Off-policy Learning in Two-stage Recommender Systems

Jiaqi Ma*
jiaqima@umich.edu
University of Michigan, Ann Arbor

Zhe Zhao
zhezhao@google.com
Google AI

Xinyang Yi
xinyang@google.com
Google AI

Ji Yang
jiyangjy@google.com
Google AI

Minmin Chen
minminc@google.com
Google AI

Jiaxi Tang*
jiaxit@sfu.ca
Simon Fraser University

Lichan Hong
lichan@google.com
Google AI

Ed H. Chi
edchi@google.com
Google AI

ABSTRACT

Many real-world recommender systems need to be highly scalable: matching millions of items with billions of users, with milliseconds latency. The scalability requirement has led to widely used *two-stage* recommender systems, consisting of efficient *candidate generation* model(s) in the first stage and a more powerful *ranking* model in the second stage.

Logged user feedback, e.g., user clicks or dwell time, are often used to build both candidate generation and ranking models for recommender systems. While it's easy to collect large amount of such data, they are inherently biased because the feedback can only be observed on items recommended by the previous systems. Recently, off-policy correction on such biases have attracted increasing interest in the field of recommender system research. However, most existing work either assumed that the recommender system is a single-stage system or only studied how to apply off-policy correction to the candidate generation stage of the system without explicitly considering the interactions between the two stages.

In this work, we propose a two-stage off-policy policy gradient method, and showcase that ignoring the interaction between the two stages leads to a sub-optimal policy in two-stage recommender systems. The proposed method explicitly takes into account the ranking model when training the candidate generation model, which helps improve the performance of the whole system. We conduct experiments on real-world datasets with large item space and demonstrate the effectiveness of our proposed method.

CCS CONCEPTS

• **Information systems → Recommender systems**; • **Computing methodologies → Neural networks**; • **Theory of computation → Reinforcement learning**.

*Work done while interning at Google.

KEYWORDS

Recommender Systems, Off-policy Learning, Two-stage Systems, Neural Networks

ACM Reference Format:
Jiaqi Ma, Zhe Zhao, Xinyang Yi, Ji Yang, Minmin Chen, Jiaxi Tang, Lichan Hong, and Ed H. Chi. 2020. Off-policy Learning in Two-stage Recommender Systems. In *Proceedings of The Web Conference 2020 (WWW '20), April 20–24, 2020, Taipei, Taiwan.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3366423.3380130

1 INTRODUCTION

With the explosive growth of online content, recommender systems are heavily relied on to help users quickly discover the small fraction of content matching their interests. Industrial recommender systems are tasked with connecting billions of users to millions or billions of items, and delivering recommendations within milliseconds. To be able to serve user highly personalized content within the latency requirement, a *two-stage* approach (see Figure 1) is widely used [4, 9, 12]. In the first stage, one or multiple efficient *candidate generation* model(s) are used to produce a candidate set that contains hundreds or thousands of items from the whole item space. In the second stage, a more powerful *ranking* model re-ranks the candidate items and recommends the top few items to the user. The candidate generation models and the ranking model are often trained independently with logged implicit feedback data (e.g., user clicks or dwell time) generated by previous versions of the recommender system. However, such data are inherently biased because feedback can only be observed on items recommended by the previous systems, i.e., different from the standard supervised learning setup, we only have partial label information. Training new recommender systems with the biased data may lead to the "rich gets richer" problem [13]. In this work, we investigate how to correct such biases with off-policy learning under the practical two-stage recommender system setting.

Off-policy learning mitigates the bias caused by observing only partial feedback in interactive systems. It corrects the biases caused by the discrepancy between the system being trained, which is called *target policy*, and the system that generated the logged implicit feedback data, which is called *behavior policy*. There has been

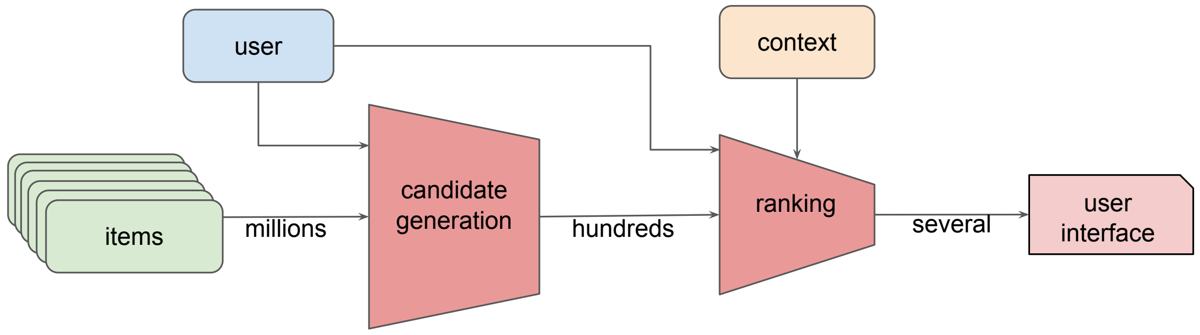


Figure 1: A two-stage recommender system that serves users highly personalized content within the strict latency. It consists of an efficient candidate generation model in the first stage and a more powerful ranking model in the second stage.

a large body of literature studying off-policy learning (a.k.a. off-policy correction) in both contextual bandit [11, 18] and reinforcement learning settings [24, 37]. One of the most common off-policy correction methods is Inverse Propensity Scoring (IPS) [15]. IPS assigns per-example importance weight, which is the probability ratio between the target policy and the behavior policy, to the empirical training objective over the logged data, so that the corrected objective is provably unbiased. Off-policy learning has been successfully applied to various applications including search engines [1, 18] and online advertisements [30].

Off-policy correction has attracted increasing interest in recommender system research in recent years [7, 14, 33]. Most existing works either assumed that the recommender system is single-stage [33], or applied off-policy correction to the candidate generation model (of the two-stage recommender system) without explicitly considering the ranking model [7]. Ignoring the interaction between the two stages causes sample inefficiency when learning the candidate generation policy. In the worse case, it introduces bias and results in learning a sub-optimal whole-system policy (when the ranking policy in the previous system that generated the training data is different from the ranking policy to be used in the current system).

To fill in the gap, we propose an efficient two-stage off-policy policy gradient method that explicitly takes the ranker model into consideration while training the candidate generation model. We first formulate the target policy of the two-stage recommender system as the composition of candidate generation policy and the ranker policy. We then derive the importance weight of the IPS-based off-policy correction for the candidate generation model. Unfortunately, the exact calculation of the importance weight of a sample in the logged data has a prohibitively large time complexity of $O(m^k)$, where m is the size of the item space and k is the candidate set size. To make the problem tractable, we propose an efficient Monte Carlo approximation algorithm by making a mild assumption about the candidate generation model.

Finally, we conduct experiments on MovieLens and Wiki10-31K, an extreme multi-label classification dataset, where we can generate semi-synthetic online data using simulators or supervised-to-bandit conversion. Empirical results show that our proposed method improves the performance of the two-stage system significantly.

2 RELATED WORK

2.1 Two-stage Recommender Systems

Two-stage recommender systems with candidate generation followed by ranking has been widely adopted in industry, including *YouTube* [9, 40, 42], *Linkedin* [4], *Pinterest* [12]. The two-stage setup makes it possible to recommend highly personalized items from a huge item space in real-time. There is still a lot of ongoing effort to improve both the efficiency [19, 40] and the recommendation quality [7, 42] following this general approach. This work mainly focuses on the latter goal. Perhaps the closest work to ours is Chen et al. [7], which shares with this work the goal of improving quality of the candidate generation model by correcting the system biases in the logged implicit feedback data. But we emphasize more on the optimization of the performance of the entire system by taking into account the ranking stage explicitly.

