Large-scale Interactive Conversational Recommendation System using Actor-Critic Framework

Ali Montazeralghaem
University of Massachusetts Amherst
Amherst, USA
montazer@cs.umass.edu

James Allan
University of Massachusetts Amherst
Amherst, USA
allan@cs.umass.edu

Philip S. Thomas
University of Massachusetts Amherst
Amherst, USA
pthomas@cs.umass.edu

ABSTRACT

We propose AC-CRS, a novel conversational recommendation system based on reinforcement learning that better models user interaction compared to prior work. Interactive recommender systems expect an initial request from a user and then iterate by asking questions or recommending potential matching items, continuing until some stopping criterion is achieved. Unlike most existing works that stop as soon as an item is recommended, we model the more realistic expectation that the interaction will continue if the item is not appropriate. Using this process, AC-CRS is able to support a more flexible conversation with users. Unlike existing models, AC-CRS is able to estimate a value for each question in the conversation to make sure that questions asked by the agent are relevant to the target item (i.e., user needs). We also model the possibility that the system could suggest more than one item in a given turn, allowing it to take advantage of screen space if it is present. AC-CRS also better accommodates the massive space of items that a real-world recommender system must handle. Experiments on real-world user purchasing data show the effectiveness of our model in terms of standard evaluation measures such as NDCG.

ACM Reference Format:
Ali Montazeralghaem, James Allan, and Philip S. Thomas. 2021. Large-scale Interactive Conversational Recommendation System using Actor-Critic Framework. In *Fifteenth ACM Conference on Recommender Systems (RecSys '21), September 27-October 1, 2021, Amsterdam, Netherlands.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3460231.3474271

1 INTRODUCTION

Conversational product search can be modeled by the following process. First, a user specifies their interest in a product using a query ("mobile phone"). The system either asks a question to clarify or drill down ("Are you interested in a phone plan or buying a phone?" or "Would you prefer Android, iOS, or something else?") or suggests some possible items for purchase ("How about a blue Galaxy S10?") from its massive product catalogue. This process continues until the search is successful or the user accepts defeat.

The goal of such a system is to help the user find an item of interest as rapidly as it can, using dialogue to winnow the set of candidate items to something manageable as quickly as possible: questions that ultimately turn out to be useless should be avoided while those that move rapidly toward an answer are to be encouraged; suggesting candidate items prematurely wastes time when they are off target, yet it may turn out that showing a candidate early may provide fruitful negative feedback.

Several studies have looked at this problem recently [1, 3, 14, 28, 34]. Each modeled the problem slightly differently in order to develop models of increasing capability. For example, Zhang et al. [34] used a simplified interaction model where the system asks questions and eventually stops with a single suggested item (they describe the possibility of continuing after that first recommendation but do not explicitly explore the ongoing interaction as part of their work). Bi et al. [3] proposed a conversational paradigm for product search to consider negative feedback in this task. However, they force the system to ask a question *and* recommend an item in each round of the conversation. Sun et al. [28] introduce a conversational recommender system based on reinforcement learning. The tension between immediate payoff and potentially delayed value of a question or suggestion makes reinforcement learning [30] an appropriate framework for this task. They learn to track the user's belief with reinforcement learning but allowed only a single recommendation. Furthermore, their model cannot jointly learn the dialogue policy and the recommendation model at the same time.

In this study, we develop and evaluate AC-CRS,[1] a novel conversational recommendation system that captures the richer task described above, where recommendation failures do not halt the process, where multiple simultaneous recommendations are possible, and where the massive item space suggests different selection steps.

In order to accommodate choosing among a large number of potential questions or items, AC-CRS adopts an end-to-end Actor-Critic model. We propose a tree-structured Actor which reduces the time complexity in the training. As a result, the model can continuously update its strategies during the interactions, until the system converges to the optimal strategy for each specific user (personalized question and recommendation). In contrast to prior work, our model can simultaneously maximize users' long-term preferences in asking questions and making recommendations. Since the actual value of each question for a given target item is unknown, the AC-CRS model predicts a specific value (the output of critic – i.e., the action value) for them during the conversation which helps the Actor to ask appropriate questions to satisfy users. Asking high quality and relevant questions is an important task in conversational recommendation systems that we address here.

[1] Actor-Critic Conversational Recommendation System

RecSys '21, September 27-October 1, 2021, Amsterdam, Netherlands

ACM ISBN 978-1-4503-8458-2/21/09…$15.00
https://doi.org/10.1145/3460231.3474271

The AC-CRS model and our evaluation approach allow an agent to recommend one *or more* items or ask a question at each step of the conversation. AC-CRS interacts with the user to get feedback over the taken action. By modeling a conversational recommender system in this way, we let the model more completely explore questions and items to improve the user's satisfaction. Unlike most existing works that stop as soon as an item is recommended [28, 34], we model the more realistic expectation that the interaction will continue if the item is not appropriate.

The core contributions of this work, presented in Section 3, are: 1) We propose a new unified model, AC-CRS, for conversation recommendation systems, one that jointly learns the dialogue policy and recommendation model at the same time; 2) AC-CRS appears to be the first attempt at estimating the value of questions in the conversational recommendation system; 3) AC-CRS introduces a tree-structured Actor for this task, allowing it to recommend items from a massive collection of possibilities; and, 4) In each round of the conversation, AC-CRS can have simultaneous recommendations if modality and screen real estate permit. Our contributions are rounded out by a set of experiments on real-world user purchasing data showing the effectiveness of AC-CRS in terms of standard evaluation measures such as NDCG. We start with some helpful background and notation as well as an overview of related work.

2 RELATED WORK

2.1 Conversational Search & Recommendation

Recently, with the emerging of intelligent conversational systems, and the process of neural approaches in the natural language process (NLP), conversational search and recommendation have achieved much more attention [5, 10, 33]. Zhang et al [34] proposed a Multi-Memory Network (MMN) architecture for conversational search and recommendation. However, their model can only predict questions in the training data which limits its flexibility and is not compatible with the nature of the conversational recommendation system. In other words, the system should be able to explore all questions to find the target item. Bi et al [3] introduced a conversational paradigm that utilizes non-relevant items in each round of the conversation. Sun et al [28] introduced a Belief Tracker model to extract facet-value pairs from user utterances during the conversation. They also proposed an approach based on reinforcement learning to learn ask questions related to the user needs. However, their model suffers from two limitations. First, their model cannot learn the dialogue policy and the recommendation model at the same time. Second, their model stops when the system decides to recommend an item and cannot support multiple recommendations. Recently, Lei et al [13] showed that the interaction between conversation and recommendation can improve the performance of these systems substantially. They also proposed a policy network to decide between asking a question or recommending an item. However, their model cannot predict a specific value for each question based on the target item and they simply considered a constant reward for each question. Moreover, the proposed model cannot learn to ask questions that are related to the category of the items and they just check whether an attribute is in the item.

