Generative Adversarial User Model for Reinforcement Learning Based Recommendation System

Xinshi Chen [1] **Shuang Li** [1] **Hui Li** [2] **Shaohua Jiang** [2] **Yuan Qi** [2] **Le Song** [1 2]

Abstract

There are great interests as well as many challenges in applying reinforcement learning (RL) to recommendation systems. In this setting, an online user is the environment; neither the reward function nor the environment dynamics are 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 develop 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 Cascading 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 a better long-term 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 they 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 hiding the rest. Thus, it is more favorable to design a recommendation strategy, such as one based on reinforcement learning (RL), which can take users' long-term interest into account. However, it is challenging to apply RL framework to recommendation system setting since the environment will correspond to the logged online user.

First, a user's interest (reward function) driving her behavior is typically unknown, yet it is critically important for the use of RL algorithms. In existing RL algorithms for recommendation systems, the reward functions are manually designed (e.g. ± 1 for click/no-click) which may not reflect a user's preference over different items (Zhao et al., 2018a; Zheng et al., 2018).

Second, model-free RL typically requires lots of interactions with the environment in order to learn a good policy. This is impractical in the recommendation system setting. An online user will quickly abandon the service if the recommendation looks random and do not meet her 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 works showed that model-based RL is much more sample efficient (Nagabandi et al., 2017; Deisenroth et al., 2015; Clavera et al., 2018). The advantage of model-based approaches is that potentially large amount of off-policy data can be pooled and used to learn a good environment dynamics model, whereas model-free approaches can only use expensive on-policy data for learning. However, previous model-based approaches are typically designed based on physics or Gaussian processes, and not tailored for complex sequences of user behaviors.

To address the above challenges, we propose a novel model-based RL framework for recommendation systems, where a user behavior model and the associated reward function are learned in unified mini-max framework, and then RL policies are learned using this model. Our main technical contributions are:

1. We develop a generative adversarial learning (GAN) formulation to model user behavior dynamics and recover her reward function. These two components are

[1]Georgia Institute of Technology, Atlanta, Georgia, USA [2]Ant Financial, Hangzhou, China. Correspondence to: Xinshi Chen <xinshi.chen@gatech.edu>.

Proceedings of the 36^{th} International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019.

estimated simultaneously via a joint mini-max optimization algorithm. The benefits of our formulation are: (i) a more predictive user model can be obtained, and the reward function are learned in a consistent way with the user model; (ii) the learned reward allows later reinforcement learning to be carried out in a more principled way, rather than relying on manually designed reward; (ii) the learned user model allows us to perform model-based RL and online adaptation for new users to achieve better results.

2. Using this model as the simulation environment, we also develop a cascading DQN algorithm to obtain a combinatorial recommendation policy. The cascading design of action-value function allows us to find the best subset of items to display from a large pool of candidates with time complexity only linear in the number of candidates.

In our experiments with real data, we showed that this generative adversarial model is a better fit to user behavior in terms of held-out likelihood and click prediction. 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 a much fewer number of user interactions compared to model-free approaches.

2. Related Work

Commonly used recommendation algorithms use a simple user model. For instance, Wide&Deep networks (Cheng et al., 2016) and other methods such as XGBOOST (Chen & Guestrin, 2016) and DFM (Guo et al., 2017) based on logistic regression assume a user chooses each item independently; Collaborative competitive filtering (Yang et al., 2011) takes into account the context of user's choice but assumes that user's choices in each page view are independent. Session-based RNN (Hidasi et al., 2016) and session-based KNN (Jannach & Ludewig, 2017) improve upon previous approaches by modeling users' history, but this model does not recover a users' reward function and can not be used subsequently for reinforcement learning. Bandit based approaches, such as LinUCB (Li et al., 2010), can deal with adversarial user behaviors, but the reward is updated in a Bayesian framework and can not be directly in 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 reduced sample complexity to obtain a good policy (Deisenroth et al., 2015; Nagabandi et al., 2017; Clavera et al., 2018). However, these approaches can not be used in the recommendation setting, as a user behavior model typically consists of sequences of discrete choices under a complex session context.

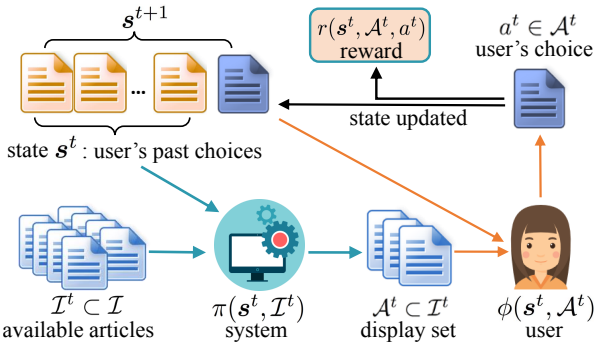

Figure 1. Illustration of the interaction between a user and the recommendation system. Green arrows represent the recommender information flow and orange represents user's information flow.

3. Setting and RL Formulation

We focus on a simplistic yet typical setting where the recommendation system (RS) and its user interact as follows:

> **Setting:** RS displays k items in one page to a user. The user provides feedback by clicking on one or none of these items, and then the system recommends a new page of k items.

Our model can be easily extended to settings with more complex page views and user interactions, but these settings are left for future studies.