2.2 Reinforcement Learning

This work formulates the recommendation problem as a reinforcement learning problem. There are two major families of model-free reinforcement learning approaches, namely value-based approaches and policy-based approaches. Value-based approaches such as Q-learning [23, 28], typically use a function approximator (e.g., a neural network) to represent the state-action value function, and then derive the optimal policy by choosing the action that maximizes the value function. The policy-based approaches such as policy gradients [21, 27], instead parameterise the policy directly and learn the policy to maximize expected long-term reward. As summarized by Chen et al. [7], value-based approaches suffer from instability with function approximations [31] compared with policy-based approaches, and extensive hyper-parameter search is often required for value-based approaches to achieve stable behavior. We therefore also focus on the policy-based approach for our recommendation problem.

2.3 Bandit Learning in Web Applications

This work is also related to the research of contextual bandits [11, 20, 38], a special case of reinforcement learning with one-step decision making. Contextual bandits have been successfully used for modeling a lot of web applications such as personalized advertisement [5, 36], learning-to-rank [29], and personalized recommendations [22, 25, 35]. Many early works focus on the online learning problem using well-known bandit algorithms, including variants of the Upper Confidence Bounds [3, 8, 22] and Thompson Sampling [2, 6] approaches. These methods trade off exploration and exploitation as they receive feedback on actions to quickly converge to a policy with minimum regrets. Recent works study the off-policy learning problem in these settings [11, 32] with logged implicit feedback data.

2.4 Off-policy Learning

Off-policy learning, referring to the setup of policy optimization with access only to logged (batch) feedback data collected by a different behavior policy, has been a popular research topic in both reinforcement learning and contextual bandits [11, 17, 24, 26]. Unlike games, e.g. Atari or Go, for which we can easily build simulation environments for on-policy learning, interactively gathering data by running the target policy for many real-world applications is expensive, risky, unethical, or even illegal [17]. For instance, in the marketing domain, running a decision making policy that has not been carefully evaluated could be risky and thus expensive. For online web services, learning a policy in an online fashion could cause both high computational cost and unreliable service quality. In these scenarios, off-policy learning offers a general recipe for policy optimization using historical data without interactive access to a simulation environment. This is desirable for web service applications such as recommender systems or search engines, where large amount of logged implicit feedback data are available, and therefore has attracted an increasing research interest in these areas [7, 22, 32, 33]. While most previous works assumed the target policy is a single end-to-end agent, this work tackles the problem of off-policy learning in two-stage recommender systems, which has not been well studied.

3 RECOMMENDATION AS A REINFORCEMENT LEARNING PROBLEM

We begin with describing the setup of training the recommender system as a reinforcement learning problem.

We first introduce the following notations of user-state and item spaces for recommendation problems:

- $\mathcal{S}$: a user-state space with each state describing a user accompanied with their contextual status, e.g., the time when the recommendation is made, or the query text given by the user;
- $\mathcal{A}$: the set of all possible items;

Different users will surely have different states and the state of one user evolves as he/she interacts with the recommender.

In the language of reinforcement learning, training a recommender system means that we are seeking a policy $\pi(a|s)$, which is a distribution over the items $a \in \mathcal{A}$ conditioned on the user state $s \in \mathcal{S}$. Such a policy aims to maximize the expected discounted cumulative reward over potentially infinite time horizon T,

$$V(\pi) = \mathbb{E}_{s_0 \sim \rho(s), a_t \sim \pi(a|s_t), s_{t+1} \sim P(s|s_t, a_t)} \Big[\sum_{t=0}^{T} \gamma^t r(s_t, a_t) \Big],$$

where $\rho(s)$ is the initial distribution of user states, $P(s|s_t, a_t)$ is the state transition probability, $\gamma \in [0, 1)$ is a discounting factor, and

$$r(s, a) \in R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$$

is the immediate reward obtained by performing an action a at the user state s.

In this work, we assume the user-state distribution of the recommender system does not change over time. This leads to the special case of one-step reinforcement learning (a.k.a. contextual bandit) problem with $T = 1$, which seeks a policy $\pi(a|s)$ optimizing

$$V(\pi) = \mathbb{E}_{s \sim \rho(s), a \sim \pi(a|s)} \Big[r(s, a) \Big]. \quad (1)$$

For simplicity, we denote $\mathbb{E}_{s \sim \rho(s), a \sim \pi(a|s)}$ by $\mathbb{E}_\pi$ in the rest of the paper.

There have been plenty of algorithms available to optimize such reinforcement learning problems, including value-based methods [28] and policy-based methods [21, 27], as summarized in Section 2.2. In this work, we focus on the policy-gradient-based approach.

Assuming that the policy takes a function form π_θ parameterized by $\theta \in \mathbb{R}^d$, the gradient of the value function w.r.t. θ can be expressed as the following REINFORCE [39] gradient thanks to the *log-trick*,

$$\nabla_\theta V(\pi_\theta) = \mathbb{E}_{\pi_\theta} \Big[r(s, a) \nabla_\theta \log \pi_\theta(a|s) \Big]. \quad (2)$$

3.1 Off-policy Learning with Inverse Propensity Scoring

Due to safety and infrastructure limitations, modern recommender systems are usually trained offline with the logged user feedback data [7, 9, 40, 42]. Here we apply off-policy learning methods to correct the biases caused by the fact that the logged data is generated by a historical *behavior policy* β, different from the *target policy* π_θ being trained.

One of the most widely used off-policy correction methods is Inverse Propensity Scoring (IPS) [7, 15, 24, 26]. One can easily shows that

$$V(\pi_\theta) = \mathbb{E}_{\pi_\theta} \Big[r(s, a) \Big] = \mathbb{E}_\beta \Big[\frac{\pi_\theta(a|s)}{\beta(a|s)} r(s, a) \Big],$$

where the second equality is due to importance weighting and $\frac{\pi_\theta(a|s)}{\beta(a|s)}$ is called an *importance weight*. Then the policy gradient in Eq. (2) can be re-written as

$$\nabla_\theta V(\pi_\theta) = \mathbb{E}_\beta \Big[\frac{\pi_\theta(a|s)}{\beta(a|s)} r(s, a) \nabla_\theta \log \pi_\theta(a|s) \Big].$$

Suppose we have a logged dataset $\mathcal{D} = \{(s_i, a_i, r_i)\}_{i=1}^n$ generated by β, where $s_i \in \mathcal{S}, a_i \in \mathcal{A}$, and $r_i \in \mathbb{R}$ is the observed reward for (s_i, a_i), for $i = 1, ..., n$. Then we can empirically estimate the off-policy policy gradient as

$$\nabla_\theta \hat{V}(\pi_\theta) = \frac{1}{n} \sum_{i=1}^{n} \frac{\pi_\theta(a_i|s_i)}{\beta(a_i|s_i)} r_i \nabla_\theta \log \pi_\theta(a_i|s_i). \quad (3)$$

Table 1: Notations.

Notation	Description
$\mathcal{S}$	user-state space
$\mathcal{A}$	item space
s	user state
a	item
$r(s,a)$	reward function
A^k	candidate set with size k
$\pi(a\|s)$	target recommender policy
$\beta(a\|s)$	behavior recommender policy
$p(A^k\|s)$	candidate generation model
$q(a\|s,A^k)$	ranking model
$V(\pi)$	value function
$\mathcal{D}$	logged dataset
(s_i,a_i,r_i)	tuple of a logged sample

Most existing work focused on the scenario where π_θ is a single-stage model parameterized by a differentiable function where the gradients are easy to calculate, e.g., a neural network [7]. In the practical case of two-stage recommender systems, however, π_θ either is not differentiable or requires intractable computation cost to calculate the gradients. Next, we will give the general formulation of a two-stage recommender system and we list the most important notations in Table 1.

