users.

Three sets of experiments are conducted and the results are summarized in table 2. Since users' behavior is not deterministic, each policy is evaluated repeatedly for 50 times on test users. There are several observations: (1) Greedy policy built on GAN model is significantly better than the policies built on other models. (2) RL policy learned from GAN is better than the greedy policy. (3) Although DQN is trained to optimize the cumulative reward, it can be seen from the results that by optimizing users' reward, the recommendation system also achieves a higher click through rate.

Table 2: Results of recommendation performances based on different user models.

	$k=2$		$k=3$		$k=5$	
model	reward	CTR	reward	CTR	reward	CTR
LR	11.93(±0.27)	0.38(±0.008)	14.55(±0.24)	0.46(±0.007)	15.40(±0.23)	0.49(±0.008)
CCF	19.13(±0.24)	0.58(±0.007)	21.83(±0.25)	0.67(±0.008)	22.78(±0.23)	0.70(±0.007)
GAN	21.57(±0.40)	0.65(±0.012)	23.67(±0.44)	0.72(±0.013)	24.35(±0.42)	0.74(±0.012)
GAN-CDQN	**22.76**(±0.21)	**0.69**(±0.007)	**24.05**(±0.26)	**0.74**(±0.008)	**25.36**(±0.30)	**0.77**(±0.009)

Since table 2 only shows average value taken over all test users, we compare the policies in user level in figure 5. GAN-DQN policy contributes higher averaged cumulative reward for most users. A similar figure which compares the click rate is attached in Appendix D.

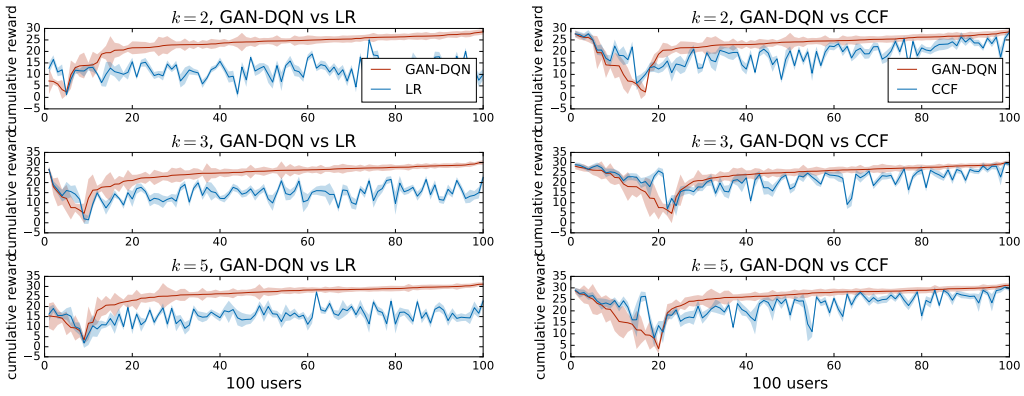

Figure 5: Cumulative rewards among 100 users under the recommendation policies based on different user models. The experiments are repeated for 50 times and standard deviation is plotted as the shaded area.

6.4 User model assisted online adaptive policies

Former results in section 6.2 and 6.3 have demonstrated that GAN is a better user model and its recommendation policy can achieve higher CTR compared to other user models. A question may arise as what if the recommender does not make any assumption on users and why the user model would be useful. In this section, we demonstrate by experiments how in general the user model helps recommendation polices capture users' interest faster and better. The RL policy assisted by GAN user model is compared with other policies that are learned from and adapted to online users: (1) **CDQN with GAN**: cascaded deep Q-learning where the function Q interacts with the learned

GAN user model before it is adapted to online users. (2) **CDQN without GAN**: cascaded deep Q-learning without pre-trained by the GAN model. It interacts with and adapts to online users directly. (3) **LinUCB**: a classic contextual bandit algorithm which assumes adversarial user behavior. We choose its stronger version - LinUCB with hybrid linear models (Li et al., 2010) - to compare with.