2.2 Reinforcement Learning

Reinforcement learning (RL) is a framework of machine learning to optimize the behavior of an agent with respect to a desired reward [29]. Due to the progress of the deep learning, RL algorithms have demonstrated an impressive potential for tackling a wide range of complex tasks, from game playing [26] to robotic manipulation [21] and relevance feedback [16]. Google's DeepMind research on the game of Go is one of the most famous success of deep RL [25, 27].

The main roles in RL are the agent and the environment. The environment is the world that is visible to the agent and that the agent can interact with. At every step of interaction, the agent sees observations in the environment and takes some actions based on these observations. The agent receives a reward according to its actions. The reward is a measure to show how good and bad are the actions taken by the agent. The agent's final goal is to maximize the cumulative reward. In this study, we utilize the Actor-Critic framework [12, 30] shown in Figure 1. In this framework, the Actor based on the current state generates an action. The Critic inputs this state-action pair and produces an action-value which is a judgment of whether the chosen action matches the current state. Finally, based on the judgment from the Critic, the Actor updates its' parameters.

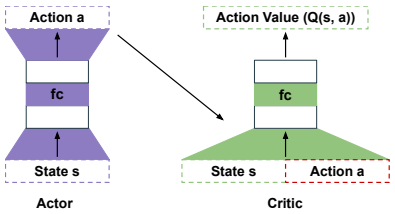

Figure 1: The Actor-Critic framework.

The Actor-Critic architecture is preferred for our task since it is suitable for large and dynamic action space. In this framework, the Critic estimates the action-value in each step which has an interesting advantage. Instead of waiting until the end of the episodes to calculate the reward, we can make an update in each step which is suitable for the conversational recommendation system. Estimating value for each action is another feature of this algorithm. In other words, we need to estimate a judgment for each question during the conversation. Therefore, we need a Critic to estimate this judgment for each question, individually. Zhao et al. [37] proposed an Actor-Critic algorithm to generate page-wise recommendations. Chen et al. [4] proposed an approach based on reinforcement learning for the interactive recommendation which is most related to ours. They trained a tree-structured policy gradient recommendation (TPGR) to deal with the large discrete action space problem in the recommendation. However, for a *conversational* recommendation system, since we do not know the actual value for each question in the conversation in the training, we have to estimate them by a Critic. Furthermore, we force the Actor to choose questions that make the current state closer to the target item. Asking high quality and relevant questions is an important task in conversational recommendation systems that we address here. So, inspired by their

work, we propose a tree-structured Actor in the Actor-Critic framework for the interactive *conversational* recommendation system in the large discrete action space.

3 METHODOLOGY

In this section, we discuss how we model a unified interactive conversational recommendation system based on reinforcement learning. We use an Actor-Critic algorithm to train the agent's policy to make a flexible conversation with users. We believe that a good solution for this task should have these features: 1) asking appropriate personalized questions in the right order to bring the system closer to the target item; 2) showing some items during the conversation to get some feedback from the user; 3) making the conversation shorter as much as possible; and 4) finding appropriate items from a massive collection of possibilities. To capture these features, we propose a model based on an Actor-Critic algorithm [12, 30]. In this model, in each round of the conversation, the Actor learns to decide between asking a question or recommending some items. The Critic evaluates the action just selected by the Actor and the Actor's parameters will be updated by this evaluation.

In the following sections, we first formulate the problem. Then, we elaborate the Actor and Critic architectures. Finally, we describe how the Actor and Critic are trained by stochastic gradient descent.

3.1 Problem Statement

Let $Q_0 = \{w_1, w_2, ..., w_i\}$ be initial request issued by a user u, $Q = \{Q_1, Q_2, ..., Q_N\}$ a set of possible questions, $A = \{A_1, A_2, ..., A_M\}$ a set of the possible answers for questions, and $V = \{v_1, v_2, ..., v_I\}$ a set of all items in the system. For each user u we have a list of clicked/purchased items P_u (i.e., $P_u \subset V$).

When a user initiates a conversation with the system, in each round of the conversation, the system can ask clarifying questions. Also, in each round of the conversation, the system should be able to show some items to the user and get some feedback. To construct a flexible conversation between a user and the system, the sequence of actions in the conversation can be shown as follows:

$$u_j \rightarrow Q_0 | a_0 f_0, a_1 f_1, ..., a_T f_T \rightarrow v_i, \tag{1}$$

where each action a_i can be from the collection of items or questions (i.e., $a_i \subset \{V, Q\}$), f_i is the feedback from the user which can be an answer for a question or a list of judgments for items shown to the user in the ith round of the conversation (i.e., $f_i \subset \{A, P_u\}$), v_i denotes the target item, and T is the maximum length of a conversation. In the following section, we show how we model the sequence of actions as a state for each round of the conversation.

3.2 State Representation

The state is designed to understand user preference in each round of the conversation. Figure 2 illustrates the model for generating the state. We introduce a RNN with Gated Recurrent Units (GRU) to capture the user's preference according to the initial request, questions asked by the Actor, and their answers. The reason we used GRU is that we want to train a model to learn the right order of the questions in the conversation [7]. We use GRU rather than Long Short-Term Memory (LSTM) because the performance of GRU is better than LSMT for capturing the user's sequential preference.

For each word in the conversation i.e., w_t, we have an embedding vector $\vec{x_t} = EL[w_t] = \vec{w_t} \in R^{1 \times D}$, where EL is the word embedding matrix to be learned in the training. The internal states of GRU is defined as follows:

$$\begin{aligned} \vec{z_t} &= \sigma(W_z \vec{x_t} + U_z \vec{h_{t-1}}) \\ \vec{r_t} &= \sigma(W_r \vec{x_t} + U_r \vec{h_{t-1}}) \\ \vec{h_t} &= (1 - \vec{z_t}).\vec{h_{t-1}} + \vec{z_t}.\vec{\hat{h}_t} \\ \vec{\hat{h}_t} &= \tanh[W \vec{x_t} + U(\vec{r_t}.\vec{h_{t-1}})]. \end{aligned} \tag{2}$$

We use the final hidden state $\vec{h_t}$ as the representation of the current user's preference.