4 OFF-POLICY LEARNING IN TWO-STAGE RECOMMENDER SYSTEMS

4.1 Two-stage Recommender Systems

A typical two-stage recommender system [9, 42] is comprised of one or multiple candidate generation models and a ranking model. To simplify analysis, here we only consider the case of top-1 recommendation, i.e., the ranking model will deliver one final recommendation to the user and only a single candidate generation model is employed. We also assume both the candidate generation model and the ranking model produce stochastic policies [7].

Let $A^k \subseteq \mathcal{A}$ denote a set of items with size k. Given a user state $s \in \mathcal{S}$, the candidate generation model can be represented as a probability over all possible candidate sets A^k conditioned on the user state s: $p(A^k|s)$; and the ranking model can be represented as a probability over all items a conditioned on the user state s and a candidate set A^k: $q(a|s,A^k)$. In practice, the final recommendation given by the ranker always comes from the candidate set, i.e.,

$$q(a|s,A^k) = 0, \text{ if } a \notin A^k. \tag{4}$$

The target policy π of the two-stage recommender system can be then decomposed as

$$\pi(a|s) = \sum_{A^k} p(A^k|s)q(a|s,A^k).$$

In practice, the candidate generation model and the ranking model are usually trained separately [7, 9, 42]. Following Chen et al. [7], we focus on the off-policy correction of the candidate generation model in this work. Specifically, we assume that the ranking model has been well-trained and remains the same during the training and serving of the candidate generation model, which is a reasonable assumption in most industrial practices nowadays [9]. [1]

We have seen that the target policy π of the whole system, which is what we care about most at serving, is different from the candidate generation policy p. And the degree of difference depends on the ranking model q. If p and q happen to have very different preferences on the items to recommend, applying an off-policy correction with p/β as the importance weight will lead to a sub-optimal solution. Next, we introduce the *two-stage off-policy policy gradient* of the candidate generation model given the ranking model.

4.2 Two-stage Off-policy Policy Gradient for the Candidate Generation Model

Assuming the candidate generation model takes the form $p_\theta(A^k|s)$ parameterized by $\theta \in \mathbb{R}^{d_g}$, the target policy can be written as

$$\pi_\theta(a|s) = \sum_{A^k} p_\theta(A^k|s)q(a|s,A^k). \tag{5}$$

Plugging Eq. 5 into Eq. 3, the two-stage off-policy policy gradient can then be written as

$$\begin{aligned}\nabla_\theta \hat{V}(\pi_\theta) &= \frac{1}{n}\sum_{i=1}^{n} \frac{\pi_\theta(a_i|s_i)}{\beta(a_i|s_i)} r_i \nabla_\theta \log \pi_\theta(a_i|s_i) \\ &= \frac{1}{n}\sum_{i=1}^{n} \frac{\nabla_\theta \pi_\theta(a_i|s_i)}{\beta(a_i|s_i)} r_i \\ &= \frac{1}{n}\sum_{i=1}^{n} \frac{\sum_{A^k} q(a_i|s_i,A^k)\nabla_\theta p_\theta(A^k|s_i)}{\beta(a_i|s_i)} r_i.\end{aligned}$$

The exact calculation of the policy gradient $\nabla_\theta \hat{V}(\pi_\theta)$ needs to iterate through all possible candidate sets with size k, which results in a complexity $O(|\mathcal{A}|^k)$. As $|\mathcal{A}|$ is at millions or billions level, this is not computationally feasible even for small k. Therefore, we next seek an efficient approximation of this policy gradient.

4.2.1 Unbiased Approximation by Candidate Set Sampling. First notice that the gradient of the target policy at the ith data point can be written as

$$\begin{aligned}\nabla_\theta \pi_\theta(a_i|s_i) &= \sum_{A^k} q(a_i|s_i,A^k)\nabla_\theta p_\theta(A^k|s_i) \\ &= \sum_{A^k} p_\theta(A^k|s_i)q(a_i|s_i,A^k)\nabla_\theta \log p_\theta(A^k|s_i) \\ &= \mathbb{E}_{A^k \sim p_\theta(A^k|s_i)}\left[q(a_i|s_i,A^k)\nabla_\theta \log p_\theta(A^k|s_i)\right],\end{aligned}$$

where the second equality is due to the log-trick and the third equality is by the definition of expectation.

Instead of iterating through all possible A^k, we can get an unbiased estimator of $\nabla_\theta \pi_\theta(a_i|s_i)$ by sampling $A^k \sim p_\theta(A^k|s_i)$. However, recall that the ranker policy $q(a_i|s_i,A^k)$ follows the constraint of Eq. (4) by definition. Therefore, if a sampled A^k does not contain the logged item a_i, the corresponding gradient will be zero.

[1]Note that the ranking model can change between the data collection (in behavior policy) and the training/serving of the candidate generation model (in target policy) though. Without taking into account of interactions between the two stages will lead to learning a sub-optimal candidate generation model.

We further mitigate this inefficiency problem by making a sampling with replacement assumption on the candidate set generation. Although in practice the candidate set is generated by sampling without replacement, we note that this is a reasonable approximation when the item space is large. Indeed, Chen et al. [7] also introduced this assumption in their top-k recommender.

4.2.2 Sampling with Replacement on Candidate Set Generation. Assuming the candidate set is generated by sampling with replacement, then we have

$$p_\theta(A^k|s) = \prod_{j=1}^{k} p_\theta(A^k_j|s). \tag{6}$$

Here we have reloaded the notation p_θ to denote both the joint probability of a candidate set and the marginal probability of each item in the candidate set corresponding to the same candidate generation model parameterized by θ. With this assumption, we can re-write $\nabla_\theta \pi_\theta(a_i|s_i)$ as

$$\begin{aligned}\nabla_\theta \pi_\theta(a_i|s_i) &= \sum_{A^k} p_\theta(A^k|s_i) q(a_i|s_i, A^k) \nabla_\theta \log p_\theta(A^k|s_i) \\ &= p_\theta(a_i|s_i) \sum_{A^{k-1}} \Big[p_\theta(A^{k-1}|s_i) q(a_i|s_i, \{a_i\} \cup A^{k-1}) \cdot \\ &\qquad \Big(\nabla_\theta \log p_\theta(a_i|s_i) + \nabla_\theta \log p_\theta(A^{k-1}|s_i) \Big) \Big] \\ &= p_\theta(a_i|s_i)\, \mathbb{E}_{A^{k-1}} \Big[q(a_i|s_i, \{a_i\} \cup A^{k-1}) \cdot \\ &\qquad \Big(\nabla_\theta \log p_\theta(a_i|s_i) + \nabla_\theta \log p_\theta(A^{k-1}|s_i) \Big) \Big],\end{aligned} \tag{7}$$

where $\mathbb{E}_{A^{k-1}}$ stands for $\mathbb{E}_{A^{k-1} \sim p_\theta(A^{k-1}|s_i)}$.

In this way, we can get non-zero gradient for any sampled candidate set A^{k-1}.

4.3 Variance Reduction Tricks

It is well-known [7, 32] that the off-policy policy gradient can have large variance due to large importance weights $\omega(s, a) = \frac{\pi(a|s)}{\beta(a|s)}$. Therefore we apply several common variance reduction tricks in the IPS methods, including weight capping

$$\hat{\omega}_{c_1,c_2}(s, a) = \max \Big\{ \min\{\omega(s, a), c_1\}, c_2 \Big\},$$

and self-normalized importance sampling

$$\hat{\omega}_n(s, a) = \frac{\omega(s, a)}{\frac{1}{n} \sum_{(s',a') \sim \beta} \omega(s', a')},$$

where c_1, c_2 are some constants and n is the logged dataset size.