The experiment setting is similar to section 6.3. All policies are evaluated on a separated set of 100 users associated with a test model. Three sets of results corresponding to different sizes of display set are plotted in figure 6. It shows how the CTR increases as each policy interacts with and adapts to 100 users over time. In fact, the performances of users' cumulative reward according to different policies are also similar, for which we attach the figure in Appendix D.3.

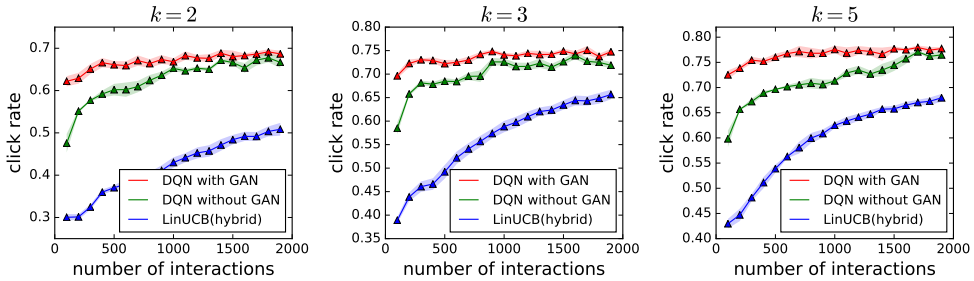

Figure 6: Comparison of the averaged click rate among 100 users under different adaptive recommendation policies. X-axis represents how many times the recommender interacts with online users. Here the recommender interact with 100 users each time, so each interaction represents 100 online data points. Y-axis is the click rate. Each point (x, y) in this figure means a click rate y is achieved after x many times of interactions with the users.

It can be easily seen from figure 6 that the CDQN policy pre-trained over a GAN user model can achieve a better initial status even when it is applied to a new set of users. It can then adapt to online users fast without wasting too many trails or possibly losing the users. Without the user model, CDQN can also adapt to the users during its interaction with them. However, it takes around 1,000 iterations (i.e., 100,000 interactive data points) to achieve similar performance as the CDQN policy assisted by GAN user model. LinUCB(hybrid) is also capturing users' interest during its interaction with users, but similarly, it takes too many interactions. In Appendix D.3, another figure is attached to compare the cumulative reward received by the user instead of click rate. Generally speaking, GAN user model provides a dynamic environment for RL policies to interact with to help the recommendation policy achieve a more satisfying status before applying to online users.

7 Conclusion and Future Work

In this paper, we propose a novel model-based reinforcement learning framework for recommendation systems, where we developed a generative adversarial network to imitate user behavior dynamics and learn her reward function. Using this user model as the simulation environment, we develop a novel DQN algorithm to obtain a combinatorial recommendation policy which can handle a large number of candidate items efficiently. Although we have try very hard to make our experiments realistic, and the experiments show clear benefits of our current method in offline and realistic simulation setting, an even stronger results could be obtained via online experimentation and A/B testing. Such experiments will be left as our future study.

REFERENCES

Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, Rohan Anil, Zakaria Haque, Lichan Hong, Vihan Jain, Xiaobing Liu, and Hemal Shah. Wide & deep learning for recommender systems. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*, DLRS 2016. ACM, 2016. doi: 10.1145/2988450.2988454.

Ignasi Clavera, Anusha Nagabandi, Ronald S. Fearing, Pieter Abbeel, Sergey Levine, and Chelsea Finn. Learning to adapt: Meta-learning for model-based control. *CoRR*, 2018.

M. P. Deisenroth, D. Fox, and C. E. Rasmussen. Gaussian processes for data-efficient learning in robotics and control. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2015. ISSN 0162-8828. doi: 10.1109/TPAMI.2013.218.

Balazs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. Session-based recommendations with recurrent neural networks. In *ICLR*, 2016.

Lihong Li, Wei Chu, John Langford, and Robert E Schapire. A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th international conference on World wide web*, pp. 661–670. ACM, 2010.

Charles F. Manski. Maximum score estimation of the stochastic utility model of choice. *Journal of Econometrics*, pp. 205 – 228, 1975. ISSN 0304-4076.

D. McFadden. Conditional logit analysis of qualitative choice behaviour. In P. Zarembka (ed.), *Frontiers in Econometrics*, pp. 105–142. Academic Press New York, 1973.