Since reinforcement learning can take into account long-term reward, it holds the promise to improve users' long-term engagement with an online platform. In the RL framework, a recommendation system aims to find a policy $\pi(s, \mathcal{I})$ to choose from a set $\mathcal{I}$ of items in user state s, such that the long-term expected reward to the user is maximized,

$$\pi^* = \arg\max_{\pi(s^t, \mathcal{I}^t)} \mathbb{E}\Big[\sum_{t=0}^{\infty} \gamma^t r(s^t, a^t)\Big], \tag{1}$$

where $s^0 \sim p^0$, $\mathcal{A}^t \sim \pi(s^t, \mathcal{I}^t)$, $s^{t+1} \sim P(\cdot|s^t, \mathcal{A}^t)$, $a^t \in \mathcal{A}^t$. The overall RL framework for recommendation is illustrated in Figure 1. Several key aspects are as follows:

(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 page view (or interaction);
(2) **State** $s^t \in \mathcal{S}$: will correspond to an ordered sequence of a user's historical clicks;
(3) **Action** $\mathcal{A}^t \in \binom{\mathcal{I}^t}{k}$ of the recommender: will correspond to a subset of k items chosen by the recommender from $\mathcal{I}^t$ to display to the user. $\binom{\mathcal{I}^t}{k}$ means the set of all subsets of k items of $\mathcal{I}^t$, where $\mathcal{I}^t \subset \mathcal{I}$ are available items to recommend at time t.
(4) **State Transition** $P(\cdot|s^t, \mathcal{A}^t) : \mathcal{S} \times \binom{\mathcal{I}}{k} \mapsto \mathcal{P}(\mathcal{S})$: will

correspond to a user behavior model which returns the transition probability for $\boldsymbol{s}^{t+1}$ given previous state $\boldsymbol{s}^t$ and the set of items $\mathcal{A}^t$ displayed by the system. It is equivalent to the distribution $\phi(\boldsymbol{s}^t, \mathcal{A}^t)$ over a user's actions, which is defined in our user model 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 a user's utility or satisfaction after making her choice $a^t \in \mathcal{A}^t$ in state $\boldsymbol{s}^t$. Here we assume that the reward to the recommendation system is the same as the user's utility. 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 company's benefit to the reward, but we will focus on users' satisfaction.
(6) **Policy** $\mathcal{A}^t \sim \pi(\boldsymbol{s}^t, \mathcal{I}^t) : \mathcal{S} \times 2^{\mathcal{I}} \mapsto \mathcal{P}(\binom{\mathcal{I}}{k})$: will correspond to a recommendation strategy which returns the probability of displaying a subset $\mathcal{A}^t$ of $\mathcal{I}^t$ in user state $\boldsymbol{s}^t$.

Remark. We note that in the above mapping, *Environment, State* and *State Transition* are associated with the user, *Action* and *Policy* are associated with the recommendation system, and *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, as the user can only choose from the display set. However, the value of the reward is determined by the user's state and the clicked item once the item occurs in the display set $\mathcal{A}^t$. In fact, $r\ (\boldsymbol{s}^t, \mathcal{A}^t, a^t) = r\ (\boldsymbol{s}^t, a^t) \cdot \mathbf{1}(a^t \in \mathcal{A}^t)$. Thus, in section 4.1 where we discuss 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.

Since both the reward function and the state transition model are unknown, we need to learn them from data. Once they are learned, the optimal policy π^* in Eq. (1) can be estimated by repeated querying the model using algorithms such as Q-learning (Watkins, 1989). In the next two sections, we will explain our formulation for the user behavior model and the reward function, and design an efficient algorithm for learning the RL recommendation policy.

4. Generative Adversarial User Model

We propose a model to imitate users' sequential choices and discuss its parameterization and estimation. The formulation is inspired by imitation learning, a powerful tool for learning sequential decision-making policies from expert demonstrations (Abbeel & Ng, 2004; Ho et al., 2016; Ho & Ermon, 2016; Torabi et al., 2018). We formulate a unified mini-max optimization to learn user behavior model and reward function simultaneously based on sample trajectories.

4.1. User Behavior As Reward Maximization

We model user behavior based on two realistic assumptions. (i) Users are not passive. Instead, given a set of k items, a user will make a choice to maximize her own reward. The reward r measures how much she will be satisfied with or interested in an item. Alternatively, the user can choose not to click on any items. Then she will receive the reward of not wasting time on boring items. (ii) The reward depends not only on the selected item but also on the user's history. For example, a user may not be interested in *Taylor Swift*'s song at the beginning, but once she happens to listen to it, she may like it and then becomes interested in her other songs. Also, a user can get bored after listening to *Taylor Swift*'s songs repeatedly. In other words, a user's evaluation of the items varies in accordance with her personal experience.

To formalize the model, we consider both the clicked item and the state of the user as the inputs to the reward function $r(\boldsymbol{s}^t, a^t)$, where the clicked item is the user's action a^t and the user's history is captured in her state $\boldsymbol{s}^t$ (non-click is treated as a special item/action). Suppose in session t, the user is presented with k items $\mathcal{A}^t = \{a_1, \cdots, a_k\}$ and their associated features $\{\boldsymbol{f}_1^t, \cdots, \boldsymbol{f}_k^t\}$ by the system. She will take an action $a^t \in \mathcal{A}^t$ according to a strategy ϕ^* which can maximize her expected reward. More specifically, this strategy is a probability distribution over $\mathcal{A}^t$, which is the result of the optimization problem below

Generative User Model:

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

where Δ^{k-1} is the probability simplex, and $R(\phi)$ is a convex regularization function to encourage exploration, and η controls the strength of the regularization.

Model interpretation. If we use the negative Shannon entropy as the regularizer, we can obtain an interpretation of our user model from the perspective of exploration-exploitation trade-off (See Appendix A for a proof).

Lemma 1. *Let the regularization term in Eq. (2) be* $R(\phi) = \sum_{i=1}^k \phi_i \log \phi_i$. *Then the optimal solution* ϕ^* *for the problem in Eq. (2) has a closed form*

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

Furthermore, in each session t, the user's optimal policy ϕ^* *is equivalent to the following discrete choice model where* ε^t *follows a Gumbel distribution.*

$$a^t = \arg\max_{a \in \mathcal{A}^t} \eta\, r(\boldsymbol{s}^t, a) + \varepsilon^t. \quad (3)$$

Essentially, this lemma makes it clear that the user greedily picks an item according to the reward function (exploitation), and yet the Gumbel noise ε^t allows the user to deviate and explore other less rewarding items. Similar models have also appeared in the econometric choice model (Manski, 1975; McFadden, 1973). The regularization parameter η is an exploration-exploitation trade-off parameter. It can be easily seen that with a smaller η, the user is more exploratory.