To overcome the additional variance led by the sampling approximation (Eq. 7), we introduce a hyper-parameter $\alpha \in [0, 1)$ down-weighting the gradient of the log-probability of the sampled candidate set A^{k-1}. That is, we replace the

$$\nabla_\theta \log p_\theta(a_i|s_i) + \nabla_\theta \log p_\theta(A^{k-1}|s_i)$$

in the last line of Eq. 7 by

$$\nabla_\theta \log p_\theta(a_i|s_i) + \alpha \cdot \nabla_\theta \log p_\theta(A^{k-1}|s_i). \tag{8}$$

Algorithm 1: Approximate the two-stage IPS loss

```
Input  : The logged dataset $\mathcal{D} = \{(s_i, a_i, r_i)\}_{i=1}^n$, behavior
         policy $\beta(a|s)$, candidate set size $k$, candidate set
         sample size $T$, ranking model $q(a|s, A^k)$, candidate
         generation model $p_\theta(a|s)$ (the right-hand side
         notation of $p_\theta$ in Eq. 6), item space size $m = |\mathcal{A}|$.
Output : Approximation of the two-stage IPS loss $J_{2\text{-IPS}}(\theta)$.
$J \leftarrow 0$;
for $i = 1, \cdots, n$ do
    $J_i \leftarrow 0$;
    for $j = 1, \cdots, n$ do
        $\tilde{p}(a_j|s_i) \leftarrow$ stop_gradient($p_\theta(a_j|s_i)$);
    end
    $\omega(a_i|s_i) \leftarrow \frac{\tilde{p}(a_i|s_i)}{\beta(a_i|s_i)}$;
    for $t = 1, \cdots, \tau$ do
        $A_t \leftarrow \{a_i\}$;
        for $v = 1, \cdots, k-1$ do
            Sample $a_v \sim \tilde{p}$;
            $A_t \leftarrow A_t \cup \{a_v\}$;
        end
        $A_t \leftarrow$ unique($A_t$);
        $l_t \leftarrow 0$;
        for $a_v \in A_t$ do
            $l_t \leftarrow l_t + \log p_\theta(a_v|s_i)$;
        end
        $J_i \leftarrow J_i + q(a_i|s, A_t) \cdot l_t$;
    end
    $J \leftarrow J - r_i \cdot \omega(a_i|s_i) \cdot \frac{1}{\tau} J_i$;
end
$J \leftarrow \frac{1}{n} J$;
Return $J$;
```

5 EXAMPLE: APPLICATION ON A SOFTMAX RECOMMENDER

Having derived the general two-stage off-policy policy gradient, next we show how to apply the two-stage off-policy correction on a softmax recommender. Softmax is commonly used as the output of recommender systems [9, 34] that treat recommendation as a classification or retrieval problem. To better illustrate the intuitions of how the proposed method works, we start from the cross-entropy loss, a standard supervised learning objective without off-policy correction, and then compare the two-stage off-policy learning objective with it. We will see that when we define binary reward on each user state and item pair (s, a), e.g.,

$$r(s, a) = \begin{cases} 1, \text{ if } a \text{ is clicked given state } s, \\ 0, \text{ else,} \end{cases}$$

the two-stage off-policy learning objective is a weighted version of cross-entropy. In the rest of this section, we will stick to the binary click reward for simplicity, although the proposed method can deal with arbitrary reward functions.

Cross-entropy Loss. Given the candidate generation model output $p_\theta(a_i|s_i)$ on all the logged samples $i = 1, \cdots, n$, the cross-entropy loss can be written as

$$J_{\text{CE}}(\theta) = -\frac{1}{n}\sum_{i=1}^{n} r(s_i, a_i)\log p_\theta(a_i|s_i). \tag{9}$$

One-stage IPS Loss. Before we step into the two-stage off-policy learning, we first give the IPS objective assuming that the candidate generation model itself forms a one-stage recommender system and outputs a single recommendation. This can be viewed as the result of directly applying existing off-policy correction methods on the candidate generation model. This objective, which we call it *one-stage IPS*, is

$$J_{\text{1-IPS}}(\theta) = -\frac{1}{n}\sum_{i=1}^{n}\frac{\text{sg}(p_\theta(a_i|s_i))}{\beta(a_i|s_i)} r(s_i, a_i)\log p_\theta(a_i|s_i), \tag{10}$$

where sg indicates a stop-gradient operation, and recall β is the behavior policy.

Two-stage IPS Loss. We then give the two-stage off-policy learning objective, which we call it *two-stage IPS*, in terms of the candidate generation model p_θ and the ranking model q:

$$J_{\text{2-IPS}}(\theta) = -\frac{1}{n}\sum_{i=1}^{n}\frac{\text{sg}(p_\theta(a_i|s_i))}{\beta(a_i|s_i)} r(s_i, a_i) h(\theta), \tag{11}$$

where

$$\begin{aligned} h(\theta) = &\sum_{A^{k-1}}\Big[\text{sg}(p_\theta(A^{k-1}|s_i))q(a_i|s_i, \{a_i\}\cup A^{k-1})\cdot \\ &\qquad\Big(\log p_\theta(a_i|s_i) + \log p_\theta(A^{k-1}|s_i)\Big)\Big] \\ \simeq &\frac{1}{\tau}\sum_{A^{k-1}\sim p_\theta}\Big[q(a_i|s_i, \{a_i\}\cup A^{k-1})\cdot \\ &\qquad\Big(\log p_\theta(a_i|s_i) + \log p_\theta(A^{k-1}|s_i)\Big)\Big]. \end{aligned} \tag{12}$$

The $\simeq$ is due to the approximation by sampling and τ is the sample size. We have summarized the procedure of approximating the two-stage IPS loss in Algorithm 1.

Finally, we finish this section by highlighting some intuitions about how the two-stage IPS loss can work better than the one-stage IPS loss. For some logged user-item pair (s_i, a_i) with positive feedback,

- if a_i is ranked low by the ranking model, i.e.,
$$q(a_i|s_i, \{a_i\}\cup A^{k-1}) \ll 1$$
for most A^{k-1}, the gradient of the candidate generation model regarding this pair will be very small. This helps the candidate generation model focus on more important items that will be favorable by the ranking model;
- consider a case that a_i is ranked high by the ranking model, but there is a strong false positive item a_j $(j \neq i)$ that is mistakenly ranked even higher by the ranking model. We will have
$$q(a_i|s_i, \{a_i\}\cup A^{k-1}) \ll 1$$
for A^{k-1} that contains a_j. If there is no positive feedback for (s_i, a_j) in the logged data (which is very likely as a_j is a false positive), the candidate sets containing a_j will have much lower weights compared to those without a_j. This will make the candidate generation model avoid nominating a_j and therefore compensate for the mistake made by the ranking model.

6 EXPERIMENT

In this section, we provide empirical evidence to demonstrate the effectiveness of our proposed two-stage off-policy policy gradient method.

We test on a softmax recommender and compare the proposed method with the two baseline learning methods mentioned in Section 5, as these methods are commonly used in current industrial recommender systems [7, 9]. To recap, the methods for comparison are:

- Cross-Entropy: the supervised learning method commonly used for the softmax recommender and the corresponding learning objective is the cross-entropy loss (Eq. 9).
- One-stage IPS (1-IPS): the method used in Chen et al. [7], which is a direct application of the off-policy policy gradient method on the candidate generation model. The corresponding learning objective is the one-stage IPS loss (Eq. 10).
- Two-stage IPS (2-IPS): the two-stage off-policy policy gradient method proposed by this paper. The corresponding learning objective is the two-stage IPS loss (Eq. 11).

Here we have used the name of the learning objective as the abbreviation of the corresponding learning method.

6.1 Experiment Methodology

Next we introduce the general experiment methodology of evaluating different learning methods.

