Reinforcement Recommendation with User Multi-aspect Preference

Xu Chen*
Beijing Key Laboratory of Big Data Management and Analysis Methods, Gaoling School of Artificial Intelligence, Renmin University of China
Beijing, China

Yali Du
Department of Computer Science
University College London
London, UK

Long Xia
School of Information Technology
York University
Toronto, Canada

Jun Wang
Department of Computer Science
University College London
London, UK

ABSTRACT

Formulating recommender system with reinforcement learning (RL) frameworks has attracted increasing attention from both academic and industry communities. While many promising results have been achieved, existing models mostly simulate the environment reward with a unified value, which may hinder the understanding of users' complex preferences and limit the model performance. In this paper, we consider how to model user multi-aspect preferences in the context of RL-based recommender system. More specifically, we base our model on the framework of deterministic policy gradient (DPG), which is effective in dealing with large action spaces. A major challenge for modeling user multi-aspect preferences lies in the fact that they may contradict with each other. To solve this problem, we introduce Pareto optimization into the DPG framework. We assign each aspect with a tailored critic, and all the critics share the same actor. The Pareto optimization is realized by a gradient-based method, which can be easily integrated into the actor and critic learning process. Based on the designed model, we theoretically analyze its gradient bias in the optimization process, and we design a weight-reuse mechanism to lower the upper bound of this bias, which is shown to be effective for improving the model performance. We conduct extensive experiments based on three real-world datasets to demonstrate our model's superiorities.

CCS CONCEPTS

• **Information systems → Personalization**.

KEYWORDS

Recommender system, Reinforcement learning, Multi-objective optimization

∗ Corresponding author.

ACM Reference Format:
Xu Chen*, Yali Du, Long Xia, and Jun Wang. 2021. Reinforcement Recommendation with User Multi-aspect Preference. In *Proceedings of the Web Conference 2021 (WWW '21), April 19–23, 2021 ,Ljubljana, Slovenia.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3442381.3449846

1 INTRODUCTION

Recommender system, as an effective remedy for information overloading, has been widely applied in a number of real-world applications, ranging from the e-commerce [15], social network [13] to music radio [9] and health caring [16]. Traditional models usually solve the recommendation task within the supervised learning frameworks, which fails to consider its basically interactive nature and users' long term engagement [2, 7, 20–22]. To alleviate these problems, recent years have witnessed an emerging trend of formulating the recommendation task as a reinforcement learning (RL) problem. Typically, the recommender system and user are regarded as the agent and environment, respectively [20, 21]. In each interaction step (see figure 1(a)), the system takes an action (recommends an item), and the user responses the action with a reward (e.g., rating and click). The final objective is to maximize the total rewards in the whole interaction sequence.

In the research of RL-based recommender system, existing models mainly focus on how to develop effective agents or simulate accurate environments to enhance the model performance. However, little attention has been paid to designing rewards for reflecting the users' complex preferences in realities, which is crucial for aligning the learned agent with the real user profiles. Previous methods usually approximate the reward by a single value, such as an integer rating reflecting the user's overall preference on the item, or a 0-1 value indicating whether the user has clicked/purchased the item. We argue that such simple reward can be problematic from many perspectives. To begin with, a unified value can be hard in distinguishing the user's fine-grained preference. Users with different preferences may give the same overall rating for the same item. For example, in figure 1(b), user A and B both score item X with the full ratings, but A is more interested in aspect d, while B likes more on aspect a. Only based on the fact that both A and B like item X, we cannot distinguish these users and accurately match them with the candidate items Y and Z, which vary a lot on different aspects. And then, an ideal agent should recommend items which can maximize the users' long-term engagement on all the aspects. However, the unified reward cannot provide enough signals to optimize the agent for some specific aspects. The high (or low) overall reward does not necessarily mean the user like (or dislike) all the item aspects.

To alleviate the above problems, in this paper, we propose to model user multi-aspect preference in the context of RL-based recommender system. The singleton reward in RL models is extended to a reward vector with each dimension corresponding the user's preference on an item aspect. While modeling user multi-aspect preference can more comprehensively understand the users, the user personalities can be quite complex and diverse, and different aspect preferences may not align (or even contradict) with each other. For example, in the hotel recommendation scenario, a user may enjoy the room size and environment, but these nice properties make the room cost more, which may lower the user's satisfaction on the price. Different aspect rewards may bring the model to different optimal solutions, and it is hard to learn a unified agent which can simultaneously maximize all the rewards. To reasonably define and learn an optimal model in such a scenario, we introduce Pareto optimization into the framework, where we aim to learn an agent, such that no other agent can concurrently increase all the aspect-level cumulated rewards. More specifically, we build our model based on deterministic policy gradient (DPG). We assign each item aspect with a tailored critic, and different critics share the same actor. For better aligning the actor output with the users' real preference, we introduce a supervised regularizer into the optimization process. For benefiting the model scalability, unlike previous heuristic strategies [8, 14], the Pareto optimization is integrated into our framework in a fully differentiable manner. We demonstrate that the introduction of Pareto optimization may bias the gradients, and we present the upper bound of the bias, which is shown to be related with the training batch size. Based on this theoretical result, a weight-reuse mechanism is further proposed to correct the gradient bias, and its effectiveness is verified in the experiments.

In a summary, in this paper, we propose to model the users' potentially inconsistent multi-aspect preferences in the RL-based recommender system. To achieve this goal, we extend traditional DPG with multi-objective rewards based on Pareto optimization. We theoretically analyze the upper bound of our model's gradient bias, and propose a weight-reuse method to correct this bias. Extensive experiments are conducted based on three real-world datasets to demonstrate our model's superiorities.

2 BACKGROUND

For more clear and integral presentation, in this section, we briefly introduce the necessary backgrounds of this work.

2.1 Recommendation as an RL Problem

RL-based recommender models hold the promise to optimize user long-term utilities. In a typical RL formulation [7, 20], the **action** is the recommended item, the **state** is represented by a user's previously interacted products. The **reward** is the rating from the user to the item. At each step t, the agent (recommender) takes an action a_t based on the current state s_t, and the user (environment) responses the action with a reward r_t. The state is transformed into

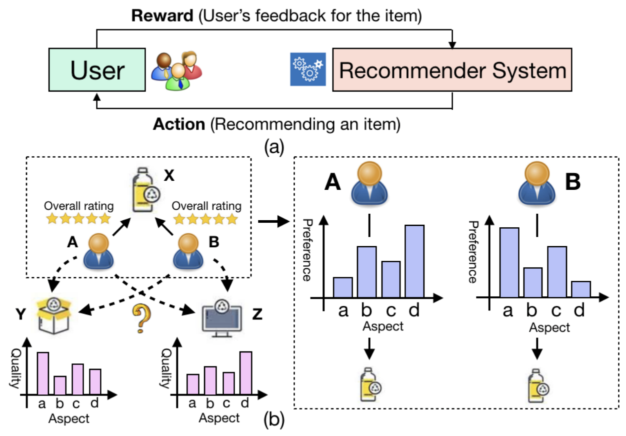

Figure 1: (a) Recommendation as a reinforcement learning problem. (b) A toy example for recommendation with user aspect-level preferences. Both A and B like item X (i.e., scoring it with 5 stars), but their specific preferences are quite different. A is more interested in aspect d, but B casts more attention on aspect a. Only based on the overall ratings, the system cannot well match these users with the candidate items (e.g., Y and Z), whose qualities varies much on different aspects.

s_{t+1} by incorporating a_t with s_t in a deterministic manner, that is, $s_{t+1} = (s_t, a_t)$. After many agent-environment interactions, we get several trajectories $(s_1, a_1, r_1, ..., s_T, a_T, r_T)$'s, and the goal is to maximize the sum of the rewards in these sequences.

A promising RL model for solving the recommendation task is the deterministic policy gradient (DPG) [12]. It can well handle the extremely large item sets by learning the actor within a continuous action space [7, 16, 19, 21]. Basically, DPG is an actor-critic framework. The critic is implemented based on a Deep Q-Network $Q(s, a|\boldsymbol{\phi})$, which is learned by:

$$\arg\min_{\boldsymbol{\phi}} \sum_{i=1}^{N} (y_i - Q(s_i, a_i|\boldsymbol{\phi}))^2, \tag{1}$$

where $y_i = r_i + \gamma Q(s_{i+1}, \mu(s_{i+1})|\boldsymbol{\phi}')$ is the target value, $\mu(s_{i+1})$ is the actor model. $\{(s_i, a_i, r_i, s_{i+1})\}_{i=1}^{N}$ are the training samples. γ is a discount factor used to balance the short- and long-term rewards. The parameter $\boldsymbol{\phi}'$ is updated from $\boldsymbol{\phi}$, but with a slower pace for stabilized training. After optimizing the critic, the actor is learned by maximizing the Q function, that is,

$$\arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{N} Q(s_i, \mu(s_i|\boldsymbol{\theta})), \tag{2}$$

In the whole training process, the critic and actor are alternatively optimized until convergence.