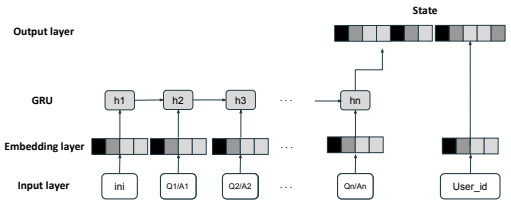

Figure 2: The model for generating the State.

A key point in conversation and recommendation is personalization [1, 34]. In other words, the Actor should ask personalized questions and recommend personalized items. To consider the inherent personalized preference of users, we utilize an embedding vector $\vec{u_i}$ for each user u_i and include this embedding to $\vec{h_t}$. Therefore, the personalized state is the concatenation of the hidden state $\vec{h_t}$ and the user embedding $\vec{u_i}$ as follows:

$$\vec{s_t} = (\vec{h_t}, \vec{u_i}). \tag{3}$$

Note that the user embeddings are randomly initialized and will be learned in the training.

3.3 Architecture of Actor

The Actor takes the state representation (i.e., Eq.3) as input and outputs the best action in the current state. The Actor has two kinds of actions: 1) ask a question or 2) recommend a list of items.

An obvious architecture for the Actor can be a feed-forward neural network and according to the output of the network, we can decide between asking a question or recommending a list of items. However, in this task, the number of items usually is much larger than the number of possible questions, making the Actor biased towards recommending more items even in the case that the user's request is not clear. Moreover, this large discrete action space makes the training inefficient and ineffective in RL-based models [37]. In other words, the Actor has to explore a large discrete action space to find the target items or appropriate questions to earn positive reward which makes the time complexity of making a decision linear to the size of the action space [6]. To tackle these challenges and achieve high effectiveness in the conversation, in this paper, the Actor is built upon a hierarchical clustering tree over items and questions. Figure 3 illustrates the architecture of the Actor. For each node, we utilize a policy network to learn the strategy of choosing the best subclass at each non-leaf node given the current state. In

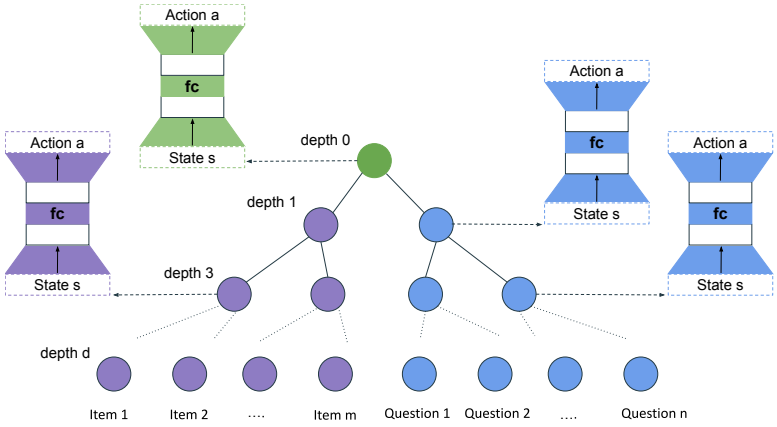

Figure 3: The Architecture of tree structured Actor.

more detail, each non-leaf node is mapped to a policy network and each leaf node is mapped to an item (the left side of the tree) or a question (the right side of the tree).

The policy network for each non-leaf node is designed with a feed-forward neural network which is composed of: the input layer $\vec{z_0}$, $l-1$ hidden layers, and the output layer $\vec{z_l}$. The input layer for all policy networks (i.e., non-leaf nodes) is the state representation in the current timestep t which is generated by Eq.3:

$$\vec{z_0} = \vec{s_t}. \tag{4}$$

Each hidden layer $\vec{z_i}$ is a fully-connected layer as follows:

$$\vec{z_i} = \psi(W_i.\vec{z_{i-1}} + b_i) \quad 1 \leq i \leq l-1 \tag{5}$$

where ψ is a non-linear activation function. The output layer $\vec{z_l}$ is a fully-connected layer same as Eq.5, but for the output layer, we used a softmax function as an activation function. The output of the softmax function is a probability distribution over possible outputs. Therefore, the probability of a choice c_i in each non-leaf node is computed as follows:

$$\pi(c_i|s_t) = \frac{e^{z_{l_i}}}{\sum_{j=1}^{C} e^{z_{l_j}}}, \tag{6}$$

where C is the number of child nodes for each non-leaf node. Therefore the size of the output layer $\vec{z_l}$ is C. Note that the number of child nodes for parents of leaf nodes can be at most C. Also, the number of child nodes for the root is $C = 2$, since in the root node, the Actor should decide between asking a question (the right side of the tree) or recommending an item (the left side of the tree). The probabilities of these two possibilities are computed by Eq.6.

In each subclass, given the current state and guided by the policy networks, a top-down moving is performed from the root to a leaf node and the corresponding item or question is performed as an action by the Actor.

3.3.1 ***Balanced Hierarchical Clustering over Items and Questions.*** As described above, the tree-structured Actor has two subclasses over items and questions. To make it efficient, we need to build up a *balanced* hierarchical clustering tree for each sub-class. One simple and popular approach to do that is *divisive approach*. In this approach, the original data points (i.e., the representation of items and questions individually) are divided into several clusters, and each cluster is divided into smaller sub-clusters.

To make the tree balance, for each node, the difference between heights of its sub-trees is at most one, and the number of child nodes for each non-leaf node is the same except for the parent of leaf nodes, whose number of child nodes can be different.

Same as Chen et al. [4], we use the PCA-based clustering algorithm which is widely used for this task. A balanced clustering tree can be constructed by applying the clustering algorithm until each item or question is assigned to a sub-cluster with only one member. The inputs of the clustering algorithm are representation vector of items or questions and an integer as a number of clusters. The representation of items and questions is computed by averaging the embedding vector of words in questions and descriptions of items.

3.3.2 ***Probability of an Action.*** Let $p_t = \{c_1, c_2, ..., c_d\}$ be a path from the root to a leaf node at timestep t decided by the Actor. The path p_t includes d (i.e., the number of layers in the tree) choices and each choice is an integer between 1 and maximum number of child of each node C (c_1 can be 1 or 2, since in the root, the Actor has two choices). Given the state, the probability of action a_t (i.e., choosing a question or an item) at timestep t is:

$$\pi_\theta(a_t|s_t) = \prod_{i=1}^{d} \pi_{\theta_i}(c_i|s_t), \tag{7}$$

where $\pi_{\theta_i}(c_i|s_t)$ is the probability of making each choice in the corresponding policy network from root to the chosen action which is computed in Eq.6.