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

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Anusha Nagabandi, Gregory Kahn, Ronald S. Fearing, and Sergey Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. *CoRR*, abs/1708.02596, 2017.

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. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

C. J. C. H. Watkins. *Learning from Delayed Rewards*. PhD thesis, King's College, Oxford, May 1989. (To be reprinted by MIT Press.).

Shuang-Hong Yang, Bo Long, Alexander J Smola, Hongyuan Zha, and Zhaohui Zheng. Collaborative competitive filtering: learning recommender using context of user choice. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, pp. 295–304. ACM, 2011.

Xiangyu Zhao, Long Xia, Liang Zhang, Zhuoye Ding, Dawei Yin, and Jiliang Tang. Deep reinforcement learning for page-wise recommendations. 2018a.

Xiangyu Zhao, Liang Zhang, Zhuoye Ding, Dawei Yin, Yihong Zhao, and Jiliang Tang. Deep reinforcement learning for list-wise recommendations. *CoRR*, 2018b.

Guanjie Zheng, Fuzheng Zhang, Zihan Zheng, Yang Xiang, Nicholas Jing Yuan, Xing Xie, and Zhenhui Li. Drn: A deep reinforcement learning framework for news recommendation. 2018.

A LEMMA

A.1 PROOF OF LEMMA 1

Lemma 1. *Let the regularization be $R(\phi) = \log(\phi)$ and $\phi \in \Delta^{k-1}$ include arbitrary mappings. Then the optimal solution ϕ^* for the problem in Eq. (2) has a closed form*

$$\phi^*(\boldsymbol{s}^t, \mathcal{A}^t)_i = \frac{\exp(\eta r(\boldsymbol{s}^t, a_i))}{\sum_{a_j \in \mathcal{A}^t} \exp(\eta r(\boldsymbol{s}^t, a_j))}. \tag{3}$$

Furthermore, at each time t, the user's decision according to her optimal policy ϕ^* is equivalent to he following discrete choice model

$$a^t = \arg\max_{a \in \mathcal{A}^t} \ \eta\, r(\boldsymbol{s}^t, a) + \varepsilon^t, \tag{4}$$

where ε^t follows a Gumbel distribution.

Proof. First, recall the problem defined in Eq. (2):

$$\phi^*(\boldsymbol{s}^t, \mathcal{A}^t) = \arg\max_{\phi \in \Delta^{k-1}} \mathbb{E}_\phi \left[r(\boldsymbol{s}^t, a^t) \right] - \frac{1}{\eta} \mathbb{E}_\phi[R(\phi)].$$

Denote $\phi^t = \phi(\boldsymbol{s}^t, \mathcal{A}^t)$. Since ϕ can be an arbitrary mapping (i.e., ϕ is not limited in a specific parameter space), ϕ^t can be an arbitrary vector in Δ^{k-1}. Recall the notation $\mathcal{A}^t = \{a_1, \cdots, a_k\}$. Then the expectation taken over random variable $a^t \in \mathcal{A}^t$ can be written as

$$\mathbb{E}_\phi \left[r(\boldsymbol{s}^t, a^t) \right] - \frac{1}{\eta} \mathbb{E}_\phi[R(\phi)] = \sum_{i=1}^k \phi_i^t r(\boldsymbol{s}^t, a_i) - \frac{1}{\eta} \sum_{i=1}^k \phi_i^t \log \phi_i^t. \tag{15}$$

By simple computation, the optimal vector $\phi^{t*} \in \Delta^{k-1}$ which maximizes Eq. (15) is

$$\phi_i^{t*} = \frac{\exp(\eta r(\boldsymbol{s}^t, a_i))}{\sum_{j=1^k} \exp(\eta r(\boldsymbol{s}^t, a_j))}, \tag{16}$$

which is equivalent to Eq. (2). Next, we show the equivalence of Eq. (16) to the discrete choice model interpreted by Eq. (4).