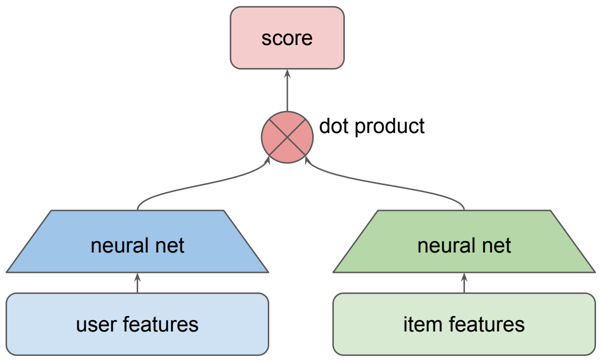
To evaluate whether an off-policy learning method can well-correct the biases in logged data, ideally we should conduct online test by running the corrected target policy in real recommendation environments and collect the feedback. However, such environments are rarely publicly available. In our experiments, we adopted two surrogate methods that are commonly used in the literature to mimic the online test for off-policy learning evaluation. The first method is called supervised-to-bandit conversion [11, 32], and the second method is online simulation [41]. We explain the two evaluation methods in more details below.

6.1.1 Supervised-to-Bandit Conversion. Following Swaminathan and Joachims [32], suppose we take a supervised multi-class mutli-label classification dataset

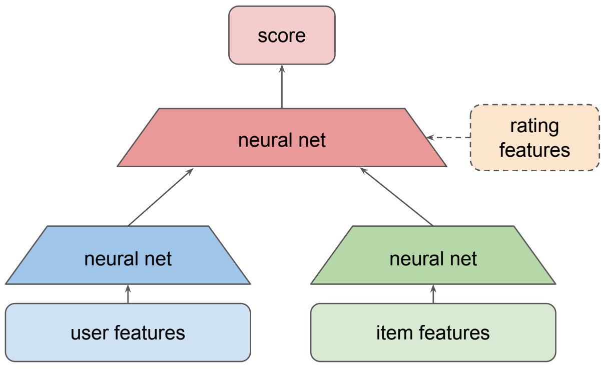
$$\mathcal{D}_{\text{full}} = \{(\mathbf{x}^i, \mathbf{y}^i)\}_{i=1}^{n},$$

where $\mathbf{x}^i \in \mathbb{R}^d$ is the feature vector of the data point i, and $\mathbf{y}^i \in \{0, 1\}^m$ is a binary vector with each element $\mathbf{y}^i_a$ indicating whether the data point i belongs to the class $a \in \{1, 2, \cdots, m\}$. In $\mathcal{D}_{\text{full}}$, we have full-label information, i.e., we have observed a binary label for every class on each sample.

To build a partial-label bandit dataset, we could use a behavior policy $\beta(a|\mathbf{x})$ that takes the feature vector $\mathbf{x}^i$ of some data point i as



(a) The two-tower model architecture for the candidate generation model and the behavior policy model. This architecture models the users and items with two separate towers and can be optimized to be very computationally efficient.



(b) The model architecture for the ranking model and the simulator model. This architecture is usually more powerful than the two-tower architecture because it can capture the crosses between the user and item features at a low level, and take extra rating features.

Figure 2: Illustration of model architectures.

input and outputs a choice of class a. Then only the corresponding label $\mathbf{y}_a^i$ is revealed. By sampling data point i and the choice of class $a \sim \beta(a|\mathbf{x}^i)$, we can form a bandit dataset

$$\mathcal{D}_{\text{bandit}} = \{(\mathbf{x}^j, a^j, p^j, r^j)\}_{j=1}^{l},$$

where l is the total number of samples, p^j is the probability of the behavior policy for a certain user-item pair, and $r^j = \mathbf{y}_{a^j}^j$ is the reward. We can then train recommender systems with various off-policy learning methods on such bandit datasets $\mathcal{D}_{\text{bandit}}$.

In the evaluation stage, we can run the target policy $\pi(a|\mathbf{x})$ on some holdout set $\mathbf{x}$ and and see if the resulted choice $a \sim \pi(a|\mathbf{x})$ can hit positive labels. We can do the online test with the supervised-to-bandit converted dataset because we actually have access to the label of any pair of $(\mathbf{x}, a)$.

6.1.2 Online Simulation. While the supervised-to-bandit conversion is convenient, it requires the full-label information of a dataset. This requirement prevents the use of most publicly available recommender system datasets because they only have partially observed user preferences over the whole item corpus. As we are interested in recommender systems, we further adopted the online simulation method [41], which is another surrogate online test method that allows us to use recommender system datasets.

We represent a recommender system dataset with m records of logged feedback as

$$\mathcal{D} = \{(u^i, a^i, r^i)\}_{i=1}^{m},$$

where $u^i \in \mathcal{U}$ indicates a user and $a^i \in \mathcal{A}$ indicates an item. We also have $m < n = |\mathcal{U}| \times |\mathcal{A}|$ to reflect we only have partial-label information.

Following Zhao et al. [41], we first train a simulator model h_0 using $\mathcal{D}$. The simulator model takes the user-item pair (u^i, a^i) and their features as input, and predicts the immediate feedback r^i. The well-trained simulator h_0 can then serve as a proxy of the real online environment, as it can give the feedback for all the user-item pairs. And we can build a full-label proxy dataset $\mathcal{D}_{\text{full}}$. Finally, we can apply supervised-to-bandit conversion on $\mathcal{D}_{\text{full}}$ in the same way as shown in Section 6.1.1 and get a bandit dataset $\mathcal{D}_{\text{bandit}}$.

6.1.3 Experiment Procedure. We then introduce the general experiment procedure after we get a partial-label bandit dataset $\mathcal{D}_{\text{bandit}}$ and its corresponding full-label dataset $\mathcal{D}_{\text{full}}$.

(1) Train a ranking model on the bandit dataset $\mathcal{D}_{\text{bandit}}$.
(2) Given a learning method, train a candidate generation model with this learning method. Note that the trained ranking model from (1) is used for two-stage IPS method.
(3) Evaluate the two-stage system assembled (in the way shown in Figure 1) with the ranking and candidate generation models trained from step (1) and (2) respectively.
(4) Repeat (2) and (3) for different learning methods (Cross-Entropy, 1-IPS, and 2-IPS).

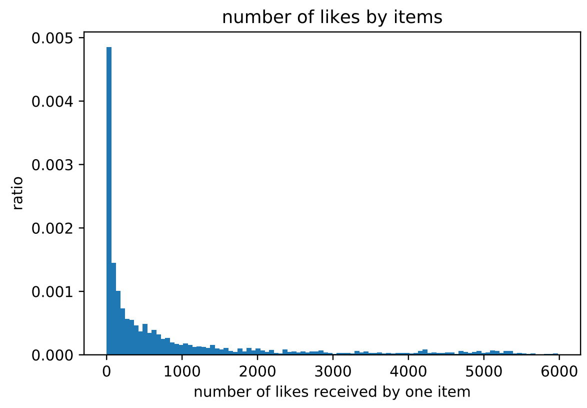
6.2 Detailed Setup

Having explained the general experiment methodology, we next introduce more details of the experiment setup.

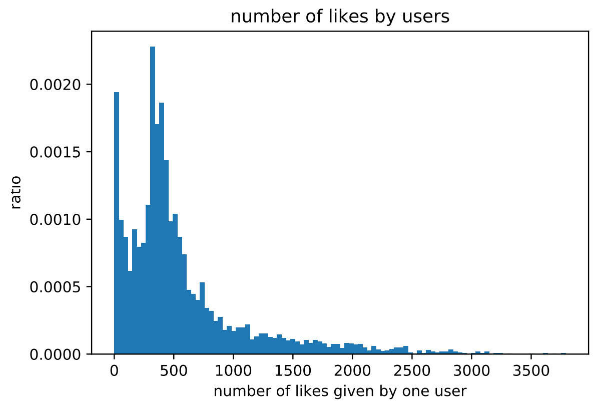
6.2.1 Datasets. We use two publicly available datasets for our experiments: *MovieLens-1M*[2], a popular recommender system benchmark dataset, and *Wiki10-31K* [43], an extreme multi-label classification dataset[3].