Thus, η reveals a part of users' character. In practice, we simply set the value $\eta = 1$ in our experiments, since it is implicitly learned in the reward r, which is a function of various features of a user.

Remark. (i) 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 the closed form. (ii) The case where the user does not click any items can be regarded as a special item which is always in the display set $\mathcal{A}^t$. It can be defined as an item with zero feature vector, or, alternatively, its reward value can be defined as a constant to be learned.

4.2. Model Parameterization

We will represent the state $\boldsymbol{s}^t$ as an embedding of the historical sequence of items clicked by the user before session t, and then we will define the reward function $r(\boldsymbol{s}^t, a^t)$ based on the state and the embedding of the current action a^t.

First, we will define the state of the user as $\boldsymbol{s}^t := h(\boldsymbol{F}_*^{1:t-1} := [\boldsymbol{f}_*^1, \cdots, \boldsymbol{f}_*^{t-1}])$, where each $\boldsymbol{f}_*^\tau \in \mathbb{R}^d$ is the feature vector of the clicked item at session τ and $h(\cdot)$ is an embedding function. One can also define a truncated M-step sequence as $\boldsymbol{F}_*^{t-m:t-1} := [\boldsymbol{f}_*^{t-m}, \cdots, \boldsymbol{f}_*^{t-1}]$. For the state embedding function $h(\cdot)$, we propose a simple and effective position weighting scheme. Let $\boldsymbol{W} \in \mathbb{R}^{m \times n}$ be a matrix where the number of rows m corresponds to a fixed number of historical steps, and each column corresponds to one set of importance weights on positions. Then the embedding function $h \in \mathbb{R}^{dn \times 1}$ can be designed as

$$\boldsymbol{s}^t = h(\boldsymbol{F}_*^{t-m:t-1}) := vec[\,\sigma\left(\boldsymbol{F}_*^{t-m:t-1}\boldsymbol{W} + \boldsymbol{B}\right)], \quad (4)$$

where $\boldsymbol{B} \in \mathbb{R}^{d \times n}$ is a bias matrix, and $\sigma(\cdot)$ is a nonlinear activation such as ReLU and ELU, 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 the position weighting scheme is that the history embedding is produced by a shallow network. It is more efficient for forward-computation and gradient backpropagation.

Next, we define the reward function and the user behavior model. A user's choice $a^t \in \mathcal{A}^t$ will correspond to an item with feature $\boldsymbol{f}_{a^t}^t$, which will be used to parameterize the reward function and user behavior model as

$$r(\boldsymbol{s}^t, a^t) := \boldsymbol{v}^\top \sigma\Big(\boldsymbol{V}\left[(\boldsymbol{s}^t)^\top, (\boldsymbol{f}_{a^t}^t)^\top\right]^\top + \boldsymbol{b}\Big) \text{ and}$$

$$\phi(\boldsymbol{s}, \mathcal{A}^t) \propto \exp\Big(\boldsymbol{v'}^\top \sigma\Big(\boldsymbol{V'}\left[(\boldsymbol{s}^t)^\top, (\boldsymbol{f}_{a^t}^t)^\top\right]^\top + \boldsymbol{b'}\Big)\Big),$$

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 $\boldsymbol{v}, \boldsymbol{v'} \in \mathbb{R}^\ell$ are the final regression parameters. See Figure 2 for an illustration of the overall parameterization. For simplicity of notation, we will denote the set of all parameters in the reward function as θ and the set of all parameters in the user model as α, and hence the notation r_θ and ϕ_α respectively.

4.3. Generative Adversarial Training

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 the data. The behavior model ϕ tries to mimic the action sequences provided by a real user who acts to maximize her reward function r. In analogy to generative adversarial networks, (i) ϕ acts as a generator which generates the user's next action based on her history, and (ii) r acts as a discriminator which tries to differentiate the user's actual actions from those generated by the behavior model ϕ. Thus, inspired by the GAN framework, we estimate ϕ and r simultaneously via a mini-max formulation.

More precisely, given a trajectory of T observed actions $\{a_{true}^1, a_{true}^2, \ldots, a_{true}^T\}$ of a user and the corresponding clicked item features $\{\boldsymbol{f}_*^1, \boldsymbol{f}_*^2, \ldots, \boldsymbol{f}_*^T\}$, we learn the user behavior model and reward function jointly by solving the following mini-max optimization

Generative Adversarial Training:

$$\min_\theta \max_\alpha \left(\mathbb{E}_{\phi_\alpha}\left[\sum_{t=1}^T r_\theta(\boldsymbol{s}_{true}^t, a^t)\right] - R(\phi_\alpha)/\eta\right) - \sum_{t=1}^T r_\theta(\boldsymbol{s}_{true}^t, a_{true}^t), \quad (5)$$

where we use $\boldsymbol{s}_{true}^t$ to emphasize that this is observed in the data. From the above optimization, one can see that the reward r_θ will extract some statistics from both real user actions and model user actions, and try to magnify their difference (or make their negative gap larger). In contrast, the user model ϕ_α will try to make the difference smaller, and hence more similar to the real user behavior. Alternatively, the mini-max optimization can also be interpreted as a game between an adversary and a learner where the adversary tries to minimize the reward of the learner by adjusting r_θ, while the learner tries to maximize its reward by adjusting ϕ_α to counteract the adversarial moves. This gives the user behavior training process a large-margin training flavor, where we want to learn the best model even for the worst scenario.