The cumulative distribution function for the Gumbel distribution is $F(\varepsilon; \alpha) = \mathbb{P}[\varepsilon \leqslant \alpha] = e^{-e^{-\alpha}}$ and the probability density is $f(\varepsilon) = e^{-e^{-\varepsilon}} e^{-\varepsilon}$. Using the definition of the Gumbel distribution, the probability of the event $[a^t = a_i]$ where a^t is defined in Eq. (4) is

$$\begin{aligned} P_i := \mathbb{P}\Big[a^t = a_i\Big] &= \mathbb{P}\Big[\eta r(\boldsymbol{s}^t, a_i) + \varepsilon_i \geqslant \eta r(\boldsymbol{s}^t, a_j) + \varepsilon_j, \text{ for all } i \neq j\Big] \\ &= \mathbb{P}\Big[\varepsilon_j \leqslant \varepsilon_i + \eta r(\boldsymbol{s}^t, a_i) - \eta r(\boldsymbol{s}^t, a_j), \text{ for all } i \neq j\Big]. \end{aligned}$$

Suppose we know the random variable ε_i. Then we can compute the choice probability P_i conditioned on this information. Let $B_{ij} = \varepsilon_i + \eta r(\boldsymbol{s}^t, a_i) - \eta r(\boldsymbol{s}^t, a_j)$ and $P_{i|\mathcal{E}}$ be the conditional probability; then we have

$$P_{i|\varepsilon_i} = \prod_{i \neq j} \mathbb{P}[\varepsilon_j \leqslant B_{ij}] = \prod_{i \neq j} e^{-e^{-B_{ij}}}.$$

In fact, we only know the density of ε_i. Hence, using the Bayes theorem, we can express P_i as

$$\begin{aligned} P_i &= \int_{-\infty}^{\infty} P_{i|\varepsilon_i} f(\varepsilon_i) \mathrm{d}\varepsilon_i = \int_{-\infty}^{\infty} \prod_{i \neq j} e^{-e^{-B_{ij}}} f(\varepsilon_i) \mathrm{d}\varepsilon_i \\ &= \int_{-\infty}^{\infty} \prod_{j=1}^k e^{-e^{-B_{ij}}} e^{e^{-\varepsilon_i}} e^{-e^{-\varepsilon_i}} e^{-\varepsilon_i} \mathrm{d}\varepsilon_i = \int_{-\infty}^{\infty} \Big(\prod_{j=1}^k e^{-e^{-B_{ij}}} \Big) e^{-\varepsilon_i} \mathrm{d}\varepsilon_i \end{aligned}$$

Now, let us look at the product itself.

$$\prod_{j=1}^{k} e^{-e^{-B_{ij}}} = \exp\Big(-\sum_{j=1}^{k} e^{-B_{ij}}\Big)$$
$$= \exp\Big(-e^{-\varepsilon_i}\sum_{j=1}^{k} e^{-(\eta r(\boldsymbol{s}^t,a_i)-\eta r(\boldsymbol{s}^t,a_j))}\Big)$$

Hence

$$P_i = \int_{-\infty}^{\infty} \exp(-e^{-\varepsilon_i}Q)e^{-\varepsilon_i}\mathrm{d}\varepsilon_i$$

where $Q = \sum_{j=1}^{k} e^{-(\eta r(\boldsymbol{s}^t,a_i)-\eta r(\boldsymbol{s}^t,a_j))} = Z/\exp(\eta r(\boldsymbol{s}^t,a_i))$.

Next, we make a change of variable $y = e^{-\varepsilon_i}$. The Jacobian of the inverse transform is $J = \frac{\mathrm{d}\varepsilon_i}{\mathrm{d}y} = -\frac{1}{y}$. Since $y > 0$, the absolute of Jacobian is $|J| = \frac{1}{y}$. Therefore,

$$P_i = \int_0^{\infty} \exp(-Qy)y|J|\mathrm{d}y = \int_0^{\infty} \exp(-Qy)\mathrm{d}y$$
$$= \frac{1}{Q} = \frac{1}{\exp(-\eta r(\boldsymbol{s}^t,a_i))\sum_j \exp(\eta r(\boldsymbol{s}^t,a_j))}$$
$$= \frac{\exp(\eta r(\boldsymbol{s}^t,a_i)}{\sum_{j=1}^{k}\exp(\eta r(\boldsymbol{s}^t,a_j))}.$$