3.3.3 ***Transition.*** Given previous state and action, transition defines the new state. As described above, the Actor has two kinds of actions. If the Actor decides to ask a question, this question and its answer will be added to the previous questions and initial request (see Figure 2) to generate a new final hidden state i.e., $\vec{h_{t+1}}$. On the other hand, if the Actor recommends an item to the user, simply the new state is the same as the previous state. One can use a more complicated approach for transition especially for item space but we leave it to future work. More formally, the state transition is

defined as:

$$s_{t+1} = \begin{cases} s_t + (Q_{a_t}, A_{a_t}), & \text{if } a_t \in \text{question space} \\ s_t, & \text{if } a_t \in \text{item space} \end{cases} \quad (8)$$

where Q_{a_t} is the a_tth question and A_{a_t} is the corresponding answer.

3.3.4 ***Reward.*** When the Actor takes action a_t at timestep t, it gets a reward which denotes how good is the current action. The current action can be recommending an item or asking a question. In both cases, the action includes a path from the root of the tree to the leaf node. We will elaborate the reward function for both kinds of actions.

Reward for Recommendation: Again, let $p_t = \{c_1, c_2, ..., c_d\}$ be the path from the root to the leaf node in the action a_t. In our case, for each layer of the tree (i.e., for each choice c_i), we set a specific reward. In more detail, the maximum reward is assigned to the last layer of the tree (i.e, c_d) which means that the Actor finds the target item, and the reward will decrease in each depth from the parent of leaf node to the root. As an example, if we have a tree with depth 3, the reward for finding the correct category of the target item in the first depth would be 10. The reward will be increased e.g., 100 if the Actor finds the correct category in the second depth. And finally, if the Actor finds the target item in the third depth, it would get the highest reward e.g., 1000.

Given an initial request for a target item, let $p'_t = \{c'_1, c'_2, ..., c'_d\}$ be the true path to the target item. Therefore, the recommendation reward is defined as follows:

$$R_r = \sum_{i=1}^{d} 10^i \mathbb{1}_{(c_i == c'_i)}, \quad (9)$$

where $\mathbb{1}_{(.)}$ represents the indicator function taking on a value of 1 if the selected choice is equal to the true choice, and 0 otherwise. This reward function reduces the exploration time and helps the Actor to find the target item by finding sub-classes instead of directly exploring the target item.

Reward for Asking Question: Intuitively, we want to ask questions that increase the rank of the target item against other items. However, in the large item space, this reward function cannot work very well and it takes a lot of time to improve the rank of the target item. Furthermore, some questions can help to find the sub-clusters of the target item and this is a key point in this task. For example, suppose the query is "charger for mobile". In this case, two possible sub-clusters can be "wireless" and "wired". Therefore, the Actor can ask a question related to these two sub-clusters to make it easier to find the target item.

Same as the reward for the recommendation, we propose a specific reward for each layer of the tree. In more detail, based on the output of the policy network of the corresponding non-leaf node we compute the reward as follows:

$$R_q = \sum_{i=1}^{d} 10^i \pi_{\theta_i}(c'_i | s_{t+1}), \quad (10)$$

where $\pi_{\theta_i}(c'_i | s_{t+1})$ is the probability of the correct choice in non-leaf nodes (see Eq.6), and s_{t+1} is computed by Eq.8. Specifically, suppose that the Actor selects a question as an action (i.e., Q_{a_t}). This question and its answer will be added to the current state by Eq.8 to compute s_{t+1}. Then, the reward function evaluates that if the added question and its answer to the state helps to find the true path to the target item or not. If the question helps to find more correct sub-clusters in the tree, the Actor will earn more rewards.

3.4 Architecture of Critic

To evaluate the action (i.e., recommending an item or asking a question) in the current state we design a Critic that takes the state and the taken action and learns an action-value function $Q(s, a)$. According to $Q(s, a)$, the Actor updates its parameters in order to generate more valuable actions in the conversation. The action-value function $Q(s, a)$ is usually non-linear, especially in our task, since the action and state spaces are enormous. Therefore, we build our Critic over a deep neural network to estimate $Q(s, a)$ (see Figure 1 and section 4.2.2 for more details). We will discuss how the parameter of the Critic will be updated in the following sections.

3.5 Time Complexity Analysis

The number of items in the item set is I (i.e., $I = |V|$, see section 3.1). Similarly, the number of questions in the question set is N (i.e., $N = |Q|$). Therefore, the time complexity for sampling an action in the normal RL-based methods would be $O(N + I)$ [6]. For sampling an action given a specific state, the AC-CRS needs to make d (i.e., the number of layers in the tree) choices. Each choice is based on a policy network with at most C output units. Therefore, the time complexity for sampling an action in the AC-CRS is $O(d \times C)$. Given the number of possible items and questions $N + I$, and d, we can set $C = \lceil (N + I)^{\frac{1}{d}} \rceil$. So, we can rewrite the time complexity in our model $O(d \times C) \simeq O(d \times (N + I)^{\frac{1}{d}})$, which shows the AC-CRS significantly reduces the time complexity.

3.6 Training and Test Procedure

3.6.1 ***Critic Training Procedure.*** The loss function for the Critic which is known by Temporal Difference (TD) learning [30] is derived from Bellman equation [2] as follows:

$$\mathcal{L}(\theta_\mu) = \mathbb{E}_{s_t, a_t, s_{t+1}, a_{t+1}, R}[(R + \gamma Q_{\theta_{\mu'}}(s_{t+1}, a_{t+1}) - Q_{\theta_\mu}(s_t, a_t))^2], \quad (11)$$

where θ_μ, γ denote all parameters in the Critic and discount factor, respectively. $\theta_{\mu'}$ shows the Critic's parameters from the previous iteration and will be fixed in the loss function $\mathcal{L}(\theta_\mu)$. Note that using the Bellman equation guarantee that we maximize long term reward (i.e., users' long term preferences). Therefore, derivations of the loss function with respect to parameters θ_μ is $\theta_\mu \leftarrow \theta_{\mu'} - \alpha \nabla_{\theta_\mu} \mathcal{L}(\theta_\mu)$ where α is the learning rate for the Critic training.

3.6.2 ***Actor Training Procedure.*** Asking questions related to the item description is a key point in this task. We used the average vector of words embedding in each item description to build our tree-structured Actor which is important for the Actor to find appropriate questions for each category. However, the embedding of words is learning during the training and this makes it hard to ask specific questions for each item. To solve this problem, we have two objectives for Actor training: the recommendation objective with the goal of improving recommendation accuracy and the questioner objective with the goal of asking questions related to the item description.