The MovieLens-1M dataset contains approximately 1M ratings of 3,900 movies made by 6,040 users. Each user has some demographic features and each movie has title and genre features. Each rating has a 5-score rating and the time of the rating. We binarize the explicit rating data by keeping ratings of 4 or higher and interpret them as the positive implicit feedback. Similarly, ratings less than 4 are interpreted as negative feedback. All the user-item interactions with observed feedback are used for training the simulator. We need relatively dense ratings of user-item pairs and user features to

[2]https://grouplens.org/datasets/movielens/1m/
[3]http://manikvarma.org/downloads/XC/XMLRepository.html



(a) The histogram of likes received by the items.



(b) The histogram of likes given by the users.

Figure 3: Histograms of the simulation data. Both histograms have long tails in the distribution.

learn a good simulator that can generalize well on missing ratings. Therefore we choose the MovieLens-1M dataset rather than its larger counterparts, e.g., MovieLens-20M[4], which do not have user features and where the rating is sparser.

The Wiki10-31K dataset contains approximately 20K samples, each sample with bag-of-words features and labels of 31K classes. Each sample can have more than one class labeled as positive. We choose this dataset rather than the ones in the UCI machine learning repository that are used in Dudík et al. [11] and Swaminathan and Joachims [32] because we want to evaluate the performance of a two-stage recommender system and the dataset with large label space is more natural for such systems. In the rest of this section, we will refer the sample in this dataset as "user" and the class as "item" to unify the terminology.

6.2.2 Model Architectures. We use neural networks for all the models involved in the experiments and we keep the architectures of the models to be the same for different learning methods for comparison.

[4]https://grouplens.org/datasets/movielens/20m/

To test these learning methods, we need to specify the architectures of three models, the behavior policy model, the candidate generation model, and the ranking model on the Wiki10-31K dataset. On MovieLens-1M, we need to specify an additional simulator model. For behavior policy and candidate generation models, we need compact and efficient model with acceptable performance. While for ranking and simulator models, we usually want them to be much more powerful. Thus, larger model capacity and auxiliary features could be used.

For all the behavior policy model and the candidate generation model, we adopt the two-tower neural network architecture from Yi et al. [40]. As shown in Figure 2a, the two-tower architecture first models the user representation and the item representation with two separate neural networks, which are called towers, and then calculate the dot product between the user and item representations to capture the relatedness between the user and the item. The separation between user and item representations, while not capturing crosses between the user and the item, can be optimized to be very computationally efficient. On the Wiki10-31K dataset, we set the user tower as a network with a word embedding layer followed by a fully-connected layer. We set all the hidden-layer dimensions as 20. The item tower is just set as an item embedding layer as there is no item feature. On the MovieLens-1M dataset, we set the user tower as a network beginning with multiple categorical feature embedding and then followed by a fully-connected layer. We set the categorical feature embedding dimension as 10 for each feature, and set the dimension of the fully-connected layer as 20. The item tower is simply the concatenation of the item embedding and item feature embedding, with a total size 20.

For the ranking model and the simulator model for MovieLens-1M, as shown in Figure 2b, we instead combine the the representations of the user and the item, and take the user-item pairs as input [9, 41]. On the Wiki10-31K dataset, we concatenate the user tower and the item tower as mentioned in the candidate generation model, and stack another fully-connected layer on top of the combined representation. All the hidden-layer dimensions are still set to 20. The ranking model on the MovieLens-1M dataset is similar except there is a time of the rating feature associated with each movie rating. So in addition to the user tower and item tower, we also concatenate the rating feature. Compared to the ranking model, the simulator model for the MovieLens-1M dataset takes a larger model capacity, with two fully-connected layers (with hidden size 100 and 50, respectively) following the combined representation.

6.2.3 Evaluation Metrics. Although we are improving the learning of the candidate generation model, we are primarily interested in the performance of the whole two-stage recommender system. So rather than directly measuring the quality of the top recommendations of the candidate generation models, we measure the quality of the top recommendations given by the ranking model conditioned on the candidate set provided by the candidate generation model. That is, given a user (and context), we first select the top-k items predicted by the candidate generation model as a candidate set, then feed this candidate set to the ranking model to re-rank these candidates. And we measure the quality of the top-recommended items after re-ranking. We use $k = 50$ for Wiki10-31K and $k = 30$ for MovieLens-1M.

Table 2: Ranking metrics (%) of two-stage evaluation on MovieLens-1M. The number after ± indicates the standard error of the mean over 20 runs. The percentage in indicates the relative improvement over the Cross-Entropy method.

	Precision@5	Precision@10	Recall@5	Recall@10	NDCG@5	NDCG@10
Cross-Entropy	95.8 ± 0.1 (+0%)	93.9 ± 0.0 (+0%)	2.2 ± 0.0 (+0%)	3.5 ± 0.0 (+0%)	96.4 ± 0.1 (+0%)	95.0 ± 0.1 (+0%)
1-IPS	95.1 ± 0.2 (-1%)	93.3 ± 0.2 (-1%)	2.0 ± 0.1 (-8%)	3.4 ± 0.1 (-3%)	95.7 ± 0.3 (-1%)	94.3 ± 0.2 (-1%)
2-IPS	**98.1** ± 0.1 (+2%)	**96.8** ± 0.1 (+3%)	**2.9** ± 0.0 (+33%)	**4.9** ± 0.0 (+39%)	**98.3** ± 0.1 (+2%)	**97.5** ± 0.1 (+3%)

Table 3: Ranking metrics (%) of one-stage evaluation on MovieLens-1M. Notations are the same as Table 2.

	Precision@5	Precision@10	Recall@5	Recall@10	NDCG@5	NDCG@10
Cross-Entropy	86.3 ± 0.1 (+0%)	86.4 ± 0.1 (+0%)	1.2 ± 0.0 (+0%)	2.4 ± 0.0 (+0%)	86.3 ± 0.1 (+0%)	86.4 ± 0.1 (+0%)
1-IPS	90.1 ± 0.4 (+4%)	88.9 ± 0.3 (+3%)	1.5 ± 0.0 (+27%)	2.8 ± 0.1 (+17%)	90.3 ± 0.4 (+5%)	89.4 ± 0.3 (+3%)
2-IPS	**95.7** ± 0.2 (+11%)	**95.3** ± 0.2 (+10%)	**2.3** ± 0.1 (+102%)	**4.5** ± 0.1 (+89%)	**95.4** ± 0.3 (+11%)	**95.3** ± 0.2 (+10%)

For MovieLens-1M, the corpus is relatively small so we report multiple finer-grained ranking metrics including: precision, recall, and normalized Discounted Cumulative Gain (NDCG) with truncation to be 5 or 10.

For Wiki10-31K, as the task is relatively hard, which requires the recommender system to recommend items from a large corpus of 31K items, we just measure the top-1 precision of the two-stage system.

As a reference, we also report these metrics that directly measure the ranking given by the candidate generation models. To differentiate the two different measuring methods, we call the former one (measuring on the predictions of the whole two-stage system) as two-stage evaluation, and the latter one as one-stage evaluation.

6.2.4 Implementation Details for MovieLens-1M. We summarize more implementation details for experiments on MovieLens-1M.

Simulator Model Training. As mentioned in Section 6.1.2, for this dataset, we need to train an online environment simulator. We split the original logged ratings from the MovieLens-1M into training, validation, and test set with 3:1:1. And we train the simulator with binary classification on each logged rating of user-item pairs. The trained simulator can achieve considerable performance on predicting the missing ratings [5].