□

A.2 Proof of Lemma 2

Lemma 2. *Consider the case when regularization is defined as $R(\phi) = \log(\phi)$ and Φ includes all mappings from $\mathcal{S} \times \binom{\mathcal{I}}{k}$ to Δ^{k-1}. Then the optimization problem in Eq. (7) is equivalent to the following maximum likelihood estimation*

$$\max_{\theta\in\Theta} \prod_{t=1}^{T} \frac{\exp(\eta r_\theta(\boldsymbol{s}^t_{true}, a^t_{true}))}{\sum_{a^t\in\mathcal{A}^t}\exp(\eta r_\theta(\boldsymbol{s}^t_{true}, a^t))}. \tag{8}$$

Proof. This lemma is a straight forward result of lemma 1. First, recall the problem defined in Eq. (7):

$$\min_{\theta\in\Theta}\left(\max_{\phi\in\Phi}\mathbb{E}_\phi\left[\sum_{t=1}^{T} r_\theta(\boldsymbol{s}^t_{true}, a^t) - \frac{1}{\eta}R(\phi)\right]\right) - \sum_{t=1}^{T} r_\theta(\boldsymbol{s}^t_{true}, a^t_{true})$$

We make a assumption that there is no repeated pair $(\boldsymbol{s}^t_{true}, a^t)$ in Eq. (7). This is a very soft assumption because $\boldsymbol{s}^t_{true}$ is updated overtime, and a^t is in fact representing its feature vector $\boldsymbol{f}^t_{a^t}$, which is in space $\mathbb{R}^d$. With this assumption, we can let ϕ map each pair $(\boldsymbol{s}^t_{true}, a^t)$ to the optimal vector ϕ^{t*} which maximize $r_\theta(\boldsymbol{s}^t_{true}, a^t) - \frac{1}{\eta}R(\phi^t)$ since there is no repeated pair. Using Eq. (16), we have

$$\max_{\phi\in\Phi}\mathbb{E}_\phi\left[\sum_{t=1}^{T} r_\theta(\boldsymbol{s}^t_{true}, a^t) - \frac{1}{\eta}R(\phi)\right] = \max_{\phi\in\Phi}\sum_{t=1}^{T}\mathbb{E}_\phi\left[r_\theta(\boldsymbol{s}^t_{true}, a^t) - \frac{1}{\eta}R(\phi)\right]$$
$$= \sum_{t=1}^{T}\left(\sum_{i=1}^{k}\phi_i^{t*}r(\boldsymbol{s}^t,a_i) - \frac{1}{\eta}\sum_{i=1}^{k}\phi_i^{t*}\log\phi_i^{t*}\right) = \sum_{t=1}^{T}\frac{1}{\eta}\log\Big(\sum_{i=1}^{k}\exp(\eta r_\theta(\boldsymbol{s}^t_{true}, a_i))\Big).$$

Eq. (7) can then be written as

$$\min_{\theta\in\Theta}\sum_{t=1}^{T}\frac{1}{\eta}\log\Big(\sum_{i=1}^{k}\exp(\eta r_\theta(\boldsymbol{s}^t_{true}, a_i))\Big) - \sum_{t=1}^{T} r_\theta(\boldsymbol{s}^t_{true}, a^t_{true}),$$