Algorithm 1 Learning AC-CRS

Input: tree depth d, number of child nodes c, discount factor γ, Critic learning rate α, Actor learning rate η, Reward function for recommendation R_r and asking question R_q, item set V, question set Q, and answer set A, N training data

Output: Actor's parameters θ_π.

Initialize Critic's parameters θ_μ, and reply memory D
Construct a balanced clustering tree T with depth d
$H = \frac{c^d - 1}{c-1}$ ▷ Number of non-leaf nodes in the tree
$\Delta\theta_\mu = 0$, and $\Delta\theta_\pi = 0$
for $i \leftarrow 1, H$ **do**
    initialize θ_i with random values
end for
$\theta_\pi = (\theta_1, \theta_2, \ldots, \theta_H)$
for $n \leftarrow 1, N$ **do**
    $[s_1, r_1, p_1, s_2, \ldots, s_m, r_m, p_m, s_{m+1}] \leftarrow Sampling(d, R_q, R_r, C, \theta_\mu, \theta_\pi, n, D, V, Q, A)$ (see Algorithm 2)
    for $j \leftarrow 1, m$ **do**
        map p_j to an item or a question a_j w.r.t T
        Update Critic and Actor by minimizing Eq.11, and Eq.15, respectively
        $\theta_\mu \leftarrow \theta_{\mu'} - \alpha \nabla_{\theta_\mu} \mathcal{L}(\theta_\mu)$, and $\theta_\pi \leftarrow \theta_{\pi'} - \Delta\nabla_{\theta_\pi} \mathcal{L}(\theta_\pi)$
    end for
end for
return θ_π ▷ Actor's parameters

Algorithm 2 Sampling Episode for AC-CRS

Input: Reward function for recommendation R_r and asking question R_q, training data n, item set V, question set Q, and answer set A, Actor's parameters θ_π, replay memory D, Maximum length of Conversation T.

Output: an Episode E.

$(user_n, target - item_n)$
$c \leftarrow 0$ ▷ Number of rounds in the conversation
$s_0 \leftarrow$ (Initial request) and $s_t \leftarrow s_0$
while $True$ **do**
    sample $a_t, path \sim \pi_\theta(s_t)$ ▷ Using Eq.7
    $p_t \leftarrow path$
    if $a_t \in recommendation$ **then** ▷ Actor chooses to recommend
        $r_t \leftarrow R_r(s_t, a_t, path)$ ▷ Using Eq.9
        $s_{t+1} \leftarrow s_t$
    else ▷ Actor chooses to ask clarifying question
        $r_t \leftarrow R_q(s_t, a_t, path)$ ▷ Using Eq.10
        $s_{t+1} \leftarrow s_t + Q_{a_t} + A_{a_t}$ ▷ calculate s_{t+1} as described in Fig.2
    end if
    store experience $< s_t, r_t, p_t, s_{t+1} >$ in replay memory D
    if a_t is equal to the target item **then**
        the conversation succeeds and break
    end if
    if c is equal to the Maximum length of Conversation **then**
        the conversation fails and break
    end if
    $c \leftarrow c + 1$
end while
for $i \leftarrow 1, m$ **do**
    $e_i \leftarrow$ sample random transition $< s_i, r_i, p_i, s_{i+1} >$ from replay memory D
    Add e_i to E
end for
return E ▷ return samples

Recommendation Objective: The first loss function forces the Actor to predict the true path from the root to the leaf nodes which is designed as follows:

$$\mathcal{L}_1(\theta_\pi) = \log \pi_\theta(a|s), \tag{12}$$

where $\pi_\theta(a|s)$ is computed according to Eq.7.

Questioner Objective: In the second loss function, we push Actor's parameters in the last layer of the tree (i.e., parent of the leaf nodes) to be similar to the item descriptions. The reason is that the Actor should ask questions related to the item description. Therefore, we make sure that the representation of each item in the tree (i.e., the output layer of the Actor in the left side of the tree in Figure 3) is similar to the item description, and this can help the Actor to ask questions that push the state representation (see section 3.2) to be similar to the item description.

Assume that v_i is the item that is selected by the Actor (not necessarily the target item). We have a textual description for this item which is generated by merging item description and the textual reviews for this item [34]. Therefore, the output layer of the Actor (in the corresponding sub-cluster) should be similar to the description of this item (if the action-value is positive and vice versa) to help the Actor to find questions related to this item. Therefore, we use a GRU to convert the textual description to a fixed size embedding vector. The parameters of this GRU are shared with one that we use in the state representation. This helps the Actor by enriching its knowledge about products and reducing the vocabulary mismatch between products and user queries. We minimize the difference between the item description and the output layer of the last layer in the tree-structured Actor as follows:

$$\mathcal{L}_2(\theta_\pi) = \|\vec{z_{l_{\theta_i}}} - \vec{h_{v_i}}\|_F, \tag{13}$$

where $\vec{z_{l_{\theta_i}}}$ is the output layer of the parent of leaf node which is mapped to the item v_i in the Actor and $\vec{h_{v_i}}$ denotes the item descriptions generated by GRU.

Loss Function: Finally, the loss function for updating the Actor's parameters is designed as follows:

$$\mathcal{L}(\theta_\pi) = \mathcal{L}_1(\theta_\pi) + \mathcal{L}_2(\theta_\pi). \tag{14}$$

We utilize REINFORCE algorithm [32] to update parameters in the Actor as follows:

$$\nabla_{\theta_\pi} J(\theta_\pi) \approx \mathbb{E}_{\pi_\theta}[\nabla_{\theta_\pi} \mathcal{L}(\theta_\pi) Q_{\theta_\mu}(s, a)] \tag{15}$$

where $Q_{\theta_\mu}(s, a)$ denotes the action value which is the output of the Critic. Algorithm 1 shows the training procedure.

3.6.3 ***The Test Procedure.*** At test time, we simply use the Actor θ_π as our conversational recommendation system. The input of the Actor in the test time will be an initial request. In each round of the conversation, the Actor can ask a question or recommend a list of items. If the Actor decides to recommend a list of the item, we select the top 100 items and add them to our ranking for this session. In the next rounds, if the Actor recommends some other items, we add them to this ranking list. If the Actor chooses to ask a question, we return the answer of this question if the answer is in the item description. The conversation will be ended if the Actor finds the target item or the number of rounds in the conversation exceed the maximum of rounds.