Then we use this simulator to generate predictions of ratings for all the user-item pairs. To do this, we need to impute the rating features of the missing user-item pairs. In this case, we have one rating feature, the time that the rating is made. We impute each rating time by sampling from a Gaussian distribution with its mean and standard deviation specified by the empirical mean and standard deviation of all the logged rating time of the corresponding user.

After we get the predictions for all user-item pairs, we choose a threshold such that the false positive rate on the validation set is relatively low (around 0.1 in this case). Then we are able to assign a binary label for each user-item pair, with 1 indicating that the user likes the item and 0 indicating that the user does not like the item.

In Figure 3, we plot the histograms of the number of positive feedback (i.e., likes) received by the items (Figure 3a) and given by the users (Figure 3b), respectively. As can be seen from the plots, the histogram of the items follows a long tail distribution, and the histogram of the users has its peak around 500 and also has a long tail in the distribution.

[5]This is measured by an AUC value of 0.723 on test set.

Behavior Policy and Ranking Models Training. We train a behavior policy model based on 10,000 random user-item pairs with labels generated by the simulator. We construct a bandit dataset by sampling the top-5 items predicted by the behavior policy for each user as the training data. And then we train a ranking model based on the bandit dataset with binary classification. For evaluation, we first de-duplicate the items that appeared in the training set and then evaluate on the remaining items. We randomly split 2K users as validation set and the remaining 4K users as test set.

Candidate Generation Models Training. Finally we train the candidate generation models with different learning methods on the bandit training dataset. For all methods, we use AdaGrad [10] as the optimizer with the initial learning rate set as 0.05. For both 1-IPS and 2-IPS, we apply self-normalization and weight capping with $c_1 = 10, c_2 = 0.01$. For 2-IPS, we set $\alpha = 1e - 2$ (see Eq. 8) and the candidate set sample size $\tau = 100$ (see Eq. 12). For each learning method, we train 20 candidate generation models initialized with different random seeds. We apply early stopping with the metric Precision@10 on the validation set and report the average values of all metrics on the test set. Note that we apply early stopping separately for one-stage evaluation and two-stage evaluation.

As a reference for the task difficulty level of the simulation data, we also run WRMF [16] on this bandit dataset, and it obtains 78.7% for Precision@5, 76.8% for Precision@10, 1.0% for Recall@5, 2.0% for Recall@10, 78.8% for NDCG@5, and 77.5% for NDCG@10.

6.2.5 Implementation Details for Wiki10-31K. We summarize more implementation details for experiments on Wiki10-31K. In this dataset, we do not need to train the simulator as we have the full-label information.

Behavior Policy and Ranking Models Training. We first split the dataset into train, validation, and test sets with size 10K:4K:6K, by different users. The test set is from the official split. Throughout our experiment, the test set is held out until the final evaluation. As there are a few dominant items that are associated with most users, we removed 30 items with the highest frequency from the dataset to make the task harder. Then we train a behavior policy model with 500 random samples from the training set, and train a ranking model on the full training set. Following standard supervised-to-bandit conversion mentioned in Section 6.1.1, we randomly sample 20K users with replacement from the training set, and reveal the label of the top-1 item predicted by the behavior policy model to construct a bandit dataset.

Candidate Generation Models Training. We train the candidate generation models with different learning methods on the bandit training dataset. We find the candidate generation models are likely to crash if they are trained with either 1-IPS or 2-IPS from scratch, possibly due to the large item space. However, we find this can be easily fixed by pre-training the candidate generation models by the Cross-Entropy method for 100 steps. And we find the final performance of the IPS methods is not sensitive to the number of pre-training steps. For both 1-IPS and 2-IPS, we apply self-normalization and weight capping with $c_1 = 10, c_2 = 0.01$. For 2-IPS, we set $\alpha = 1e - 5$ and the candidate set sample size $\tau = 100$. We use the same setting as that on MovieLens-1M dataset for the optimizer, repeated runs, and early stopping strategy (except for using the top-1 precision as the early stopping metric).

6.3 Off-policy Learning Results

6.3.1 Results on MovieLens-1M. We first report the experiment results on the MovieLens-1M dataset, which are shown in Table 2 and Table 3. We give the mean, the standard error of the mean, and the relative improvement over the Cross-Entropy method for each metric. From both tables, it is easy to see that the relative trend is consistent for all ranking metrics.

Table 2 shows the ranking metrics of different methods in the two-stage evaluation. The proposed 2-IPS outperforms all the baselines in all ranking metrics, indicating that the two-stage off-policy policy gradient indeed improves the performance of the whole two-stage recommender systems over learning methods commonly used in industrial recommender systems.

Table 3 shows the same metrics in the one-stage evaluation. While the 1-IPS method outperforms the Cross-Entropy method as expected, the proposed 2-IPS method also outperforms the 1-IPS method in this evaluation regime. We conjecture this may be caused by the fact that the ranking model is more powerful than the candidate generation model, and there is a transfer learning effect as a side-product of the 2-IPS method.

Bringing the two types of evaluations together, it is interesting to see that, while the 1-IPS method outperforms the Cross-Entropy method largely in the one-stage evaluation, the advantage does not carry to the two-stage evaluation. In fact, the 1-IPS method is even a little worse than the Cross-Entropy method in two-stage evaluation. This highlights the importance of taking the whole system into consideration when designing the learning methods because improving part of the system does not necessarily lead to the improvement of the whole system.

6.3.2 Results on Wiki10-31K. The results on Wiki10-31K are shown in Table 4. We also give the mean, the standard error of the mean, and the relative improvement over the Cross-Entropy method for each metric. Similar with the results on MovieLens-1M, the proposed 2-IPS method outperforms all the baseline methods on both two-stage evaluation and one-stage evaluation, indicating the effectiveness of our method. On this dataset, the trends of the two types of the evaluations are consistent and the 1-IPS method outperforms the Cross-Entropy method in both one-stage and two-stage evaluations.

Table 4: Top-1 precisions (%) of two-stage evaluation and one-stage evaluation on Wiki10-31K. Notations are the same as Table 2.

	One-stage Eval	Two-stage Eval
Cross-Entropy	19.3 ± 0.4 (+0%)	38.8 ± 0.3 (+0%)
1-IPS	20.8 ± 0.4 (+8%)	39.6 ± 0.3 (+2%)
2-IPS	**21.7** ± 0.4 (+12%)	**40.5** ± 0.3 (+4%)

7 CONCLUSION AND FUTURE WORK

In this work, we tackled a novel challenge of correcting the biases in learning a two-stage recommender system from the logged implicit feedback. By formulating the two-stage recommender system policy as the decomposition of the candidate generation policy and ranking policy, we derived a two-stage off-policy policy gradient method for the candidate generation model. We also overcame the expensive exact calculation of this gradient by an efficient Monte Carlo approximation algorithm. Through extensive experiments on real-world datasets, we demonstrated the proposed method is able to improve the performance of the two-stage recommender systems over common learning methods in current industrial practices.

Following the idea of optimizing the whole-system performance in this paper, we would like to look into more practical recommender system settings in future work. For example, how to extend this work from a two-stage recommender system to a multi-stage recommender system? And, while end-to-end training of the ranking model and the candidate generation model simultaneously seems engineeringly impractical for now, can we also incorporate the knowledge of the candidate generation model into the off-policy learning of the ranking model? We believe that this is an important direction to further improve large-scale recommender systems.

Acknowledgments

The authors would like to thank Lihong Li and Zhaoran Wang for providing helpful insights on off-policy learning; Cheng Li and Weijing Tang for useful discussions related to this project.

REFERENCES

[1] Aman Agarwal, Ivan Zaitsev, Xuanhui Wang, Cheng Li, Marc Najork, and Thorsten Joachims. 2019. Estimating position bias without intrusive interventions. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*. ACM, 474–482.