which is the negative log-likelihood function and is equivalent to lemma 2. □

B ALOGRITHM BOX

Algorithm 2 Cascading deep Q-learning (CDQN) with Experience Replay

Initialize replay memory $\mathcal{M}$ to capacity N
Initialize parameter Θ_j of $\widehat{Q^j}$ with random weights for each $1 \leq j \leq k$
for iteration $i = 1$ to L **do**
    Sample a batch of users $\mathcal{U}$ from training set
    Initialize the states $\boldsymbol{s}^0$ to a zero vector for each $u \in \mathcal{U}$
    for $t = 1$ to T **do**
        for each user $u \in \mathcal{U}$ simultaneously **do**
            With probability ϵ select a random subset $\mathcal{A}^t$ of size k
            Otherwise, $\mathcal{A}^t = \text{ARGMAX_Q}(\boldsymbol{s}_u^t, \mathcal{I}^t, \Theta_1, \cdots, \Theta_k)$
            Recommend $\mathcal{A}^t$ to user u, observe user action $a^t \sim \phi(\boldsymbol{s}^t, \mathcal{A}^t)$ and update user state $\boldsymbol{s}^{t+1}$
            Add tuple $\left(\boldsymbol{s}^t, \mathcal{A}^t, r(\boldsymbol{s}^t, a^t), \boldsymbol{s}^{t+1}\right)$ to $\mathcal{M}$
        end for
        Sample random minibatch $B \overset{\text{iid.}}{\sim} \mathcal{M}$
        For each j, update Θ_j by SGD over the loss $\left(y - \widehat{Q^j}(\boldsymbol{s}^t, A_{1:j}^t; \Theta_j)\right)^2$ for B
    end for
end for
return $\Theta_1, \cdots, \Theta_k$

C DATASET DESCRIPTION

(1) MovieLens public dataset contains large amounts of movie ratings collected from their website. We randomly sample 1,000 active users from this dataset. On average, each of these active users rated more than 500 movies (including short films), so we assume they rated almost every movie that they watched and thus equate their rating behavior with watching behavior. MovieLens dataset is the most suitable public dataset for our experiments, but it is still not perfect. In fact, none of the public datasets provides the context in which a user's choice is made. Thus, we simulate this missing information in a reasonable way. For each movie watched(rated) on the date d, we collect a list of movies released within a month before that day d. On average, movies run for about four weeks in theater. Even though we don't know the actual context of user's choice, at least the user decided to watch the rated movie instead of other movies in theater. Besides, we control the maximal size of each displayed set by 40. **Features:** In MovieLens dataset, only titles and IDs of the movies are given, so we collect detailed movie information from Internet Movie Database(IMDB). Categorical features as encoded as sparse vectors and descriptive features are encoded as dense vectors. The combination of such two types of vectors produces 722 dimensional raw feature vectors. To further reduce dimensionality, we use logistic regression to fit a wide&deep networks (Cheng et al., 2016) and use the learned input and hidden layers to reduce the feature to 10 dimension.

(2) An online news article recommendation dataset from Ant Financial is anonymously collected from Ant Financial news article online platform. It consists of 50,000 users' clicks and impression logs for one month, involving dozens of thousands of news. It is a time-stamped dataset which contains user features, news article features and the context where the user clicks the articles. The size of the display set is not fixed, since a user can browse the news article platform as she likes. On average a display set contains 5 new articles, but it actually various from 2 to 10. **Features:** The news article raw features are approximately of dimension 100 million because it summarizes the key words in the article. Apparently it is too expensive to use these raw features in practice. The features we use in the experiments are 20 dimensional dense vector embedding produced from the raw feature by wide&deep networks. The reduced 20 dimensional features are widely used in this online platform and revealed to be effective in practice.

D MORE FIGURES FOR EXPERIMENTAL RESULTS

D.1 FIGURES FOR SECTION 6.2

An interesting comparison is shown in Figure 4 and more similar figures are provided here. The blue curve is the trajectory of a user's actual choices of movies over time. The orange curves are simulated trajectories predicted by GAN and CCF, respectively. Similar to what we conclude in section 6.2, these figures reveal the good performances of GAN user model in terms of capturing the evolution of users' interest.

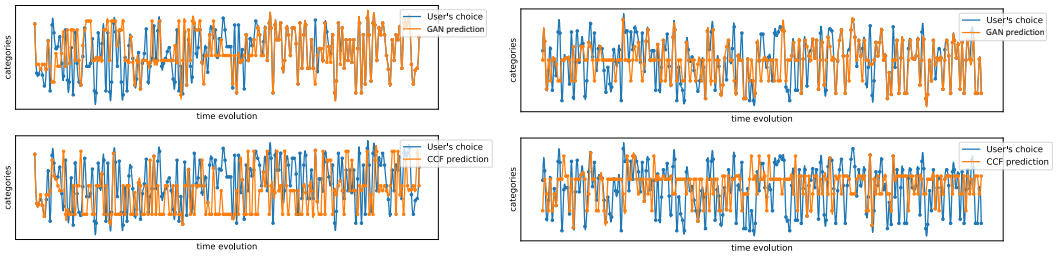

Figure 7: Two more examples: comparison of the true trajectory(blue) of user's choices, the simulated trajectory predicted by GAN model (orange curve in upper sub-figure) and the simulated trajectory predicted by CCF (orange curve in the lower sub-figure) for the same user. Y-axis represents 80 categories of movies.