For general regularization function $R(\phi_\alpha)$, the optimal solution in Eq. (5) does not have a closed form, and typically needs to be solved by alternatively updating ϕ_α and r_θ, e.g.

$$\begin{cases} \alpha \leftarrow \alpha + \gamma_1 \nabla_\alpha \mathbb{E}_{\phi_\alpha}\left[\sum_{t=1}^T r_\theta\right] - \gamma_1 \nabla_\alpha R(\phi_\alpha)/\eta; \\ \theta \leftarrow \theta - \gamma_2 \mathbb{E}_{\phi_\alpha}\left[\sum_{t=1}^T \nabla_\theta r_\theta\right] + \gamma_2 \sum_{t=1}^T \nabla_\theta r_\theta. \end{cases} \quad (6)$$

The process may be unstable due to the non-convexity nature of the problem. To stabilize the training process, we will leverage a special regularization for initializing the training process. More specifically, for entropy regularization, we can obtain a closed form solution to the inner-maximization for user behavior model, which makes the learning of reward

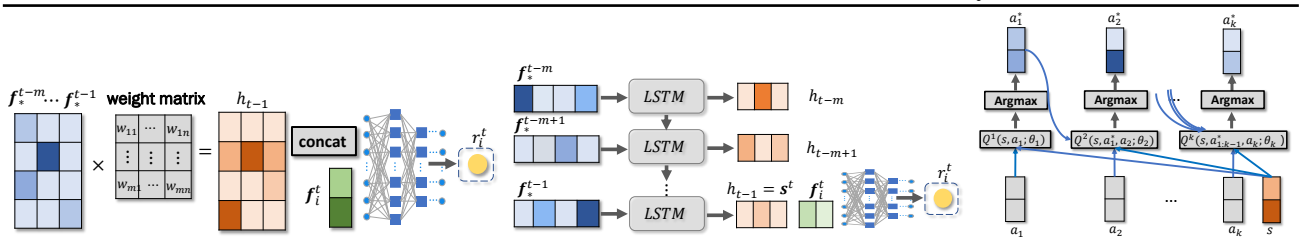

Figure 2. Architecture of our models parameterized by either (a) position weight (PW) or (b) LSTM. (c) Cascading Q-networks.

function easy (See lemma 2 below and Appendix A for a proof). Once the reward function is learned for entropy regularization, it can be used to initialize the learning in the case of other regularization functions which may induce different user behavior models and final rewards.

Lemma 2. *When $R(\phi) = \sum_{i=1}^{k} \phi_i \log \phi_i$, the optimization problem in Eq. (5) is equivalent to the following maximum likelihood estimation*

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

5. Cascading RL Policy for Recommendation

Using the estimated user behavior model ϕ and the corresponding reward function r as the simulation environment, we can then use reinforcement learning to obtain a recommendation policy. The recommendation policy needs to choose from a *combinatorial action space* $\binom{\mathcal{I}}{k}$, where each action is a subset 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 cascading Q-networks to handle the combinatorial action space, and an algorithm to estimate the parameters by interacting with the GAN user model.

5.1. Cascading Q-Networks

We will use the Q-learning framework where an optimal action-value function $Q^*(\boldsymbol{s}, \mathcal{A})$ will be learned and satisfies $Q^*(\boldsymbol{s}^t, \mathcal{A}^t) = \mathbb{E}\big[r(\boldsymbol{s}^t, \mathcal{A}^t, a^t) + \gamma \max_{\mathcal{A}' \subset \mathcal{I}} Q^*(\boldsymbol{s}^{t+1}, \mathcal{A}')\big]$, $a^t \in \mathcal{A}^t$. Once the action-value function is learned, an optimal policy for recommendation can be obtained as

$$\pi^*(\boldsymbol{s}^t, \mathcal{I}^t) = \arg\max_{\mathcal{A}^t \subset \mathcal{I}^t} Q^*(\boldsymbol{s}^t, \mathcal{A}^t), \qquad (7)$$

where $\mathcal{I}^t \subset \mathcal{I}$ is the set of items available at time t. The action space contains $\binom{K}{k}$ many choices, which can be very large even for moderate K (e.g. 1,000) and k (e.g. 5). Furthermore, an item put in different combinations can have different probabilities 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 a user's attention. Thus, the policy in Eq. (7) 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. (7).

Denote the recommender actions as $\mathcal{A} = \{a_{1:k}\} \subset \mathcal{I}$ and the optimal action as $\mathcal{A}^* = \{a^*_{1:k}\} = \arg\max_{\mathcal{A}} Q^*(\boldsymbol{s}, \mathcal{A})$. Our cascading Q-networks are inspired by the key fact:

$$\max_{a_{1:k}} Q^*(\boldsymbol{s}, a_{1:k}) = \max_{a_1} \big(\max_{a_{2:k}} Q^*(\boldsymbol{s}, a_{1:k}) \big). \qquad (8)$$

Also, there is a set of mutually consistent $Q^{1*}, \ldots, Q^{k*}$:

Cascading Q-Networks:

$$a^*_1 = \arg\max_{a_1} \{Q^{1*}(\boldsymbol{s}, a_1) := \max_{a_{2:k}} Q^*(\boldsymbol{s}, a_{1:k})\},$$
$$a^*_2 = \arg\max_{a_2} \{Q^{2*}(\boldsymbol{s}, a^*_1, a_2) := \max_{a_{3:k}} Q^*(\boldsymbol{s}, a_{1:k})\},$$
$$\cdots$$
$$a^*_k = \arg\max_{a_k} \{Q^{k*}(\boldsymbol{s}, a^*_{1:k-1}, a_k) := Q^*(\boldsymbol{s}, a_{1:k})\}.$$

Thus, we can obtain an optimal action in $O(k|\mathcal{I}|)$ computations by applying these functions in a cascading manner. See Algorithm 1 and Figure 2(c) for a summary. However, this cascade of Q^{j*} functions are usually not available and need to be estimated from the data.