4 EXPERIMENTS

4.1 Datasets

Following previous works on this task [1, 34], we use the Amazon product dataset. This dataset comprises millions of products and customers, and rich metadata such as product descriptions, multi-level product categories, and reviews for products[2]. There are 24 sub-datasets of different product types. In this study, we use *Electronics* and *CDs&Vinyl*, which are large-scale datasets [34] to test our model performance. Table 1 shows the basic statistics of these two datasets. We used the question-answer (or aspect-value) pair extraction toolkit [35, 36] to extract QA pairs for each dataset same as Zhang et al. [34].

[2] https://nijianmo.github.io/amazon/index.html

Table 1: Basic statistics of the experimental datasets, where l(Request) is the average length of initial requests.

Dataset	#Users	#Items	#Questions	#Answers	#QA pairs	#Request	#l(Request)	#User-Query pairs
CDs & Vinyl	64,847	60,405	514	747	659,737	694	5.71	99,759
Electronics	142,421	53,278	479	500	475,020	989	6.40	381,688

Table 2: Comparison of proposed model AC-CRS and baselines. The superscript ▲ indicates that the improvements over all baselines are statistically significant.

Dataset	Metric	QL	BM25	LSE	MMN	PMMN	CRM	EAR	AC-CRS
CDs & Vinyl	*MRR*	0.0013	0.0027	0.0149	0.0441	0.0631	0.0574	0.1106	**0.1379**▲
	NDCG	0.0009	0.0016	0.0070	0.0081	0.0107	0.0080	0.0208	**0.0332**▲
Electronics	*MRR*	0.0405	0.0400	0.0450	0.0581	0.0864	0.0813	0.1212	**0.1409** ▲
	NDCG	0.0054	0.0043	0.0056	0.0066	0.0201	0.0160	0.0225	**0.0363**▲

4.1.1 ***Initial Request Construction.*** Following Zhang et al. [34], we used a three-step paradigm of product search to construct the initial request Q_0 for each user u_i which purchased a item v_j: 1) extract the multi-level category of information of item v_j from the metadata, 2) concatenate the terms in this information, and 3) remove stopwords and duplicate words. The number of initial requests and their average length are in Table 1. We randomly select 70% of the data for each user in the training/validation set and keep the other 30% to make the test set.

4.1.2 ***Baselines.*** We compare our model with the representative product search or recommendation methods as baseline: 1) **QL:** The query likelihood model proposed by Ponte and Croft [20], a language model approach to rank items according to the log-likelihood of the query, 2) **BM25:** An effective and widely-used retrieval method [22] to rank items based on term frequency, inverse document frequency of query terms and item description length, 3) **LSE:** The latent semantic entity which is a non-personalized product search model [31], 4) **MMN:** A non-personalized model for the conversational recommendation. MMN utilizes Multi-Memory Network (MMN) architecture to track the conversation between user and system [34], 5) **PMMN:** A personalized version of MMM [34], 6) **CRM:** A conversational recommendation system which utilize reinforcement learning to track the conversation between user and the system [28], 7) **EAR:** A three-stage approach named Estimation–Action–Reflection emphasizes the interaction between the conversation and recommendation components [13].

4.2 Experimental and parameter setting

4.2.1 ***Evaluation Measures and Evaluation Methodology.*** For evaluating the performance of the models we use mean reciprocal rank (MRR) and normalized discounted cumulative gain (NDCG) at 10 [9]. Note that we assume that for each conversation we have a single target item, so MRR is equal to mean average precision (MAP).

For evaluating multiple recommendations in an interactive conversational recommendation system, we utilize the freezing ranking paradigm [24]. In each round of the conversation, if our model is recommending some items to the user, we select the top 100 items and add them to our ranking. Note that in this case, the successive recommendation cannot affect the top 10, and NDCG@10 will have a fixed value. In the next rounds of the conversation, if the model recommends some other items, we add them to this ranking list. At the end of the conversation, we evaluate the ranking list provided by the system. Statistically significant differences of performance are determined using the two-tailed paired t-test at a 95% confidence level ($p\_value < 0.05$). For our model and baselines, we tuned all hyper-parameters on the validation set.

4.2.2 ***Parameter Setting.*** We implemented and trained our model using Tensorflow [3]. The networks parameter in Actor and Critic models are trained with Adam optimizer [11] according to the back-propagation algorithm [23]. The learning rate for the Actor and Critic were selected from $[1e-4, 5e-5, 1e-5, 5e-6]$, and $[1e-3, 5e-4, 1e-4, 5e-5]$, respectively. The learning rate for Critic is larger than Actor since we expect that Critic should learn faster than the Actor. For each sub-networks we used 4 hidden layers. The layer size was selected from $\{100, 150, 200\}$. The discount factor in Eq.11 was set to 0.99 since it has been shown this value works well in the reinforcement learning [12]. The size of the replay memory was set to 200. The depth of the tree was selected from $\{1, 2, 3, 4\}$. We observed that the best value for the depth of the tree is 4. We initialize the embedding matrix by pre-trained GloVe [19] vectors trained on Wikipedia dump 2014 plus Gigaword 5[4] with dimension of 100. The maximum length of conversation T was set to 7.

4.3 Results and Discussion

4.3.1 ***Comparison with the Baselines.*** In the first experiments, we evaluate our model against popular state-of-the-art models for conversational recommendation systems. The results for our model and baselines are reported in Table 2. The first observation from this Table 2 is that the results of QL, BM25, and LSE are comparable. QL and BM25 cannot capture semantic relation between words while LSE is able to detect semantic matching between words. According to the results in Table 2, semantic matching improves the performance a little [8, 15, 17, 18]. So, in our model, we considered semantic matching by learning embedding vectors.

According to Table 2, personalized approaches (i.e., PMMN and CRM) works much better than non-personalized models (i.e., MMN

[3] https://www.tensorflow.org/
[4] https://nlp.stanford.edu/projects/glove/

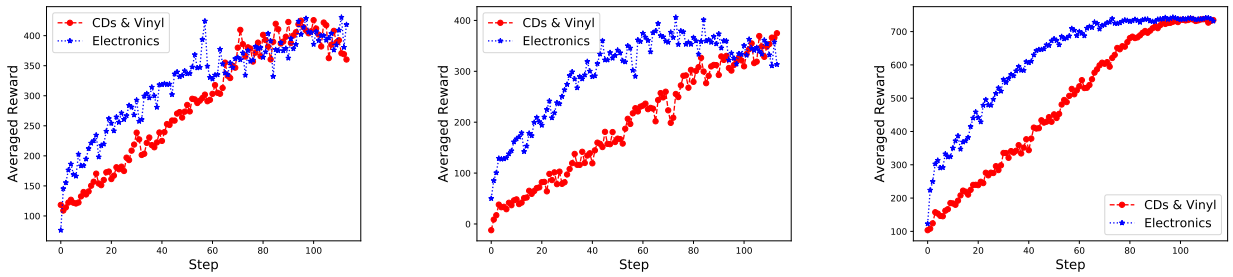

Figure 4: Average reward for asking questions (left), recommending items (middle), and both actions (right).