[2] Shipra Agrawal and Navin Goyal. 2013. Thompson sampling for contextual bandits with linear payoffs. In *International Conference on Machine Learning*. 127–135.

[3] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. 2002. Finite-time analysis of the multiarmed bandit problem. *Machine learning* 47, 2-3 (2002), 235–256.

[4] Fedor Borisyuk, Krishnaram Kenthapadi, David Stein, and Bo Zhao. 2016. CaSMoS: A framework for learning candidate selection models over structured

queries and documents. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 441–450.

[5] Djallel Bouneffouf, Amel Bouzeghoub, and Alda Lopes Gançarski. 2012. A contextual-bandit algorithm for mobile context-aware recommender system. In *International Conference on Neural Information Processing*. Springer, 324–331.

[6] Olivier Chapelle and Lihong Li. 2011. An empirical evaluation of thompson sampling. In *Advances in neural information processing systems*. 2249–2257.

[7] Minmin Chen, Alex Beutel, Paul Covington, Sagar Jain, Francois Belletti, and Ed H Chi. 2019. Top-k off-policy correction for a REINFORCE recommender system. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*. ACM, 456–464.

[8] Wei Chu, Lihong Li, Lev Reyzin, and Robert Schapire. 2011. Contextual bandits with linear payoff functions. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. 208–214.

[9] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM conference on recommender systems*. ACM, 191–198.

[10] John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research* 12, Jul (2011), 2121–2159.

[11] Miroslav Dudík, John Langford, and Lihong Li. 2011. Doubly robust policy evaluation and learning. *arXiv preprint arXiv:1103.4601* (2011).

[12] Chantat Eksombatchai, Pranav Jindal, Jerry Zitao Liu, Yuchen Liu, Rahul Sharma, Charles Sugnet, Mark Ulrich, and Jure Leskovec. 2018. Pixie: A system for recommending 3+ billion items to 200+ million users in real-time. In *Proceedings of the 2018 World Wide Web Conference*. International World Wide Web Conferences Steering Committee, 1775–1784.

[13] Seth Flaxman, Sharad Goel, and Justin M Rao. 2016. Filter bubbles, echo chambers, and online news consumption. *Public opinion quarterly* 80, S1 (2016), 298–320.

[14] Alexandre Gilotte, Clément Calauzènes, Thomas Nedelec, Alexandre Abraham, and Simon Dollé. 2018. Offline a/b testing for recommender systems. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*. ACM, 198–206.

[15] Daniel G Horvitz and Donovan J Thompson. 1952. A generalization of sampling without replacement from a finite universe. *Journal of the American statistical Association* 47, 260 (1952), 663–685.

[16] Yifan Hu, Yehuda Koren, and Chris Volinsky. 2008. Collaborative filtering for implicit feedback datasets. In *2008 Eighth IEEE International Conference on Data Mining*. Ieee, 263–272.

[17] Nan Jiang and Lihong Li. 2015. Doubly robust off-policy value evaluation for reinforcement learning. *arXiv preprint arXiv:1511.03722* (2015).

[18] Thorsten Joachims, Adith Swaminathan, and Tobias Schnabel. 2017. Unbiased learning-to-rank with biased feedback. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*. ACM, 781–789.

[19] Wang-Cheng Kang and Julian McAuley. 2019. Candidate Generation with Binary Codes for Large-Scale Top-N Recommendation. *arXiv preprint arXiv:1909.05475* (2019).

[20] John Langford and Tong Zhang. 2008. The epoch-greedy algorithm for multi-armed bandits with side information. In *Advances in neural information processing systems*. 817–824.

[21] Sergey Levine and Vladlen Koltun. 2013. Guided policy search. In *International Conference on Machine Learning*. 1–9.

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

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

[24] Rémi Munos, Tom Stepleton, Anna Harutyunyan, and Marc Bellemare. 2016. Safe and efficient off-policy reinforcement learning. In *Advances in Neural Information Processing Systems*. 1054–1062.

[25] Feiyang Pan, Qingpeng Cai, Pingzhong Tang, Fuzhen Zhuang, and Qing He. 2019. Policy Gradients for Contextual Recommendations. In *The World Wide Web Conference*. ACM, 1421–1431.

[26] Doina Precup, Richard S Sutton, and Sanjoy Dasgupta. 2001. Off-policy temporal-difference learning with function approximation. In *ICML*. 417–424.

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

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

[29] Aleksandrs Slivkins, Filip Radlinski, and Sreenivas Gollapudi. 2013. Ranked bandits in metric spaces: learning diverse rankings over large document collections. *Journal of Machine Learning Research* 14, Feb (2013), 399–436.

[30] Alex Strehl, John Langford, Lihong Li, and Sham M Kakade. 2010. Learning from logged implicit exploration data. In *Advances in Neural Information Processing Systems*. 2217–2225.

[31] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. 2000. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*. 1057–1063.

[32] Adith Swaminathan and Thorsten Joachims. 2015. Batch learning from logged bandit feedback through counterfactual risk minimization. *Journal of Machine Learning Research* 16, 1 (2015), 1731–1755.

[33] Adith Swaminathan, Akshay Krishnamurthy, Alekh Agarwal, Miro Dudik, John Langford, Damien Jose, and Imed Zitouni. 2017. Off-policy evaluation for slate recommendation. In *Advances in Neural Information Processing Systems*. 3632–3642.

[34] Jiaxi Tang, Francois Belletti, Sagar Jain, Minmin Chen, Alex Beutel, Can Xu, and Ed H Chi. 2019. Towards neural mixture recommender for long range dependent user sequences. In *The World Wide Web Conference*. ACM, 1782–1793.

[35] Liang Tang, Yexi Jiang, Lei Li, Chunqiu Zeng, and Tao Li. 2015. Personalized recommendation via parameter-free contextual bandits. In *Proceedings of the 38th international ACM SIGIR conference on research and development in information retrieval*. ACM, 323–332.

[36] Liang Tang, Romer Rosales, Ajit Singh, and Deepak Agarwal. 2013. Automatic ad format selection via contextual bandits. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*. ACM, 1587–1594.

[37] Philip Thomas and Emma Brunskill. 2016. Data-efficient off-policy policy evaluation for reinforcement learning. In *International Conference on Machine Learning*. 2139–2148.

[38] Chih-Chun Wang, Sanjeev R Kulkarni, and H Vincent Poor. 2005. Bandit problems with side observations. *IEEE Trans. Automat. Control* 50, 3 (2005), 338–355.

[39] Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning* 8, 3-4 (1992), 229–256.

[40] Xinyang Yi, Ji Yang, Lichan Hong, Derek Zhiyuan Cheng, Lukasz Heldt, Aditee Kumthekar, Zhe Zhao, Li Wei, and Ed Chi. 2019. Sampling-bias-corrected neural modeling for large corpus item recommendations. In *Proceedings of the 13th ACM Conference on Recommender Systems*. ACM, 269–277.

[41] Xiangyu Zhao, Liang Zhang, Zhuoye Ding, Long Xia, Jiliang Tang, and Dawei Yin. 2018. Recommendations with negative feedback via pairwise deep reinforcement learning. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 1040–1048.

[42] Zhe Zhao, Lichan Hong, Li Wei, Jilin Chen, Aniruddh Nath, Shawn Andrews, Aditee Kumthekar, Maheswaran Sathiamoorthy, Xinyang Yi, and Ed Chi. 2019. Recommending what video to watch next: a multitask ranking system. In *Proceedings of the 13th ACM Conference on Recommender Systems*. ACM, 43–51.

[43] Arkaitz Zubiaga. 2012. Enhancing navigation on wikipedia with social tags. *arXiv preprint arXiv:1202.5469* (2012).