D.2 FIGURES FOR SECTION 6.3

We demonstrate the policy performance in user level in figure 5 by comparing the cumulative reward. Here we attach the figure which compares the click rate. In each sub-figure, red curve represents GAN-DQN policy and blue curve represents the other. GAN-DQN policy contributes higher averaged click rate for most users.

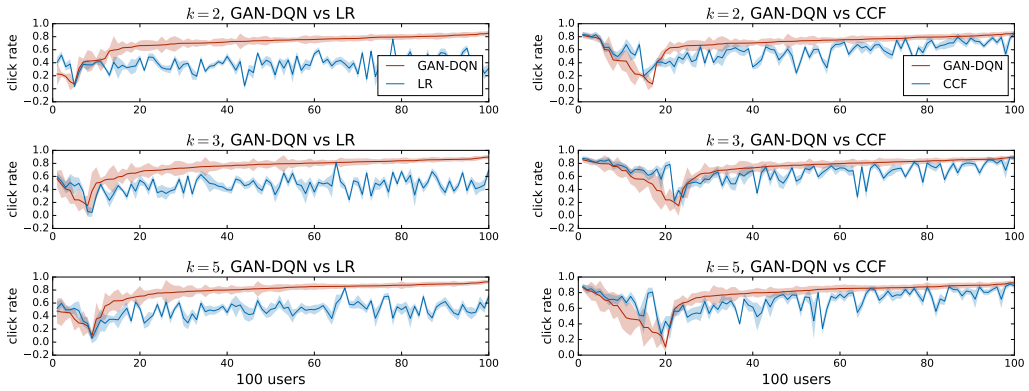

Figure 8: Comparison of click rates among 100 users under the recommendation policies based on different user models. In each figure, red curve represents GAN-DQN policy and blue curve represents the other. The experiments are repeated for 50 times and standard deviation is plotted as the shaded area. This figure is similar to figure 5, except that it plots the value of click rates instead of user's cumulative rewards.

D.3 FIGURES FOR SECTION 6.4

This figure shows three sets of results corresponding to different sizes of display set. It reveals how users' cumulative reward(averaged over 100 users) increases as each policy interacts with and adapts to 100 users over time. It can be easily that the CDQN policy pre-trained over a GAN user model can adapt to online users much faster then other model-free policies and can reduce the risk of losing

the user at the beginning. The experiment setting is similar to section 6.3. All policies are evaluated on a separated set of 100 users associated with a test model. We need to emphasize that the GAN model which assists the CDQN policy is learned from a training set of users without overlapping test users. It is different from the test model which fits the 100 test users.

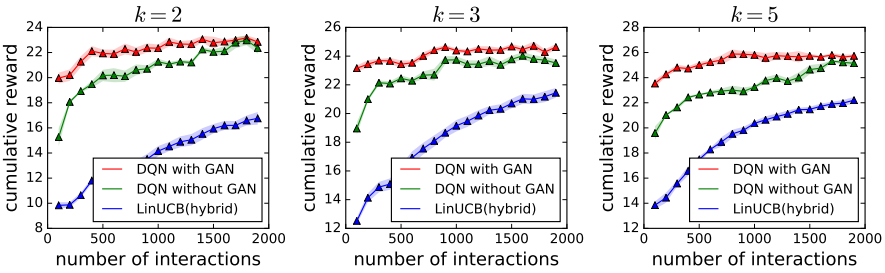

Figure 9: Comparison of the averaged cumulative reward among 100 users under different adaptive recommendation policies. X-axis represents how many times the recommender interacts with online users. Here the recommender interact with 100 users each time, so in fact each interaction represents 100 online data points. Y-axis is the click rate. Each point (x, y) in this figure means a click rate y is achieved after x many times of interactions with the users.