2.2 Pareto Optimization

Pareto optimization stems from the economics, and has been recently leveraged to solve multi-objective optimization problems (MOOP) [8, 11] in the machine learning community. In MOOP, the models are required to optimize a set of loss functions $\mathcal{L}(\theta) =$

$\{\mathcal{L}_1(\theta), \mathcal{L}_2(\theta), ..., \mathcal{L}_M(\theta)\}$. Usually, it is hard to find a unified parameter θ, which can simultaneously minimize all the $\mathcal{L}_i$'s. In such a scenario, Pareto optimization provides a reasonable method to define the optimal solutions. To begin with, different parameters are compared based on the following concept:

Definition 1. Pareto dominance. Suppose we have two parameters θ_A and θ_B, we say θ_A can dominant θ_B (denoted by $\theta_A \succ \theta_B$), if and only if $\mathcal{L}_i(\theta_A) \le \mathcal{L}_i(\theta_B)$, $\forall i \in \{1, 2, ...M\}$ and $\mathcal{L}_i(\theta_A) < \mathcal{L}_i(\theta_B)$, $\exists i \in \{1, 2, ...M\}$.

Intuitively, Pareto optimization aims to find a parameter θ^*, such that no other parameters can concurrently decrease all the loss functions. We call such parameter as Pareto efficient solution, which is formally defined as:

Definition 2. Pareto efficiency. For a parameter θ^*, if there is no other $\hat{\theta}$, such that $\hat{\theta} \succ \theta^*$, then we say θ^* is a Pareto efficient solution.

In this paper, we leverage Pareto optimization to extend DPG for optimizing multiple inconsistent rewards, and further apply the designed model to the task of recommender system.

3 PARETO DETERMINISTIC POLICY GRADIENT

In this section, we first define the problem studied in this paper, and then revise traditional DPG to make it compatible for multi-aspect rewards based on Pareto optimization (we call our model as PDPG). At last, we theoretically analyze the designed model by presenting the upper bound of its gradient bias, and propose a weight-reuse mechanism to lower this upper bound.

3.1 Problem Definition

Suppose we are given a user set $\mathcal{U} = \{u_1, u_2, ..., u_{|\mathcal{U}|}\}$ and an item set $\mathcal{I} = \{i_1, i_2, ..., i_{|\mathcal{I}|}\}$. The interactions between the users and items are collected in the set of $\mathcal{O} = \{(u,i)|\text{u has interacted with i}, u \in \mathcal{U}, i \in \mathcal{I}\}$. For each element $(u, i) \in \mathcal{O}$, the user can score on the item from multiple aspects, for example, in the hotel recommendation, a user can give ratings to a room on its environment, size, price and etc. We define the rating set $\mathcal{R}$ as $\{\boldsymbol{r}_{ui}|(u, i) \in \mathcal{O}\}$, where each $\boldsymbol{r}_{ui} = \{r_{ui,m}\}_{m=1}^{M}$ represents a user's ratings on an item's different aspects, and M is the aspect number. Given $\{\mathcal{U}, \mathcal{I}, \mathcal{O}, \mathcal{R}\}$, our task is to build an RL-based recommender model, such that it can maximize the users' long-term engagement on all the aspects.

3.2 Multi-aspect Critic

Different from previous RL-based recommender models, where the reward is unified in the optimization process, we have multiple rewards, which can be inconsistent with each other due to the users' diverse preferences on different item aspects. To well handle these rewards, we assign each aspect with a tailored critic.

Formally, suppose we have M item aspects, then the critics are defined as: $\{Q_1(s, a|\boldsymbol{\phi}_1), Q_2(s, a|\boldsymbol{\phi}_2), ..., Q_M(s, a|\boldsymbol{\phi}_M)\}$, and we optimize them based on:

$$\arg\min_{\boldsymbol{\phi}_m} \sum_{i=1}^{N} (y_{i,m} - Q_m(s_i, a_i|\boldsymbol{\phi}_m))^2, \ m = 1, 2, ...M \tag{3}$$

where $y_{i,m} = r_{i,m} + \gamma Q_m(s_{i+1}, a_{i+1}|\boldsymbol{\phi}'_m)$ is the target value for the mth critic. $\boldsymbol{r}_i = \{r_{i,m}\}_{m=1}^{M}$ is reward vector with each $r_{i,m}$ corresponding the user's preference on the mth aspect of item a_i.

Remark. i) If we have some prior knowledge about the relationship between different item aspects, the corresponding Q_m's can partially (or fully) share their parameters, which makes our critic optimization similar to a multi-task learning problem [11]. ii) Remember that the Q values represent the user's long-term engagements on different aspects. Larger Q value means the user may prefer more on the corresponding aspect. Thus, we can explain the recommendation by highlighting the aspect (e.g., x) with the largest Q value. A possible explanation template can be that "we recommend this item to you because it can satisfy your long-term engagement on aspect [x]". Such explanation cares more on the users' long-term preference, which is different from previous short-term recommendation explanations.

3.3 Pareto-efficient Actor

As mentioned in section 2.1, the actor in DPG is learned by maximizing the Q function (i.e., equation (2)). However, in our framework, there are multiple Q_m's, and it is difficult to find a unified $\boldsymbol{\theta}$ which can maximize all the Q functions. Straightforwardly, we can average different Q_m's with some predefined weights, and use single-reward models to learn the parameters. However, such method is limited in two aspects: on one hand, different objectives may vary in scale and importance. To find appropriate weights, one has to grid search the value for each weight. For M aspects with d search points, the model have to be optimized for d^{M-1} times, which is quite time-consuming and labor-intensive, especially when the aspect number becomes larger. On the other hand, such method only guarantees that the sum of the Q functions is maximized, but there is no mechanism to make sure that each Q_m is continually increased in the optimization process.

To alleviate these problems, we introduce Pareto optimization into the actor learning process. More specifically, we still average different Q functions, and the weights are assumed to be $\boldsymbol{w} = \{w_1, w_2, ..., w_M\}$, which induces a loss function:

$$l(\boldsymbol{\theta}) = -\sum_{m=1}^{M} w_m \sum_{i=1}^{N} Q_m(s_i, \mu(s_i|\boldsymbol{\theta})) \tag{4}$$

However, different from the previous methods, we dynamically adjust the weights to guarantee that different Q functions can be simultaneously increased, and finally achieve a Pareto efficient solution.

Formally, the weights are determined by solving the following quadratic programming (QP) problem:

$$\begin{aligned} &\min_{\boldsymbol{w}} || \sum_{m=1}^{M} w_m \nabla_{\boldsymbol{\theta}} \sum_{i=1}^{N} Q_m(s_i, \mu(s_i|\boldsymbol{\theta}))||_2^2 \\ &s.t. \ \boldsymbol{e}_k^T \boldsymbol{w} \ge b_k, \ \forall \ k \in [1, K] \\ &\mathbf{1}^T \boldsymbol{w} = 1, \ w_m \ge 0, \ \forall \ m \in [1, M] \end{aligned} \tag{5}$$

where $\mathbf{1}$ is an all-one vector. $\{\boldsymbol{e}_1, \boldsymbol{e}_2, ..., \boldsymbol{e}_K\}$ and $\{b_1, b_2, ..., b_K\}$ are predefined preference vectors and values. $\boldsymbol{e}_k = \{e_{k,1}, e_{k,2}, ..., e_{k,M}\}$, $\sum_{m=1}^{M} e_{k,m} = 1$, and $e_{k,m} \ge 0, b_k \ge 0, \forall \ m \in [1, M], \ k \in [1, K]$.

Remark. i) With the preference vectors and values, we actually incorporate the prior knowledge on different aspects into the training process implicitly. For example, if $\boldsymbol{e}_k$ is a one-hot vector, then the constraint in equation (5) aims to set an importance-level for the corresponding Q function by b_k. More general constraints can also be added according to the specific applications. ii) $\boldsymbol{w}$ can be computed as long as we know $\nabla_{\boldsymbol{\theta}} \sum_{i=1}^{N} Q_m(s_i, \mu(s_i|\boldsymbol{\theta}))$. Careful readers may find that $\nabla_{\boldsymbol{\theta}} \sum_{i=1}^{N} Q_m(s_i, \mu(s_i|\boldsymbol{\theta}))$ is just the gradient of $\boldsymbol{\theta}$ originally needed for optimizing the actor. This means that the gradient information is reusable, and our adopted Pareto optimization method can be smoothly infused into the DPG framework.