5.2. Parameterization and Estimation

Each Q^{j*} function is parameterized by a neural network

$$\boldsymbol{q}_j^\top \sigma\Big(\boldsymbol{L}_j \big[\boldsymbol{s}^\top, \boldsymbol{f}_{a^*_1}^\top, \ldots, \boldsymbol{f}_{a^*_{j-1}}^\top, \boldsymbol{f}_{a_j}^\top\big]^\top + \boldsymbol{c}_j\Big), \ \forall j, \qquad (9)$$

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, and we use the same embedding for the state $\boldsymbol{s}$ as in Eq. (4). Now the problem left is how we can estimate these functions. Note that the set of Q^{j*} functions need to satisfy a large set of constraints. At the optimal point, the value of Q^{j*} is the same as Q^* for all j, i.e.,

$$Q^{j*}(\boldsymbol{s}, a^*_1, \cdots, a^*_j) = Q^*(\boldsymbol{s}, a^*_1, \cdots, a^*_k), \quad \forall j. \qquad (10)$$

It is not easy to strictly enforce these constraints, we take them into account in a soft and approximate way. That is, we define the loss as

$$\big(y - Q^j\big)^2, \text{ where}$$
$$y = r(\boldsymbol{s}^t, \mathcal{A}^t, a^t) + \gamma Q^k(\boldsymbol{s}^{t+1}, a^*_{1:k}; \Theta_k), \ \forall j. \qquad (11)$$

All Q^j networks are fitting against the same target y. Then the parameters Θ_k can be updated by performing gradient steps over the above loss. We note that in our experiments

Algorithm 1 Recommend using Q^j Cascades

Let $\mathcal{A}^* = \emptyset$ be empty, remove clicked items $\mathcal{I} = \mathcal{A} \setminus \boldsymbol{s}$
For $j = 1$ *to* k **do**
  $a_j^* = \arg\max_{a_j \in \mathcal{I} \setminus \mathcal{A}^*} Q^j(\boldsymbol{s}, a_{1:j-1}^*, a_j; \Theta_j)$
  Update $\mathcal{A}^* = \mathcal{A}^* \cup \{a_j^*\}$
return $\mathcal{A}^* = (a_1^*, \cdots, a_k^*)$

the set of learned Q^j networks satisfies the constraints nicely with a small error (Figure 5).

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. Besides, both the experience replay (Mnih et al., 2013) and ε-exploration techniques are applied.

6. Experiments

We conduct three sets of experiments to evaluate our generative adversarial user model (called GAN user model) and the resulting RL recommendation policy. Our experiments are designed to investigate the following questions: **(1)** Can GAN user model lead to better user behavior prediction? **(2)** Can GAN user model lead to higher user reward and click rate? and **(3)** Can GAN user model help reduce the sample complexity of reinforcement learning?

Dataset and Feature Description. We use 6 real-world datasets: **(1) MovieLens** contains a large number of movie ratings, from which we randomly sample 1,000 active users. Each display set is simulated by collecting 39 movies released near the time the movie is rated. Movie features are collected from IMDB. Categorical and descriptive features are encoded as sparse and dense vectors respectively; **(2) Last.fm** contains listening records from 359,347 users. Each display set is simulated by collecting 9 songs with the nearest time-stamp. **(3) Yelp** contains users' reviews to various businesses. Each display set is simulated by collecting 9 businesses with the nearest location. **(4) Taobao** contains the clicking and buying records of users in 22 days. We consider the buying records as positive events. **(5) RecSys15 YooChoose** contains click-streams that sometimes end with purchase events. **(6) Ant Financial News dataset** contains clicks records from 50,000 users for one month, involving dozens of thousands of news. On average each display set contains 5 news articles. It also contains user-item cross features which are widely used in this online platform. (More details in Appendix C)

6.1. Predictive Performance of User Model

To assess the predictive accuracy of GAN user model with position weight (GAN-PW) and LSTM (GAN-LSTM), we choose a series of most widely used or state-of-the-arts as the baselines, including: (1) W&D-LR (Cheng et al., 2016), a wide & deep model with logistic regression loss function; (2) CCF (Yang et al., 2011), an advanced collaborative filtering model which takes into account the context information in the loss function; we further augment it with wide & deep feature layer (W&D-CCF); (3) IKNN (Hidasi et al., 2015), one of the most popular item-to-item solutions, which calculates items similarly according to the number of co-occurrences in sessions; (4) S-RNN (Hidasi et al., 2016), a session-based RNN model with a pairwise ranking loss; (5) SCKNNC (Jannach & Ludewig, 2017), a strong methods which unify session based RNN and KNN by cascading combination; (6) XGBOOST (Chen & Guestrin, 2016), a parallel tree boosting; (7) DFM (Guo et al., 2017) is a deep neural factorization-machine based on wide & deep features. S-RNN (Hidasi et al., 2016), a session-based RNN model with a pairwise ranking loss; (8) SCKNNW (Jannach & Ludewig, 2017) and (9) SCKNNC (Jannach & Ludewig, 2017), two methods which unify S-RNN and CKNN by weighted combination and cascading combination respectively; (10) XGBOOST (Chen & Guestrin, 2016), a parallel tree boosting, which is also known as GBDT and GBM.

Top-k precision (Prec@k) is employed as the evaluation metric. It is the proportion of top-k ranked items at each page view that are actually clicked by the user, averaged across test page views and users. Users are randomly divided into train(50%), validation(12.5%) and test(37.5%) sets. The results in Table 1 show that GAN model performs significantly better than baselines. Moreover, GAN-PW performs nearly as well as GAN-LSTM, but it is more efficient to train. *Thus we use GAN-PW for later experiments and refer to it as GAN.*

We also tested different types of regularization (Table 2). In general, Shannon entropy performs well and it is also favored for its closed form solution. However, on the Yelp dataset, we find that L_2 regularization $R(\phi) = \|\phi\|_2^2$ leads to a better user model. It is noteworthy that the user model with L_2 regularization is trained with Shannon entropy initialization scheme proposed in section 4.3.