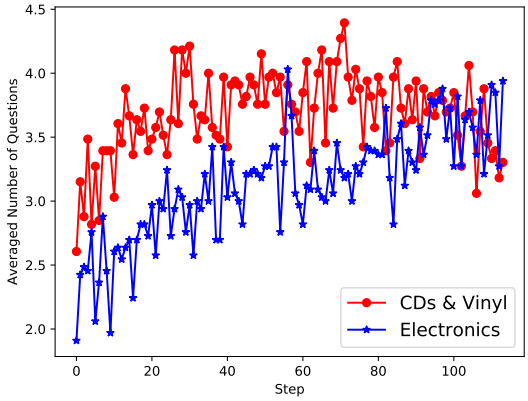

Figure 5: Average Number of questions.

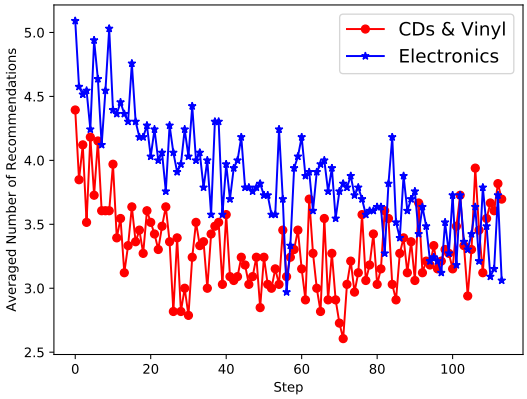

Figure 6: Average number of recommendations.

and LSE). In other words, personalized conversational recommendation enables models to perform better. Our model also takes advantage of the personalized recommendation (see Eq.3). Also, this makes our model be able to ask personalized questions based on the knowledge of the user obtained in the interaction.

Another observation from the results reported in Table 2 is that AC-CRS outperforms personalized and non-personalized models. This shows joint modeling and optimizing of dialogue policy and recommendation offers substantial improvements. Also, EAR underperforms our model in two datasets. A key reason is that our model can find sub-clusters of a target item at first and this will help to learn more about the user's preference and finally find the target item. Another reason is related to the Critic in our model. In more detail, our model can detect bad questions and recommended items in a conversation even the conversation succeeds. However, the EAR model assigns positive values for each question in a successful conversation.

The improvements achieved by the AC-CRS in terms of MRR are much better than NDCG. This shows that in most cases the model is not able to detect target items in the first round of the recommendation and needs to show some items to the user and get some feedback to find the target items in the following rounds. In other words, this observation implies the importance of the multi recommendation in this task.

4.3.2 ***Analysis of the Rewards.*** As pointed out earlier, AC-CRS is flexible to ask a question or recommend a list of item in each round of the conversation and gets rewards. In this experiment, we evaluate the performance of these two actions in terms of reward. To evaluate our model under both actions, the averaged reward for asking a question and recommending an item are shown in Figure 4. According to these figures, in each round of the conversation, if the Actor decides to ask a question, this question increases the rank of the target item (see Eq.10). Note that, our model has the ability to ask questions related to the sub-classes and increase the chance of finding the target item. For example, suppose the query is "charger for mobile". In this case, two possible clusters can be "wireless" and "wired" and the Actor can ask a question related to this situation to make it easier to find the target item. A similar observation can be seen in Figure 4 for recommending an item. According to this figure, the system can get more rewards since the Actor explores to find the sub-classes of the target item and increase his chance to recommend the target item.

Figure 4 (on the right) indicates the average reward that the system receives in general. According to this figure, the system tries to continue the process of finding the target product until it becomes stable.

4.3.3 ***Analysis of the Number of Asked questions and Recommendations.*** Figure 5 shows the average number of questions asked by the Actor. Similarly, Figure 6 indicates the average number of recommendations in each conversation. According to these figures, at first, the number of recommendations in a conversation is more than the number of questions in both collections. However, according to Figure 4 (on the right), the averaged reward at the first is not high. This implies that, at first, the Actor recommends items without knowing exactly what the user needs. For this reason, the Actor starts to ask more questions and recommend fewer products (see Figures 5 and 6). Although the Actor recommends fewer products, however, it can recommend with more quality and get more rewards for a recommendation (see Figure 4). This behavior is due to the interaction of the dialogue policy and recommendation model in the training.

Another observation from the results reported in Figure 5 and 6 is that in CDs & Vinyl dataset, after some steps (around 70), the Actor starts to ask fewer questions and recommend more items. This implies that the Actor after asking some questions is more confident about the users' needs and begin to recommend items with fewer questions in a conversation.

4.3.4 ***Effect of tree depth.*** We evaluate our approach with different tree depth to show the tree depth influence the performance. The tree-structured tree helps the Actor to find the target item with less time. To illustrate this, we show the number of finding the target product (with an equal number of steps) with different depth of tree in Figure 7. According to this figure, with increasing the depth of the tree, the Actor can find more target items since it can first find sub-classes and this brings the Actor closer to the target item and find it.

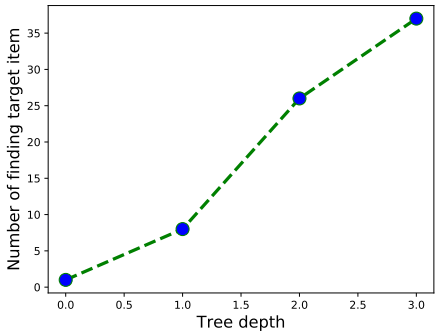

Figure 7: Effect of tree depth

5 CONCLUSIONS AND FUTURE WORK

We argue and propose a unified framework based on the Actor-Critic algorithm to jointly learn the dialogue policy and the recommendation model at the same time. Our model can have flexible interaction with the user to reach the goal of the conversation. The Actor in our framework is built upon a hierarchical clustering tree to allow the Actor to recommend items from a massive collection of possibilities. We showed the effectiveness of our model on real-world user purchasing data in terms of standard evaluation measures such as NDCG. As in our experiments showed that ACCRS at the first of the training tries to recommend items without asking questions. However, after some steps in the training, ACCRS learns to ask questions about user needs that are ambiguous. This behavior is due to the interaction of the dialogue policy and recommendation model.