To see why the weights derived from equation (5) can lead to a Pareto-efficient solution, we have the following theory:

Theorem 1. *If $\boldsymbol{w}$ is determined by solving the quadratic programming (QP) problem of (5), then either one of the following holds:*
i) The solution to the optimization problem is 0, then the local Pareto efficient solution is achieved.
ii) $\boldsymbol{d} = \sum_{m=1}^{M} w_m \nabla_{\boldsymbol{\theta}} \sum_{i=1}^{N} Q_m(s_i, \mu(s_i|\boldsymbol{\theta}))$ is a gradient direction which does not decrease any Q function.

Proof. For i), if $|| \sum_{m=1}^{M} w_m \nabla_{\boldsymbol{\theta}} \sum_{i=1}^{N} Q_m(s_i, \mu(s_i|\boldsymbol{\theta}))]||_2^2 = 0$, then $\sum_{m=1}^{M} w_m \nabla_{\boldsymbol{\theta}} \sum_{i=1}^{N} Q_m(s_i, \mu(s_i|\boldsymbol{\theta})) = 0$. $\boldsymbol{\theta}$ cannot be improved to increase all Q functions, thus the local Pareto efficient solution achieves [6, 11].

For ii), we write the Lagrangian of problem (5) as:

$$\begin{aligned} &|| \sum_{m=1}^{M} w_m \nabla_{\boldsymbol{\theta}} \sum_{i=1}^{N} Q_m(s_i, \mu(s_i|\boldsymbol{\theta}))||_2^2 \\ &\quad + \sum_{k=1}^{K} \lambda_k (b_k - \sum_{m=1}^{M} e_{k,m} w_m) + \beta(1 - \sum_{m=1}^{M} w_m) \end{aligned} \tag{6}$$

with $\lambda_k \geq 0, \beta \geq 0, \forall k \in [1, K]$. The KKT condition for this Lagrangian yields:

$$\begin{aligned} &(\sum_{m=1}^{M} w_m \nabla_{\boldsymbol{\theta}} \sum_{i=1}^{N} Q_m(s_i, \mu(s_i|\boldsymbol{\theta})))^T \nabla_{\boldsymbol{\theta}} \sum_{i=1}^{N} Q_m(s_i, \mu(s_i|\boldsymbol{\theta})) \\ &\quad = \sum_{k=1}^{K} \lambda_k e_{k,m} + \beta \geq 0, \forall m \in [1, M] \end{aligned} \tag{7}$$

Recall that $\boldsymbol{d} = \sum_{m=1}^{M} w_m \nabla_{\boldsymbol{\theta}} \sum_{i=1}^{N} Q_m(s_i, \mu(s_i|\boldsymbol{\theta}))$, thus, for all m, we have $\boldsymbol{d}^T \nabla_{\boldsymbol{\theta}} \sum_{i=1}^{N} Q_m(s_i, \mu(s_i|\boldsymbol{\theta})) \geq 0$, which means $\boldsymbol{d}$ is a direction which does not decrease any Q function. □

Once we have determined $\boldsymbol{w}$, the actor parameter can be updated by $\boldsymbol{\theta} \longleftarrow \boldsymbol{\theta} + \alpha_{\theta} \boldsymbol{d}$, where α_{θ} is the learning rate.

Supervised regularization. Above, the policy $\mu(\cdot)$ is learned only based on the Q-values. For further constraining the optimal actions in a reasonable and safe space [16], we align the output of $\mu(\cdot)$ with the user's real preference on different items. In specific, we regularize $\mu(\cdot)$ in a supervised manner. The predicted action is forced to be closer to the items with positive feedback, and simultaneously stay away from the negative ones. The objective to be maximized is:

$$L(\boldsymbol{\theta}) = \sum_{i=1}^{N} y_i \log \sigma(\boldsymbol{q}_o^T \mu(s_i|\boldsymbol{\theta})) + (1 - y_i) \log(1 - \sigma(\boldsymbol{q}_o^T \mu(s_i|\boldsymbol{\theta}))) \tag{8}$$

where $\boldsymbol{q}_o$ is the embedding of item o, $y_i = 1$ if o is exactly the user's purchased item a_i, and 0 for the negatively sampled items. In this objective, the knowledge learned from the supervision signal is expected to influence the RL model, such that the predicted action is not far from the users' real preference.

When optimizing $L(\boldsymbol{\theta})$, we can straightforwardly merge $l(\boldsymbol{\theta})$ (i.e., equation 4) and $L(\boldsymbol{\theta})$ by a hyper-parameter (e.g., β) or learn them alternatively. However, both of these methods are suboptimal. For the former method, even if we determine $\boldsymbol{w}$ by solving problem (5), the gradient direction $\nabla_{\boldsymbol{\theta}}(l(\boldsymbol{\theta}) + \beta L(\boldsymbol{\theta}))$ does not necessarily satisfy equation (7), which is crucial for achieving the Pareto efficient solution. For the latter strategy, after optimizing $l(\boldsymbol{\theta})$, the parameter $\boldsymbol{\theta}$ will be further changed by $L(\boldsymbol{\theta})$, which cannot guarantee the increasing of all the Q-values.

To overcome these drawbacks, one may find that the objective (8) can be rewritten as $L(\boldsymbol{\theta}) = \tilde{Q}((o, y_i), \mu(s_i|\boldsymbol{\theta}))$, which is similar to a Q function if we regard (o, y_i) as a pseudo-state. Thus we take $L(\boldsymbol{\theta})$ as a special Q function, and incorporate it into equation (4) and (5) to learn the Pareto efficient solution jointly, where we assign $\tilde{Q}$ with an additional Pareto weight $\tilde{w}$. With this method, all the Q functions and $L(\boldsymbol{\theta})$ can be simultaneously optimized along a non-decreasing direction.

Learning algorithm. We present the whole training procedure of our framework in Algorithm 1. To begin with, many transactions are generated according to the current policy, and we push them into the "replay buffer" B (*line 5-11*). Then the critic is optimized based on the multi-aspect ratings (*line 12-19*). In the next, we derive the gradients of Q_m (or $\tilde{Q}$) w.r.t. $\boldsymbol{\theta}$, which will be used for computing $\boldsymbol{w}$ and the actor learning (*line 21-23*). The weight $\boldsymbol{w}$ is computed by solving problem (5) (*line 24*). And based on the learned $\boldsymbol{w}$, the actor is optimized by stochastic gradient ascent (*line 25-27*). At last, the target parameters are updated in a soft manner (*line 28-29*).

3.4 Implementation of the Critic and Actor

Before describing the architectures of the critic and actor, we firstly introduce how to derive the environment state $\boldsymbol{s}$. In an RL-based recommender system, the state concludes the user's current status. In our model, it is computed from the embeddings of a user and her previously interacted items. For a user u, suppose her interacted items are $\{i_1, i_2, ...i_{l_u}\}$, then the state $\boldsymbol{s}$ is computed as:

$$\boldsymbol{s} = \boldsymbol{p}_u + \sum_{m=1}^{l_u} \boldsymbol{q}_{i_m} \tag{9}$$

where $\boldsymbol{p}_u$ and $\boldsymbol{q}_{i_m}$ are the user and item embeddings[1], respectively.

In our critic, the state-action pair is transformed into a Q-value based on a two-layer neural network, that is,

$$\boldsymbol{Q}(\boldsymbol{s}, \boldsymbol{a}) = \boldsymbol{W}_2 \text{ReLU}(\boldsymbol{W}_1 \text{ReLU}(\boldsymbol{W}^Q[\boldsymbol{s}; \boldsymbol{a}]) + \boldsymbol{b}_1) + b_2, \tag{10}$$

where $\{\boldsymbol{W}_2, \boldsymbol{W}_1, \boldsymbol{b}_1, b_2, \boldsymbol{W}_s^Q, \boldsymbol{W}_a^Q\}$ are weighting parameters to be learned, ReLU is the activation function. $[\cdot;\cdot]$ is the concatenate