An interesting result on Movielens is shown in Figure 3 (see Appendix D.1 for similar figures). It shows GAN performs much better as time goes by, while the items predicted by W&D-CCF are concentrated on several categories. This indicates a drawback of static models — it fails to capture user interest evolution.

6.2. Recommendation Policy Based on User Model

With a learned user model, we can immediately derive a greedy policy to recommend k items with the highest estimated likelihood. We will compare the strongest baseline methods **W&D-LR, W&D-CCF** and **GAN-Greedy** in this setting. Furthermore, we will learn an RL policy using the cascading Q-networks from section 5 (**GAN-**

Table 1. Comparison of predictive performances, where we use Shannon entropy for GAN-PW and GAN-LSTM.

	(1) MovieLens		(2) LastFM		(3) Yelp		(4) Taobao		(5) YooChoose		(6) Ant Financial	
Model	prec(%)@1	prec(%)@2	prec(%)@1	prec(%)@2	prec(%)@1	prec(%)@2	prec(%)@1	prec(%)@2	prec(%)@1	prec(%)@2	prec(%)@1	prec(%)@2
IKNN	38.8(±1.9)	40.3(±1.9)	20.4(±0.6)	32.5(±1.4)	57.7(±1.8)	73.5(±1.8)	32.8(±2.6)	46.6(±2.6)	39.3(±1.5)	69.8(±2.1)	20.6(±0.2)	32.1(±0.2)
S-RNN	39.3(±2.7)	42.9(±3.6)	9.4(±1.6)	17.4(±0.9)	67.8(±1.4)	73.2(±0.9)	32.7(±1.7)	47.0(±1.4)	41.8(±1.2)	69.9(±1.9)	32.2(±0.9)	40.3(±0.6)
SCKNNC	49.4(±1.9)	51.8(±2.3)	21.4(±0.5)	26.1(±1.0)	60.3(±4.5)	71.6(±1.8)	35.7(±0.4)	47.9(±2.1)	40.8(±2.5)	70.4(±3.8)	34.6(±0.7)	43.2(±0.8)
XGBOOST	66.7(±1.1)	76.0(±0.9)	10.2(±2.6)	19.2(±3.1)	64.1(±2.1)	79.6(±2.4)	30.2(±2.5)	51.3(±2.6)	60.8(±0.4)	80.3(±0.4)	41.9(±0.1)	65.4(±0.2)
DFM	63.3(±0.4)	75.9(±0.3)	10.5(±0.4)	20.4(±0.1)	72.1(±2.1)	80.3(±2.1)	30.1(±0.8)	48.5(±1.1)	**61.3**(±0.3)	**82.5**(±1.5)	41.7(±0.1)	64.2(±0.2)
W&D-LR	61.5(±0.7)	73.8(±1.2)	7.6(±2.9)	16.6(±3.3)	62.7(±0.8)	86.0(±0.9)	34.0(±1.1)	54.6(±1.5)	51.9(±0.8)	75.8(±1.5)	37.5(±0.2)	60.9(±0.1)
W&D-CCF	65.7(±0.8)	75.2(±1.1)	15.4(±2.4)	25.7(±2.6)	**73.2**(±1.8)	88.1(±2.2)	34.9(±1.1)	53.3(±1.3)	52.1(±0.5)	76.3(±1.5)	37.7(±0.1)	61.1(±0.1)
GAN-PW	66.6(±0.7)	75.4(±1.3)	**24.1**(±0.8)	**34.9**(±0.7)	72.0(±0.2)	**92.5**(±0.5)	34.7(±0.6)	54.1(±0.7)	52.9(±0.7)	75.7(±1.4)	41.9(±0.1)	65.8(±0.1)
GAN-LSTM	**67.4**(±0.5)	**76.3**(±1.2)	24.0(±0.9)	34.9(±0.8)	73.0(±0.2)	88.7(±0.4)	**35.9**(±0.6)	**55.0**(±0.7)	52.7(±0.3)	75.9(±1.2)	**42.1**(±0.2)	**65.9**(±0.2)

Table 2. GAN-LSTM user model with SE (Shannon entropy) versus L_2 regularization on Yelp dataset. pr is the short for prec(%).

	Split 1		Split 2		Split 3	
Model	pr@1	pr@2	pr@1	pr@2	pr@1	pr@2
SE	73.1	88.8	72.8	89.0	73.1	88.2
L_2	**73.5**	**89.0**	**78.8**	**91.5**	**76.1**	**91.1**

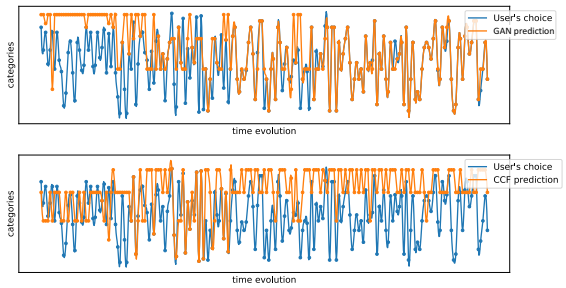

Figure 3. Comparison of the true trajectory (blue) of a user's choices, the simulated trajectory predicted by GAN model (orange curve in upper sub-figure) and the simulated trajectory predicted by W&D-CCF (the orange curve in the lower sub-figure). Y-axis represents 80 categories of movies. Each data point (t, c) represents time step t and the category c of the clicked item.

CDQN) and compare it with two RL methods: a cascading Q-network trained with ± 1 reward (**GAN-RWD1**), and an additive Q-network policy (He et al., 2016), $Q(\boldsymbol{s}, a_1, \cdots, a_k) := \sum_{j=1}^{k} Q(\boldsymbol{s}, a_j)$ trained with learned reward (**GAN-GDQN**).