[1]Here, we use different embedding metrics for the critic and the actor

Algorithm 1: Pareto Deterministic Policy Gradient

```
1 Initialize Actor parameter $\boldsymbol{\theta}$ and Target Actor parameter $\boldsymbol{\theta}' \leftarrow \boldsymbol{\theta}$.
2 Initialize Critic parameter $\boldsymbol{\phi}_m$ and Target Critic parameter $\boldsymbol{\phi}'_m \leftarrow \boldsymbol{\phi}_m, \forall m \in [1,\ M]$.
3 Initialize Pareto weights $\boldsymbol{w} = \{\frac{1}{M+1}, \frac{1}{M+1}, ..., \frac{1}{M+1}\}$ and replay buffer $B$.
4 for episode number in [1, K] do
5     i) Trajectory Generation
6     Get start state $s_1$
7     for step t in [1, T] do
8         Select an action according to $a_t = \mu(s_t|\boldsymbol{\theta}) + N_t$, $N_t$ is an exploration noise.
9         Execute $a_t$ to obtain the new state $s_{t+1}$ and the reward vector $\boldsymbol{r}_t = \{r_{t,1}, r_{t,2}, ..., r_{t,M}\}$.
10        Push $\{s_t, a_t, \boldsymbol{r}_t, s_{t+1}\}$ into the replay buffer $B$
11    end
12    ii) Update Critic
13    Sample Z instances $\{s_i, a_i, \boldsymbol{r}_i, s_{i+1}\}$ from $B$
14    for critic m in [1, M] do
15        for i in [1, Z] do
16            Compute $y_i = r_{i,m} + \gamma Q_m(s_{i+1}, \mu(s_{i+1}|\boldsymbol{\theta}')|\boldsymbol{\phi}'_m)$.
17        end
18        $\boldsymbol{\phi}_m \leftarrow \boldsymbol{\phi}_m - \alpha_\phi \nabla_{\boldsymbol{\phi}_m}\{\frac{1}{Z}\sum_{i=1}^{Z}(y_i - Q_m(s_i, a_i|\boldsymbol{\phi}_m))^2\}$.
19    end
20    iii) Update Pareto Weight
21    for i in [1, Z] do
22        for m in [1, M] do
23            $\boldsymbol{p}_{i,m} = \nabla_a Q_m(s,a)|_{s=s_i, a=\mu_\theta(s_i)} \nabla_{\boldsymbol{\theta}}\mu(s|\boldsymbol{\theta})|_{s=s_i}$.
24        end
25        $\tilde{\boldsymbol{p}}_i = \nabla_a \tilde{Q}(s,a))|_{s=(o,y_i), a=\mu_\theta(s_i)} \nabla_{\boldsymbol{\theta}}\mu(s|\boldsymbol{\theta})|_{s=s_i}$.
26    end
27    Update $\boldsymbol{w} = \{w_1, w_2, ..., w_M, \tilde{w}\}$ by Solving (5).
28    iv) Update Actor
29    $\boldsymbol{d} = \frac{1}{Z}\sum_{i=1}^{Z}\sum_{m=1}^{M} w_m \boldsymbol{p}_{i,m} + \tilde{w}\tilde{\boldsymbol{p}}_i$.
30    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha_\theta \boldsymbol{d}$.
31    $\boldsymbol{\theta}' \leftarrow \tau\boldsymbol{\theta} + (1-\tau)\boldsymbol{\theta}'$.
32    $\boldsymbol{\phi}'_m \leftarrow \tau\boldsymbol{\phi}_m + (1-\tau)\boldsymbol{\phi}'_m \quad \forall m \in [1, M]$.
33 end
```

operation. $\boldsymbol{a}$ is the output from the actor (when updating the actor and computing the target Q) or the real item embedding (when updating the critic). In the actor, we project a state into an action in a deterministic manner, that is,

$$\mu(\boldsymbol{s}) = \boldsymbol{W}_4\text{ReLU}(\boldsymbol{W}_3\text{ReLU}(\boldsymbol{W}^A\boldsymbol{s}) + \boldsymbol{b}_3) + \boldsymbol{b}_4, \tag{11}$$

where $\{\boldsymbol{W}_4, \boldsymbol{W}_3, \boldsymbol{b}_4, \boldsymbol{b}_3, \boldsymbol{W}_s^A\}$ are model parameters. Once our model is learned, the final recommendations are generated by selecting the items whose embeddings are closer to the output of $\mu(\boldsymbol{s})$ [19].

3.5 Analysis of the Gradient Bias

In the above sections, we have described the implementation of our framework. Here, we provide some theoretical insights on the designed model. In the field of neural network optimization, gradient-based methods are very common and effective. In this section, we are interested in whether or not the gradients used in our model is biased from the true gradients, and if yes, how large is this bias.

Suppose we have many training batches $\{\boldsymbol{B}_1, \boldsymbol{B}_2, ...\}$, and each $\boldsymbol{B}_i$ is composed of Z samples. For the ith training step, $\boldsymbol{w}$ is derived based on $\boldsymbol{B}_i$ via problem (5). We re-denote $\boldsymbol{w}$ by $\boldsymbol{w}(\boldsymbol{B}_i)$ to highlight the relation between $\boldsymbol{w}$ and $\boldsymbol{B}_i$, and the loss[2] for each training batch $\boldsymbol{B}_i$ is:

$$\hat{l}(\boldsymbol{\theta}) = -\sum_{m=1}^{M+1} w_m(\boldsymbol{B}_i)\left(\frac{1}{Z}\sum_{s_b \in \boldsymbol{B}_i} Q_m(s_b, \mu(s_b|\boldsymbol{\theta}))\right) \tag{12}$$

For easy derivation, we denote $\nabla_{\boldsymbol{\theta}} Q_m(s, \mu(s|\boldsymbol{\theta}))$ by $f_m(s;\boldsymbol{\theta}) \in \mathbb{R}^d$, where $\boldsymbol{\theta}$ is assumed to be a d-dimensional vector. Let the true and mini-batch stochastic gradients be $\nabla_{\boldsymbol{\theta}} l(\boldsymbol{\theta})$ and $\nabla_{\boldsymbol{\theta}} \hat{l}(\boldsymbol{\theta})$, respectively. Then their discrepancy is represented as:

$$G = \left|\mathbb{E}_{\boldsymbol{B}_i}\Big[\sum_{m=1}^{M+1} w_m(\boldsymbol{B}_i)(\frac{1}{Z}\sum_{s_b \in \boldsymbol{B}_i} f_m(s_b;\boldsymbol{\theta}) - \mathbb{E}_s[f_m(s;\boldsymbol{\theta})])\Big]\right| \tag{13}$$

For G, we have the following theory:

Theorem 2. *Suppose i) $\nabla_a Q_m(s,a)$ and $\nabla_{\boldsymbol{\theta}}\mu(s|\boldsymbol{\theta})$ are bounded by X_m and Y, that is, $||\nabla_a Q_m(s,a)||_2 \le X_m$ and $||\nabla_{\boldsymbol{\theta}}\mu(s|\boldsymbol{\theta})||_2 \le Y$. ii) The batched gradient of the action-value function for each objective is unbiased, that is: $\mathbb{E}_{\boldsymbol{B}_i}[\frac{1}{Z}\sum_{s_b \in \boldsymbol{B}_i} f_m(s_b;\boldsymbol{\theta})] = \mathbb{E}_s[f_m(s;\boldsymbol{\theta})]$. iii) $f_m(s_b;\boldsymbol{\theta})$ follows a normal distribution $\mathcal{N}(\mathbb{E}_s[f_m(s;\boldsymbol{\theta})], \sigma^2 I)$, where $I \in \mathbb{R}^{d\times d}$ is an identity matrix and σ is a scalar. Then we have:*

$$G \le \sum_{m=1}^{M+1} \mathbb{E}_{\boldsymbol{B}_i}\Big[w_m(\boldsymbol{B}_i)(|\frac{1}{Z}\sum_{s_b \in \boldsymbol{B}_i} f_m(s_b;\boldsymbol{\theta}) - \mathbb{E}_s[f_m(s;\boldsymbol{\theta})]|)\Big] \tag{14}$$

$$\le \frac{X_{m*}Y\sqrt{d}}{\sqrt{Z}} \tag{15}$$

where $m^* = \arg\max_m |\frac{1}{Z}\sum_{s_b \in \boldsymbol{B}_i} f_m(s_b;\boldsymbol{\theta}) - \mathbb{E}_s[f_m(s;\boldsymbol{\theta})]|$.

Here, we present a scratch of the proof, and the complete version can be seen in the Appendix.

Proof. Since $w_m(\boldsymbol{B}_i) \ge 0$ and $\sum_{m=1}^{M+1} w_m(\boldsymbol{B}_i) = 1$, by Jensen's inequality, we can derive

$$G \le \mathbb{E}_{\boldsymbol{B}_i}\Big[||\frac{1}{Z}\sum_{s_b \in \boldsymbol{B}_i} f_{m^*}(s_b;\boldsymbol{\theta}) - \mathbb{E}_s[f_{m^*}(s;\boldsymbol{\theta})]||\Big]. \tag{16}$$

We denote $\hat{\boldsymbol{g}} = \frac{1}{Z}\sum_{s_b \in \boldsymbol{B}_i} f_{m^*}(s_b;\boldsymbol{\theta})$ and $\boldsymbol{g} = \mathbb{E}_s[f_m(s;\boldsymbol{\theta})]$, then $\hat{\boldsymbol{g}}$ and $\boldsymbol{g}$ are d-dimensional random variables, where the ith dimensions are defined as $\hat{g}_i$ and g_i, respectively. According to assumptions iii), we have $g_i = \mathbb{E}[\hat{g}_i]$ and $\sigma_g^2 = \mathbb{V}[\hat{g}_i] \le \frac{X_{m^*}^2 Y^2}{Z}$ Since we assume $\hat{g}_i$ follows a normal distribution, then $v_i = \frac{\hat{g}_i - g_i}{\sigma_g} \sim \mathcal{N}(0,1)$,

[2]As mentioned above, $L(\theta)$ can be seen as a special Q function, and we absorb it into the sum operation by allocating an additional Pareto weight.

and $\sum_{i=1}^{d} v_i^2$ follows Chi-square distribution $\tilde{\chi}_d^2$. Thus we have:

$$\mathbb{E}_{\boldsymbol{B}_i}\left[\|\frac{1}{Z}\sum_{s_b\in\boldsymbol{B}_i} f_{m^*}(s_b;\boldsymbol{\theta}) - \mathbb{E}_s[f_{m^*}(s;\boldsymbol{\theta})]\|\right] \tag{17}$$

$$\leq \left(\mathbb{E}[\sum_{i=1}^{d}(\hat{g}_i - g_i)^2]\right)^{\frac{1}{2}} = \sqrt{d}\sigma_g \leq \frac{X_{m*}Y\sqrt{d}}{\sqrt{Z}} \tag{18}$$

□

From this theory, we can see, the gradients used in our model is biased, and the bias's upper bound is in inverse proportion to the batch size. This suggests that if we use a larger batch size, the upper bound of G can be lowered, and we may potentially learn more accurate parameters. However, larger batch size means more cost on the computational resources (e.g., GPU memory). If one looks deeper into this theory, she may find that If w_m is not related with $\boldsymbol{B}_i$, then $w_m(\boldsymbol{B}_i)$ can be removed out of the expectation in equation (14). At this moment, G becomes 0, given that $\frac{1}{Z}\sum_{s_b\in\boldsymbol{B}_i} f_m(s_b;\boldsymbol{\theta})$ is an unbiased estimator of $\mathbb{E}_s[f_m(s;\boldsymbol{\theta})]$. Inspired by this phenomenon, we design the following "weight-reuse" mechanism.

Weight-reuse mechanism. In this method, we introduce a container $\boldsymbol{W} \in \mathbb{R}^{L\times(M+1)}$ for storing previously derived Pareto weights. For each training batch $\boldsymbol{B}_i$, $\boldsymbol{w} \in \mathbb{R}^{M+1}$ is not always computed by solving problem (5). We firstly check the weights in the container:

(1) If there is a candidate $\boldsymbol{w}^* \in \boldsymbol{W}$, such that its corresponding $\boldsymbol{d}^* = \sum_{m=1}^{M+1} w_m^* \nabla_{\boldsymbol{\theta}}\left(\frac{1}{Z}\sum_{s_b\in\boldsymbol{B}_i} Q_m(s_b,\mu(s_b|\boldsymbol{\theta}))\right)$ can increase all the Q functions, that is, $(\boldsymbol{d}^*)^T\nabla_{\boldsymbol{\theta}}\left(\frac{1}{Z}\sum_{s_b\in\boldsymbol{B}_i} Q_m(s_b,\mu(s_b|\boldsymbol{\theta}))\right) > 0, \forall m \in [1, M+1]$, then we set $\boldsymbol{w} = \boldsymbol{w}^{*}$[3]. Since the weights in $\boldsymbol{W}$ is not derived from $\boldsymbol{B}_i$, the bias G becomes 0 at this moment.

(2) If there is no such weight in $\boldsymbol{W}$, we solve problem (5) to derive $\boldsymbol{w}$, which is then pushed into the container for future "reuse". In this scenario, G is not 0, which is bounded by equation (15).

From the above analysis, we can see, under the weight-reuse mechanism, the bias G has some chances to become 0, thus the overall upper bound is lowered. In practice, the container size is fixed as L, and the earliest weights will be moved out when the container is full. The instances for deriving the weights in $\boldsymbol{W}$ are temporally frozen to avoid of being sampled into $\boldsymbol{B}_i$, and causing dependency[4].

4 RELATED WORK

4.1 RL-based Recommender Models

Users' long-term engagement in a recommender system has recently attracted increasing attention [23–25]. To capturing such information, reinforcement learning, as a powerful tool for balancing short- and long-term rewards, has became an interesting framework for building recommender models. Previously, many models focus on designing effective agents to generate accurate recommendations. For example, [20] proposes a GRU based model to capture user historical behaviors, which is then incorporated into the DQN framework. [22] also bases itself on DQN, but further involves more contextual features and continuous time information. [7, 19, 21] build their model based on DPG. Since the item set (action space) can be very large in real-world recommender systems, these models can be more efficient when learning the agent. Meanwhile, many models are proposed to build better user environments for providing more reliable rewards. For example, [2] explicitly builds a model to simulate user decision process, and the user model and recommender agent are jointly learned. [1] leverages model-based RL to formulate the recommendation task, where the user environment is explicitly learned to accommodate the recommender agent. Existing models mainly focus on how to design effective agents or environments, while little effort has been devoted to studying the rewards, which is important for understanding the users and learning more accurate recommendation policies. In our work, we take a step towards more comprehensive user reward shaping, where we explicitly model the users' diverse preferences on different item aspects.

4.2 Pareto Optimization

In many real-world problems, the machine learning models usually need to simultaneously optimize multiply objectives. Different objectives may not always consistent with each other, and the optimal parameter for one objective may not perform well on the other ones. In such a scenario, Pareto optimization provides a reasonable method to trade-off different objectives. In specific, under the supervised learning framework, [11] leverages multi-objective optimization technique to solve the multi-task learning problem. [6] extends [11] by adding preference vectors for generating more evenly distributed Pareto frontier. Many efforts have also been devoted to applying Pareto optimization to enhance the reinforcement learning framework. Typically, many models [8, 14] study how to design multi-objective DQNs based on heuristic Pareto optimization strategies. These methods have achieved promising results for the problems with small and discrete action sets. However, little attention has been paid to the extremely large or continuous action spaces, which is yet important for real-world applications. In this paper, we fill in this gap by extending DPG with multi-objective rewards, and more importantly, we theoretically analyze the designed model by presenting and lowering the upper bound of the gradient bias.

5 EXPERIMENTS

In this section, we conduct extensive experiments to demonstrate the effectiveness of our model, where we focus on the following research questions:

RQ1: Whether our model can outperform the state-of-the-art methods?

RQ2: How does different components in our model contribute the final results?

RQ3: How different hyper-parameters influence our model's performance?

We begin with the experiment setup, and then present and analyze the results to answer the above questions.

5.0.1 Experiment Setup. **Datasets.** We base our experiments on three real-world datasets including RateBeer, BeerAdvocate[5] and TripAdvisor[6]. RateBeer and BeerAdvocate contain user ratings on

[3]If there are multiple $\boldsymbol{w}^*$'s, we sample one from them.

[4]Here, we assume that different samples are independent

[5]RateBeer and BeerAdvocate are from https://cseweb.ucsd.edu/jmcauley/datasets.html

[6]http://www.cs.virginia.edu/ hw5x/dataset.html

Table 1: Statistics of the datasets. #Asp indicates the number of aspects.

Dataset	#User	#Item	#Asp	#Interaction	Density
RateBeer	6717	20453	5	2345477	1.70%
BeerAdvocate	10701	14228	5	1386424	0.91%
TripAdvisor	1657	5206	7	28578	0.33%

different beers, while TripAdvisor is a travel dataset including the hotel ratings from the customers. In all these datasets, in addition to an *overall* rating, we also have users' ratings on different item-aspects. For each beer in RateBeer and BeerAdvocate, people are allowed to make ratings on its *appearance*, *aroma*, *palate*, and *taste*. For the hotels in TripAdvisor, we have user ratings on the *service*, *cleanliness*, *value*, *sleep quality*, *rooms, and location*. The ratings of these datasets are scaled into the range of $[1, 10]$. The statistics of these datasets are summarized in Table 1. We can see these datasets cover different characters, *e.g.*, TripAdvisor is small and sparse, while RateBeer is much larger and denser. Based on these diverse datasets, our model can be evaluated under different settings in a comprehensive manner.

Baselines. We select the following representative methods as our baselines:

• **BPR [10]:** This is a well-known recommender method for modeling user implicit feedback. We use matrix factorization as its predictive function in the experiments.

• **NCF [3]:** This is a state-of-the-art deep recommender model, where the user and item representations are fed into multiple non-linear layers to predict the final results.

• **EFM [17]:** This is a well known explainable recommender model, where the user preference on different item aspects is incorporated into the matrix factorization method.