Since we cannot perform online experiments at this moment, we use collected data from the online news platform to fit a user model, and then use it as a test environment. To make the experimental results trustful and solid, we fit the test model based on a randomly sampled test set of 1,000 users and keep this set isolated. The RL policies are learned from another set of 2,500 users without overlapping the test set. The performances are evaluated by two metrics: (1) **Cumulative reward**: For each recommendation action, we can observe a user's behavior and compute her reward $r(\boldsymbol{s}^t, a^t)$ using the test model. Note that we never use the reward of test users when we train the RL policy. The numbers shown in Table 3 are the cumulative rewards averaged over time horizon first and then averaged over all users. It can be formulated as $\frac{1}{N}\sum_{u=1}^{N}\frac{1}{T}\sum_{t=1}^{T} r_u^t$, where r_u^t is the reward received by user u at time t. (2) **CTR (click through rate)**: it is the ratio of the number of clicks and the number of steps it is run. The values displayed in Table 3 are also averaged over 1,000 test users.

Table 3. Comparison of recommendation performance.

	$k = 3$		$k = 5$	
model	reward	ctr	reward	ctr
W&D-LR	14.46(±0.42)	0.46(±0.01)	15.18(±0.38)	0.48(±0.01)
W&D-CCF	19.93(±1.09)	0.62(±0.03)	20.94(±1.03)	0.65(±0.03)
GAN-Greedy	21.37(±1.24)	0.67(±0.04)	22.97(±1.22)	0.71(±0.03)
GAN-RWD1	22.17(±1.07)	0.68(±0.03)	25.15(±1.04)	**0.78**(±0.03)
GAN-GDQN	23.60(±1.06)	0.72(±0.03)	23.19(±1.17)	0.70(±0.03)
GAN-CDQN	**24.05**(±0.98)	**0.74**(±0.03)	**25.36**(±1.10)	0.77(±0.03)
DQN-Off	20.31(±0.14)	0.63(±0.01)	21.82(±0.08)	0.67(±0.01)

Experiments with different numbers of items in each page view are conducted and the results are summarized in Table 3. Since users' behaviors are not deterministic, each policy is evaluated repeatedly for 50 times on test users. The results show that: (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 GAN-CDQN is trained to optimize the cumulative reward, the recommendation policy also achieves a higher CTR compared to GAN-RWD1 which directly optimizes ± 1 reward. The learning of GAN-CDQN may have benefited from the well-known reward shaping effects of the learned continuous reward (Mataric, 1994; Ng et al., 1999; Matignon et al., 2006). (4) While the computational cost of GAN-CDQN is about the same as that of GAN-GDQN (both are linear in the total number of items), our proposed GAN-CDQN is a more flexible parametrization and achieved better results.

Since Table 3 only shows average values taken over test users, we compare the policies in user level and the results are shown in figure 4. GAN-CDQN policy results in higher averaged cumulative reward for most users. A similar figure which compares the CTR is deferred to Appendix D. Figure 5 shows that the learned cascading Q-networks satisfy constraints in Eq. (10) well when $k = 5$.

6.3. User Model Assisted Policy Adaptation

Former results in section 6.1 and 6.2 have demonstrated that GAN is a better user model and RL policy based on it can achieve higher CTR compared to other user models, but this user model may be misspecified. In this section, we show that our GAN model can help an RL policy to quickly adapt to a new user. The RL policy assisted by GAN user model is compared with other policies that are learned from and

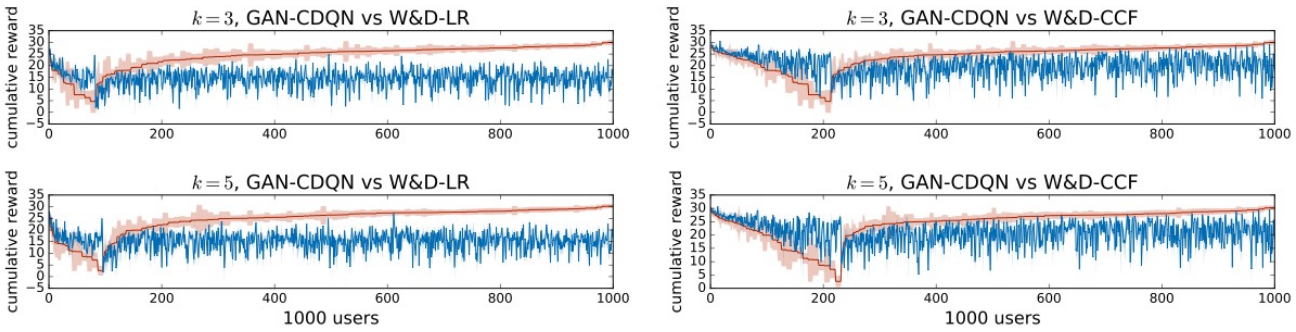

Figure 4. Cumulative rewards among 1,000 users under the recommendation policies based on different user models. The experiments are repeated for 50 times and the standard deviation is plotted as the shaded area.

adapted to online users: (1) **CDQN with GAN**: cascading Q-networks which are first trained using the learned GAN user model from other users and then adapted online to a new user using MAML (Finn et al., 2017). (2) **CDQN model free**: cascading Q-networks 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 6.2. All policies are evaluated on a set of 1,000 test users associated with a test model. Two 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 users over time. In fact, the performances of users' cumulative reward according to different policies are also similar, and the corresponding figure is deferred to Appendix D.3.

The results show that the CDQN policy pre-trained over a GAN user model can quickly achieve a high CTR even when it is applied to a new set of users (Figure 6). 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' interests during its interaction with users. 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 CTR. Generally speaking, GAN user model provides a dynamical environment for RL policies to interact with. It helps the policy achieve a more satisfying status before applying to online users.