• **MATF [5]:** This is a multi-aspect recommendation model based on tensor factorization, where we optimize it based on the pair-wise BPR loss for fair comparison.

• **GRU4Rec [4]:** This is a sequential recommender model, where the interacted items are modeled by a recurrent neural network.

• **DRR [7]:** This is a recently proposed RL-based recommender model, where the overall rating of each user-item pair is regarded as the reward.

Environment simulation. Ideally, a model should be trained and evaluated in an on-line recommender systems to get real user rewards. However, unlike classical RL problems (*e.g.*, playing Atari Games), where we can interact with the environment and obtain the reward with little effort, it is costly and not safe to directly deploy an immature RL model onto real-world systems [2, 7]. Thus, we follow previous works [2, 7, 21] to build simulators for approximating the user reward generation process. In general, the simulator should well balance the trade-off between the simplicity (efficiency) and performance (effectiveness), such that our RL model can be trained with acceptable speed and accuracy. Our simulator is designed as a two-layer fully connected neural network with ReLu as the activation function. The input is a state-action pair, and the outputs are the estimated ratings for different item aspects. The user simulator is learned based on the training and validation sets, and the average rating prediction performance is satisfied in terms of RMSE, which is about 0.96 for RateBeer and BeerAdvocate, and 0.91 for TripAdvisor, respectively.

Implementation details. Following the common practice [18], we chronologically organized each user's interacted items as a sequence in the beginning. And then, we split the sequence when the time interval between successive interactions is larger than some threshold, which results in many shorter but more coherent sessions. We leverage each user's last 30% sessions as the testing set, while the others are left for training. The commonly used metrics including F_1@5, NDCG@5 and Cumulated Reward (Cum-Reward) are adopted to evaluate our models. Among these metrics, F_1 aims to measure the overlap between the recommended items and the ground truth. NDCG is a ranking-based metric, and a hit with higher ranked prediction contributes more to the final results. Cumulated reward is utilized to evaluate the users' long-term satisfaction, and we report the sum of the rewards for each aspect in the testing episodes. In our model, we leverage stochastic gradient decent (SGD) to optimize the parameters, and the learning rates for the actor and critic are determined in $[0.0001, 0.001, 0.01, 0.1]$. The batch size and discount factor γ are tuned in $[32, 64, 128, 256, 512, 1024]$ and $[0.1, 0.3, 0.5, 0.7, 0.9]$, respectively. The container size of the weight-reuse mechanism is selected in the range of $[1, 2, 3, 4, 5]$.

*5.0.2 Overall Comparison (**RQ1**).* The overall comparison between our model and the baselines are presented in Table 2, from which we can see:

• On different datasets, NCF, EFM and MATF perform better than BPR in most cases, which agrees with the previous work [3, 17]. The reasons can be that NCF can leverage neural networks to model non-linear user-item relationships, and EFM and MATF are able to incorporate user multi-aspect preference into their modeling process. As a result, they both exhibit better performance than BPR.

• It is interesting to see that the sequential model GRU4Rec did not achieve superior performance than the non-sequential ones. We speculate that the sequential characters of our datasets are not significant, the users may comment on the beers or hotels in a quite random manner. Leveraging recurrent architectures, such as GRU, to model our data may impose too strong assumptions, which may lead to unsatisfied performance.

• DRR can usually obtain larger cumulated rewards than the other baselines, which verifies the capability of RL for modeling users' long-term engagement on the recommendation task. For the other cases, DRR does not perform very well, which may imply that maximizing the expected Q values do not always align with the accuracy-based metrics, such as F1 and NDCG.

• Encouragingly, by incorporating multi-objective rewards into the DPG framework, our model can achieve the best performance on all the metrics across all the datasets. This observation demonstrates the effectiveness of our model, which positively answers the first research question. Comparing with DRR, the modeling of multi-aspect preference enables us to more comprehensively profile the users, and incorporated supervised regularizer compensates the Q-function optimization by constraining the generated actions into a safe space. Both of these designs help to better understand the users and improve the final recommendation performance. Comparing with the other baselines, which only optimize the users' immediate preference, our model can appropriately trade off the short- and

Table 2: Performance comparison between the baselines and our model. For each metric on different datasets, we use bold fonts and * to label the best performance and the best baseline performance, respectively. Impr. is short for improvement, and the last column shows the relative improvement of our results against the best baseline. BeerAd and TripAd are short for the datasets of BeerAdvocate and TripAdvisor. The aspects are abbreviated as the capitalization of their first two letters,*e.g.*, AP is short for *appearance*.

Dataset			BPR	NCF	EFM	MATF	GRU4Rec	DRR	PDPG	Impr.
RateBeer	F_1@5		0.221	0.246*	0.241	0.235	0.234	0.221	**0.258**	4.94%
	NDCG@5		0.502	0.547*	0.532	0.521	0.519	0.517	**0.586**	7.07%
	Cum-Reward	AP	392.15	384.41	387.93	391.43	395.12	412.22*	**434.99**	5.52%
		AR	379.47	367.21	378.36	382.12	398.18	405.13*	**415.42**	2.54%
		PA	379.01	369.64	377.11	381.91	389.36	401.22*	**422.20**	5.23%
		TA	383.97	377.43	385.60	390.01	394.05	404.36*	**421.49**	4.23%
		OV	388.46	377.24	387.12	391.78	399.17	412.77*	**428.25**	3.75%
BeerAd	F_1@5		0.223	0.261	0.269*	0.258	0.251	0.254	**0.281**	4.42%
	NDCG@5		0.552	0.603*	0.595	0.582	0.574	0.572	**0.651**	7.99%
	Cum-Reward	AP	371.51	382.61	396.52	391.32	399.61	401.14*	**420.81**	4.90%
		AR	359.81	388.16	386.41	384.51	387.99	396.46*	**419.57**	5.83%
		PA	361.72	382.91	382.12	380.22	397.44	400.13*	**420.93**	5.20%
		TA	369.81	384.41	392.71	391.40	396.78	401.29*	**430.72**	7.33%
		OV	366.69	372.42	381.41	379.33	385.31	389.72*	**424.55**	8.93%
TripAd	F_1@5		0.193	0.219	0.259*	0.244	0.217	0.226	**0.283**	9.49%
	NDCG@5		0.519	0.523	0.578*	0.561	0.533	0.551	**0.619**	7.13%
	Cum-Reward	SE	401.15	409.33	402.25	401.12	412.13	431.12*	**458.80**	6.42%
		CL	413.69	421.21	409.14	407.35	411.55	446.66*	**472.64**	5.81%
		VA	401.14	411.34	404.77	402.39	415.14	421.22*	**447.89**	6.33%
		SL	402.56	439.20*	391.35	389.49	401.34	432.90	**464.41**	5.74%
		LO	398.09	407.44	402.42	400.11	414.17	429.31*	**448.27**	4.41%
		RO	403.53	409.23	406.41	403.54	431.19	467.21*	**489.47**	7.43%
		OV	401.76	412.01	405.92	402.37	407.93	439.31*	**473.69**	7.82%

long-term user engagements, which leads to superior results on different metrics.

*5.0.3 Ablation Study (**RQ2**).* In the above section, we have evaluated our model as a whole. In order to verify whether different model components are useful for the final result, we conduct ablation studies in this section. In the experiments, the model parameters are fixed as the optimal values, and the performance is evaluated based on F_1@5 and NDCG@5, respectively. We are interested in the following questions: (1) Whether Pareto optimization is necessary? (2) Whether the weight-reuse mechanism is benefit for the performance? (3) Whether the supervised regularizer can improve the evaluation results? (4) Whether the Q-function can lead to better actor optimization?

For answering these questions, we compare our model with its five variants, that is, (i) PDPG (random pooling): in this method, different Q-functions are merged by a set of random weights. (ii) PDPG (average pooling): in this method, we directly average different Q-functions. In both PDPG (random pooling) and PDPG (average pooling), the weights for different Q functions are fixed in the optimization process. (iii) PDPG (−*reuse*): in this method, we drop the weight-reuse mechanism. (iv) PDPG (−*super*): in this method, we do not use the supervised regularizer (i.e., equation (8)), and the actor is solely optimized based on the Q-functions. (v) PDPG (−*Q*): in this method, we drop the Q-functions, and only use equation (8) to learn the actor in a supervised manner. We present the comparison results in Table 3.

• We can see, the winner and performance gap between PDPG (random pooling) and PDPG (average pooling) varies across different datasets, e.g., on RateBeer, PDPG (random pooling) shows slightly inferior performance than PDPG (average pooling), while on TripAdvisor, PDPG (random pooling) outperforms PDPG (average pooling) by a considerable margin. This result manifests that the weights for merging different Q functions can be data dependent, and we may need to search a large space to determine their optimal values. An encouraging observation is that our final model can consistently achieve better performance than both of these variants. This observation verifies the effectiveness of introducing Pareto optimization to coordinate different learning objectives. Based on the weights derived from problem (5), all the targets are continually optimized along a non-decreasing direction, which is shown to be effective in promoting the final recommendation performance.

• Comparing with the final model, if we drop the weight-reuse mechanism, the performance is lowered on all the datasets. This result is as expected, and positively answers question (2). An interesting observation is that, in some cases (e.g., on RateBeer), despite leveraging Pareto optimization, PDPG (−*reuse*) performs worse than the fixed weight models, which manifests that the simple Pareto optimization method may not necessarily bring improved performances. As mentioned in theory 2, the parameter gradients

Table 3: Comparison between our model and its variants. We use bold fonts to label the best performance. PDPG (ran) and PDPG (ave) are short for PDPG (random pooling) and PDPG (average pooling), respectively. For saving the space, we omit the recommendation number "@5".

Models	RateBeer		BeerAd		TripAd	
	F_1	NDCG	F_1	NDCG	F_1	NDCG
PDPG (ran)	0.252	0.551	0.269	0.604	0.205	0.456
PDPG (ave)	0.253	0.574	0.266	0.572	0.164	0.294
PDPG ($-reuse$)	0.238	0.468	0.270	0.558	0.281	0.607
PDPG ($-super$)	0.216	0.491	0.268	0.575	0.158	0.280
PDPG ($-Q$)	0.243	0.559	0.266	0.598	0.251	0.567
PDPG	**0.258**	**0.586**	**0.281**	**0.651**	**0.283**	**0.619**

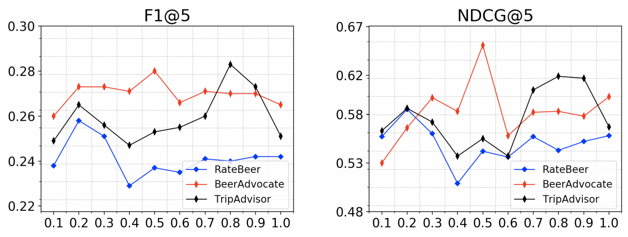

Figure 2: Importance of the supervised regularizer.

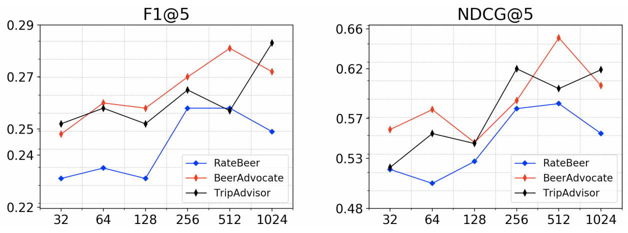

Figure 3: Influence of the batch size on the model performance.

are biased after introducing the Pareto optimization, which may negatively impact the actor learning process. As a remedy, we design the weight-reuse mechanism to lower the upper bound of the bias, such that the actor optimization can be more aligned with the true gradients, which is shown effective by the superior performance of the final model.

• For questions (3) and (4), we can see neither PDPG ($-super$) nor PDPG ($-Q$) can outperform the complete model, and the results are consistent across all the datasets. These observations imply that both the Q function and supervised regularizer are necessary for the final better performance, which verifies our claims in previous sections. The balance between these components are also studied by tunning the hyper-parameters in the following experiments.

*5.0.4 Parameter Analysis (**RQ3**).* Different hyper-parameters may influence the model performance with various sensibilities. In this section, we firstly study the importance of the supervised regularizer, and then we investigate the influences of the batch size and discount factor for the final results, respectively. When studying one parameter, we fixed the other ones as their optimal values.

Study on the importance of the supervised regularizer. By the supervised objective (8), we aim to regularize the action into a safe space, which is not far from the users' real preference. In this section, we study the importance of this supervised regularizer for the final performance. More specifically, we constrain the weight of the regularizer $\tilde{w}$ by imposing different preference vectors and values. For example, if we want set the importance level as 0.1, then the preference vector and value is set as $[0, 0, 0, 0, 0, 1]$ and 0.1^7, respectively. We tune the importance level in the range of $[0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]$, and the results are presented in Figure 2. We can see, the optimal importance level is small for the datasets of RateBeer and BeerAdvocate, while larger for TripAdvisor. Considering that TripAdvisor is a much sparser dataset, the optimization of the Q functions can be highly insufficient. In such a scenario, we speculate that the supervision signals can be more needed for compensating the Q function learning, and thus lead to better performance when the importance level is higher.

Study on the batch size. Batch size is an important hyper-parameter for training the neural models. In this section, we tune the batch size in the range of $[32, 64, 128, 256, 512, 1024]$, and the performance is evaluated based on F1@5 and NDCG@5, respectively. From the results shown in Figure 3, we can see: the best results are usually achieved when the batch size is relatively large. This observation agrees with theory 2, that is, larger batch size can lower the upper bound of the gradient bias, which may potentially correct the gradient error, and improve the model performance.

Study on the discount factor. In the context of RL-based recommender system, γ is used to balance the short- and long-term rewards. Smaller γ pays more attention to the users' immediate preference, while larger γ puts more focus on the future engagement. In this experiment, we study the influence of γ by tunning it in the range of $[0.1, 0.3, 0.5, 0.7, 0.9]$. The results are presented in Figure 4. We find that the optimal γ for different datasets varies. For example, on TripAdvisor and RateBeer, smaller γ can lead to better results, while on BeerAdvocate, a moderate γ is more preferred. This observation suggests that γ is sensitive to the dataset, and should be well tunned in practice for achieving the best performance.

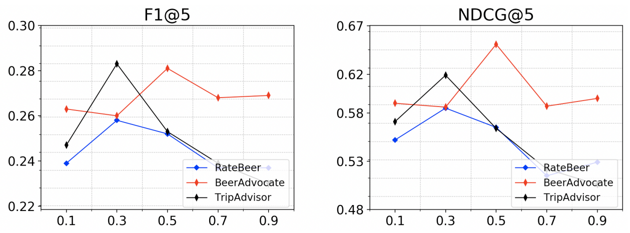

Figure 4: Influence of gamma on the model performance.

[7]Here, we suppose there are five aspects.

6 CONCLUSION

In this paper, we propose to capture user multi-aspect preferences in the context of RL-based recommender system. To this end, we extend traditional deterministic policy gradient with multi-objective rewards, and seamlessly infuse Pareto optimization into the modeling process. We provide a theoretical analysis on the designed framework, and also propose a mechanism to correct the gradient bias. To demonstrate our model's effectiveness, extensive experiments are conducted based on the real-world datasets.

Different from previous RL-based recommender models, which mostly focus on the design of the agent or environment, this paper opens the door for modeling complex or even conflict user rewards. We believe there is still much room for improving this work. To begin with, we can make a more thorough study on the preference vectors and values, based on which we plan to propose the concept of "aspect-level" fairness, that is, we should try to equally optimize different aspects in the training process. In addition, we may also design more advanced weight-reuse mechanisms, such that we can find the optimal Pareto weights more efficiently.

ACKNOWLEDGMENTS

This work is supported in part by Beijing Outstanding Young Scientist Program NO. BJJWZYJH012019100020098 and National Natural Science Foundation of China (No. 61832017).

7 APPENDIX

7.1 Complete Proof of Theorem 2

Proof. According to the Jensen's inequality, we have $|\mathbb{E}[X]| \leq \mathbb{E}[|X|]$. We also know $w_m(\boldsymbol{B}_i) \geq 0$ and $\sum_{m=1}^{M+1} w_m(\boldsymbol{B}_i) = 1$, thus:

$$G = \left\|\mathbb{E}_{\boldsymbol{B}_i}\Big[\sum_{m=1}^{M+1} w_m(\boldsymbol{B}_i)\Big(\frac{1}{Z}\sum_{s_b \in \boldsymbol{B}_i} f_m(s_b;\boldsymbol{\theta}) - \mathbb{E}_s[f_m(s;\boldsymbol{\theta})]\Big)\Big]\right\| \quad (19)$$

$$\leq \sum_{m=1}^{M+1} \mathbb{E}_{\boldsymbol{B}_i}\Big[w_m(\boldsymbol{B}_i)\Big(\|\frac{1}{Z}\sum_{s_b \in \boldsymbol{B}_i} f_m(s_b;\boldsymbol{\theta}) - \mathbb{E}_s[f_m(s;\boldsymbol{\theta})]\|\Big)\Big] \quad (20)$$

$$\leq \sum_{m=1}^{M+1} \mathbb{E}_{\boldsymbol{B}_i}\Big[w_m(\boldsymbol{B}_i)\Big(\|\frac{1}{Z}\sum_{s_b \in \boldsymbol{B}_i} f_{m^*}(s_b;\boldsymbol{\theta}) - \mathbb{E}_s[f_{m^*}(s;\boldsymbol{\theta})]\|\Big)\Big] \quad (21)$$

$$= \mathbb{E}_{\boldsymbol{B}_i}\Big[\|\frac{1}{Z}\sum_{s_b \in \boldsymbol{B}_i} f_{m^*}(s_b;\boldsymbol{\theta}) - \mathbb{E}_s[f_{m^*}(s;\boldsymbol{\theta})]\|\Big] \quad (22)$$

(20) holds because of the Jensen's inequality. (21) holds because of the definition that $m^* = \arg\max_m |\frac{1}{Z}\sum_{s_b \in \boldsymbol{B}_i} f_m(s_b;\boldsymbol{\theta}) - \mathbb{E}_s[f_m(s;\boldsymbol{\theta})]|$, and (22) holds since $\sum_m w_m(B_i) = 1$.