7. Conclusion and Future Work

We proposed a novel model-based reinforcement learning framework for recommendation systems, where we developed a GAN formulation to model user behavior dynamics and her associated reward function. Using this user model as the simulation environment, we develop a novel cascading Q-network for combinatorial recommendation policy which can handle a large number of candidate items efficiently. Although the experiments show clear benefits of our method in an offline and realistic simulation setting, even stronger results could be obtained via future online A/B testing.

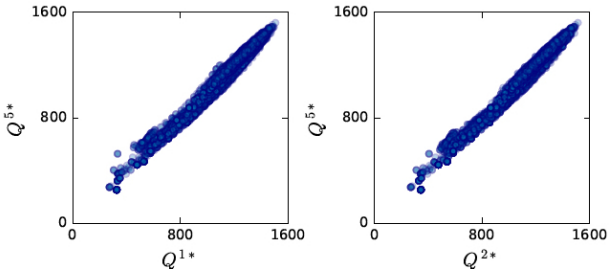

Figure 5. Each scatter-plot compares Q^{j^*} with Q^{5*} values in Eq. (10) evaluated at the same set of k recommended items. In the ideal case, all scattered points should lie along the diagonal.

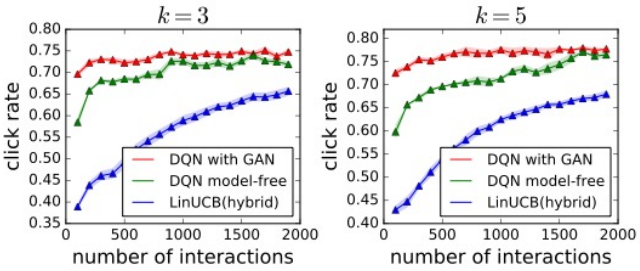

Figure 6. Comparison of the averaged click rate averaged over 1,000 users under different recommendation policies. X-axis represents how many times the recommender interacts with online users. Y-axis is the click rate. Each point (x, y) means the click rate y is achieved after x times of user interactions.

Acknowledgement

This project was supported in part by NSF IIS-1218749, NIH BIGDATA 1R01GM108341, NSF CAREER IIS-1350983, NSF IIS-1639792 EAGER, NSF IIS-1841351 EAGER, NSF CNS-1704701, ONR N00014-15-1-2340, Intel ISTC, NVIDIA, Google and Amazon AWS.

References

Abbeel, P. and Ng, A. Y. Apprenticeship learning via inverse reinforcement learning. In *ICML*, 2004.

Chen, T. and Guestrin, C. Xgboost: A scalable tree boosting system. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 785–794. ACM, 2016.

Cheng, H.-T., Koc, L., Harmsen, J., Shaked, T., Chandra, T., Aradhye, H., Anderson, G., Corrado, G., Chai, W., Ispir, M., Anil, R., Haque, Z., Hong, L., Jain, V., Liu, X., and Shah, H. Wide & deep learning for recommender systems. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*. ACM, 2016.

Clavera, I., Nagabandi, A., Fearing, R. S., Abbeel, P., Levine, S., and Finn, C. Learning to adapt: Meta-learning for model-based control. *arXiv preprint arXiv:1803.11347*, 2018.

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

Finn, C., Abbeel, P., and Levine, S. Model-agnostic meta-learning for fast adaptation of deep networks. *arXiv preprint arXiv:1703.03400*, 2017.

Guo, H., Tang, R., Ye, Y., Li, Z., and He, X. Deepfm: a factorization-machine based neural network for ctr prediction. *arXiv preprint arXiv:1703.04247*, 2017.

He, J., Ostendorf, M., He, X., Chen, J., Gao, J., Li, L., and Deng, L. Deep reinforcement learning with a combinatorial action space for predicting popular reddit threads. In *EMNLP*, 2016.

Hidasi, B., Karatzoglou, A., Baltrunas, L., and Tikk, D. Session-based recommendations with recurrent neural networks. *arXiv preprint arXiv:1511.06939*, 2015.

Hidasi, B., Karatzoglou, A., Baltrunas, L., and Tikk, D. Session-based recommendations with recurrent neural networks. In *ICLR*, 2016.

Ho, J. and Ermon, S. Generative adversarial imitation learning. In *NIPS*, 2016.

Ho, J., Gupta, J. K., and Ermon, S. Model-free imitation learning with policy optimization. In *ICML*, 2016.

Jannach, D. and Ludewig, M. When recurrent neural networks meet the neighborhood for session-based recommendation. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*, pp. 306–310. ACM, 2017.

Li, L., Chu, W., Langford, J., and Schapire, R. E. 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.

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

Mataric, M. J. Reward functions for accelerated learning. In *ICML*, 1994.

Matignon, L., Laurent, G. J., and Fort-Piat, N. L. Reward function and initial values: Better choices for accelerated goal-directed reinforcement learning. In *ICANN*, 2006.

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

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Nagabandi, A., Kahn, G., Fearing, R. S., and Levine, S. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. *arXiv preprint arXiv:1708.02596*, 2017.

Ng, A. Y., Harada, D., and Russell, S. J. Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, 1999.

Torabi, F., Warnell, G., and Stone, P. Behavioral cloning from observation. In *IJCAI*, 2018.

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

Yang, S.-H., Long, B., Smola, A. J., Zha, H., and Zheng, Z. 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.

Zhao, X., Xia, L., Zhang, L., Ding, Z., Yin, D., and Tang, J. Deep reinforcement learning for page-wise recommendations. 2018a.

Zhao, X., Zhang, L., Ding, Z., Yin, D., Zhao, Y., and Tang, J. Deep reinforcement learning for list-wise recommendations. *CoRR*, 2018b.

Zheng, G., Zhang, F., Zheng, Z., Xiang, Y., Yuan, N. J., Xie, X., and Li, Z. Drn: A deep reinforcement learning framework for news recommendation. 2018.