We denote $\hat{\boldsymbol{g}} = \frac{1}{Z}\sum_{s_b \in \boldsymbol{B}_i} f_{m^*}(s_b;\boldsymbol{\theta})$ and $\boldsymbol{g} = \mathbb{E}_s[f_m(s;\boldsymbol{\theta})]$, then $\hat{\boldsymbol{g}}$ and $\boldsymbol{g}$ are d-dimensional random variables, where the ith dimensions are defined as $\hat{g}_i$ and g_i, respectively.

According to assumptions iii) in Theorem 2, we have

$$g_i = \mathbb{E}[\hat{g}_i] \quad (23)$$

$$\sigma_g^2 = \mathbb{V}[\hat{g}_i] \leq \mathbb{E}[\hat{g}_i^2] \quad (24)$$

$$\leq \mathbb{E}[||\frac{1}{Z}\sum_{s_b \in \boldsymbol{B}_i} \nabla_a Q_{m^*}(s_b, a)\nabla_{\boldsymbol{\theta}}\mu(s_b|\boldsymbol{\theta})||_2^2]] \quad (25)$$

$$\leq \frac{X_{m^*}^2 Y^2}{Z}. \quad (26)$$

Since we assume $\hat{g}_i$ follows a normal distribution in Theorem 2, we have:

$$v_i = \frac{\hat{g}_i - g_i}{\sigma_g} \sim \mathcal{N}(0,1). \quad (27)$$

Thus $\sum_{i=1}^{d} v_i^2$ follows Chi-square distribution $\tilde{\chi}_d^2$, and we have $\mathbb{E}[\sum_{i=1}^{d} v_i^2] = d$, then:

$$\mathbb{E}_{\boldsymbol{B}_i}\Big[\|\frac{1}{Z}\sum_{s_b \in \boldsymbol{B}_i} f_{m^*}(s_b;\boldsymbol{\theta}) - \mathbb{E}_s[f_{m^*}(s;\boldsymbol{\theta})]\|\Big] \quad (28)$$

$$= \mathbb{E}\big[\|\hat{\boldsymbol{g}} - \boldsymbol{g}\|\big] = \mathbb{E}\Big[\Big(\sum_{i=1}^{d} ||\hat{g}_i - g_i||_2^2\Big)^{\frac{1}{2}}\Big] \quad (29)$$

$$\leq \left(\mathbb{E}\Big[\sum_{i=1}^{d}(\hat{g}_i - g_i)^2\Big]\right)^{\frac{1}{2}} = \sqrt{d}\sigma_g \leq \frac{X_{m*}Y\sqrt{d}}{\sqrt{Z}} \quad (30)$$

□

REFERENCES

[1] Xueying Bai, Jian Guan, and Hongning Wang. 2019. A Model-Based Reinforcement Learning with Adversarial Training for Online Recommendation. In *Advances in Neural Information Processing Systems.* 10735–10746.
[2] Xinshi Chen, Shuang Li, Hui Li, Shaohua Jiang, Yuan Qi, and Le Song. 2018. Generative adversarial user model for reinforcement learning based recommendation system. *arXiv preprint arXiv:1812.10613* (2018).
[3] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web.* 173–182.
[4] Balázs Hidasi and Alexandros Karatzoglou. 2018. Recurrent neural networks with top-k gains for session-based recommendations. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management.* 843–852.
[5] Nurkhairizan Khairudin, Nurfadhlina Mohd Sharef, Norwati Mustapha, and Shahrul Azman Mohd Noah. 2018. Enhancing Multi-Aspect Collaborative Filtering for Personalized Recommendation. In *2018 Fourth International Conference on Information Retrieval and Knowledge Management (CAMP).* IEEE, 1–6.
[6] Xi Lin, Hui-Ling Zhen, Zhenhua Li, Qing-Fu Zhang, and Sam Kwong. 2019. Pareto Multi-Task Learning. In *Advances in Neural Information Processing Systems.* 12037–12047.
[7] Feng Liu, Ruiming Tang, Xutao Li, Weinan Zhang, Yunming Ye, Haokun Chen, Huifeng Guo, and Yuzhou Zhang. 2018. Deep reinforcement learning based recommendation with explicit user-item interactions modeling. *arXiv preprint arXiv:1810.12027* (2018).
[8] Thanh Thi Nguyen. 2018. A multi-objective deep reinforcement learning framework. *arXiv preprint arXiv:1803.02965* (2018).
[9] Marcus O'Dair and Andrew Fry. 2020. Beyond the black box in music streaming: the impact of recommendation systems upon artists. *Popular Communication* 18, 1 (2020), 65–77.
[10] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2012. BPR: Bayesian personalized ranking from implicit feedback. *arXiv preprint arXiv:1205.2618* (2012).
[11] Ozan Sener and Vladlen Koltun. 2018. Multi-task learning as multi-objective optimization. In *Advances in Neural Information Processing Systems.* 527–538.
[12] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. 2014. Deterministic policy gradient algorithms.
[13] Peijie Sun, Le Wu, and Meng Wang. 2018. Attentive recurrent social recommendation. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval.* 185–194.

[14] Tomasz Tajmajer. 2018. Modular multi-objective deep reinforcement learning with decision values. In *2018 Federated conference on computer science and information systems (FedCSIS)*. IEEE, 85–93.

[15] Jizhe Wang, Pipei Huang, Huan Zhao, Zhibo Zhang, Binqiang Zhao, and Dik Lun Lee. 2018. Billion-scale commodity embedding for e-commerce recommendation in alibaba. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 839–848.

[16] Lu Wang, Wei Zhang, Xiaofeng He, and Hongyuan Zha. 2018. Supervised reinforcement learning with recurrent neural network for dynamic treatment recommendation. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2447–2456.

[17] Yongfeng Zhang, Guokun Lai, Min Zhang, Yi Zhang, Yiqun Liu, and Shaoping Ma. 2014. Explicit factor models for explainable recommendation based on phrase-level sentiment analysis. In *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval*. 83–92.

[18] Wayne Xin Zhao, Shanlei Mu, Yupeng Hou, Zihan Lin, Kaiyuan Li, Yushuo Chen, Yujie Lu, Hui Wang, Changxin Tian, Xingyu Pan, et al. 2020. RecBole: Towards a Unified, Comprehensive and Efficient Framework for Recommendation Algorithms. *arXiv preprint arXiv:2011.01731* (2020).

[19] Xiangyu Zhao, Long Xia, Liang Zhang, Zhuoye Ding, Dawei Yin, and Jiliang Tang. 2018. Deep reinforcement learning for page-wise recommendations. In *Proceedings of the 12th ACM Conference on Recommender Systems*. 95–103.

[20] 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*. 1040–1048.

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

[22] Guanjie Zheng, Fuzheng Zhang, Zihan Zheng, Yang Xiang, Nicholas Jing Yuan, Xing Xie, and Zhenhui Li. 2018. DRN: A deep reinforcement learning framework for news recommendation. In *Proceedings of the 2018 World Wide Web Conference*. 167–176.

[23] Lixin Zou, Long Xia, Zhuoye Ding, Jiaxing Song, Weidong Liu, and Dawei Yin. 2019. Reinforcement learning to optimize long-term user engagement in recommender systems. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2810–2818.

[24] Lixin Zou, Long Xia, Pan Du, Zhuo Zhang, Ting Bai, Weidong Liu, Jian-Yun Nie, and Dawei Yin. 2020. Pseudo Dyna-Q: A reinforcement learning framework for interactive recommendation. In *Proceedings of the 13th International Conference on Web Search and Data Mining*. 816–824.

[25] Lixin Zou, Long Xia, Yulong Gu, Xiangyu Zhao, Weidong Liu, Jimmy Xiangji Huang, and Dawei Yin. 2020. Neural Interactive Collaborative Filtering. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 749–758.