Constructing a richer and more complicated architecture for the Actor can be an interesting future work. We are also interested in generating more interesting questions in this task, allowing the Actor to choose between question derived from product descriptions and reviews rather than simply from stored database values.

ACKNOWLEDGEMENTS

This work was supported in part by the Center for Intelligent Information Retrieval and in part by Amazon.com. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect those of the sponsors.

REFERENCES

[1] Qingyao Ai, Yongfeng Zhang, Keping Bi, Xu Chen, and W Bruce Croft. 2017. Learning a hierarchical embedding model for personalized product search. In *SIGIR'17*. 645–654.
[2] Richard Bellman. 1966. Dynamic programming. *Science* 153, 3731 (1966), 34–37.
[3] Keping Bi, Qingyao Ai, Yongfeng Zhang, and W Bruce Croft. 2019. Conversational product search based on negative feedback. In *CIKM'19*. 359–368.
[4] Haokun Chen, Xinyi Dai, Han Cai, Weinan Zhang, Xuejian Wang, Ruiming Tang, Yuzhou Zhang, and Yong Yu. 2019. Large-scale interactive recommendation with tree-structured policy gradient. In *AAAI'19*, Vol. 33. 3312–3320.
[5] Konstantina Christakopoulou, Filip Radlinski, and Katja Hofmann. 2016. Towards conversational recommender systems. In *SIGKDD'16*. 815–824.
[6] Gabriel Dulac-Arnold, Richard Evans, Hado van Hasselt, Peter Sunehag, Timothy Lillicrap, Jonathan Hunt, Timothy Mann, Theophane Weber, Thomas Degris, and Ben Coppin. 2015. Deep reinforcement learning in large discrete action spaces. *arXiv preprint arXiv:1512.07679* (2015).
[7] Ayyoob Imani, Amir Vakili, Ali Montazer, and Azadeh Shakery. 2019. An Axiomatic Study of Query Terms Order in Ad-hoc Retrieval. In *European Conference on Information Retrieval*. Springer, 196–202.
[8] Ayyoob Imani, Amir Vakili, Ali Montazer, and Azadeh Shakery. 2019. Deep neural networks for query expansion using word embeddings. In *European Conference on Information Retrieval*. Springer, 203–210.
[9] Kalervo Järvelin and Jaana Kekäläinen. 2002. Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems (TOIS)* 20, 4 (2002), 422–446.
[10] Tom Kenter and Maarten de Rijke. 2017. Attentive memory networks: Efficient machine reading for conversational search. *arXiv preprint arXiv:1712.07229* (2017).
[11] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *ICLR'15* (San Diego, CA, USA).
[12] Vijay R Konda and John N Tsitsiklis. 2000. Actor-critic algorithms. In *Advances in neural information processing systems*. 1008–1014.
[13] Wenqiang Lei, Xiangnan He, Yisong Miao, Qingyun Wu, Richang Hong, Min-Yen Kan, and Tat-Seng Chua. 2020. Estimation-action-reflection: Towards deep interaction between conversational and recommender systems. In *Proceedings of the 13th International Conference on Web Search and Data Mining*. 304–312.
[14] Raymond Li, Samira Ebrahimi Kahou, Hannes Schulz, Vincent Michalski, Laurent Charlin, and Chris Pal. 2018. Towards deep conversational recommendations. In *Advances in neural information processing systems*. 9725–9735.
[15] Ali Montazeralghaem, Razieh Rahimi, and James Allan. 2020. Relevance Ranking Based on Query-Aware Context Analysis. *Advances in Information Retrieval* 12035 (2020), 446.
[16] Ali Montazeralghaem, Hamed Zamani, and James Allan. 2020. A reinforcement learning framework for relevance feedback. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 59–68.
[17] Ali Montazeralghaem, Hamed Zamani, and Azadeh Shakery. 2016. Axiomatic analysis for improving the log-logistic feedback model. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*. 765–768.

[18] Ali Montazeralghaem, Hamed Zamani, and Azadeh Shakery. 2017. Term proximity constraints for pseudo-relevance feedback. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 1085–1088.

[19] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation. In *EMNLP'14*. 1532–1543.

[20] Jay M Ponte and W Bruce Croft. 1998. A language modeling approach to information retrieval. In *SIGIR'98*. 275–281.

[21] Ivaylo Popov, Nicolas Heess, Timothy Lillicrap, Roland Hafner, Gabriel Barth-Maron, Matej Vecerik, Thomas Lampe, Yuval Tassa, Tom Erez, and Martin Riedmiller. 2017. Data-efficient deep reinforcement learning for dexterous manipulation. *arXiv preprint arXiv:1704.03073* (2017).

[22] Stephen E Robertson and Steve Walker. 1994. Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval. In *SIGIR'94*. Springer, 232–241.

[23] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. 1986. Learning representations by back-propagating errors. *nature* 323, 6088 (1986), 533.

[24] Ian Ruthven and Mounia Lalmas. 2003. A survey on the use of relevance feedback for information access systems. *The Knowledge Engineering Review* 18, 2 (2003), 95–145.

[25] 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.

[26] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. 2017. Mastering the game of Go without human knowledge. *Nature* 550, 7676 (2017), 354.

[27] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. 2017. Mastering the game of go without human knowledge. *Nature* 550, 7676 (2017), 354–359.

[28] Yueming Sun and Yi Zhang. 2018. Conversational recommender system. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*. 235–244.

[29] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.

[30] Richard S Sutton, Andrew G Barto, et al. 1998. *Introduction to reinforcement learning*. Vol. 135. MIT press Cambridge.

[31] Christophe Van Gysel, Maarten de Rijke, and Evangelos Kanoulas. 2016. Learning latent vector spaces for product search. In *CIKM'16*. 165–174.

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

[33] Liu Yang, Hamed Zamani, Yongfeng Zhang, Jiafeng Guo, and W Bruce Croft. 2017. Neural matching models for question retrieval and next question prediction in conversation. *arXiv preprint arXiv:1707.05409* (2017).

[34] Yongfeng Zhang, Xu Chen, Qingyao Ai, Liu Yang, and W Bruce Croft. 2018. Towards conversational search and recommendation: System ask, user respond. In *CIKM'18*. 177–186.

[35] 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.

[36] Yongfeng Zhang, Haochen Zhang, Min Zhang, Yiqun Liu, and Shaoping Ma. 2014. Do users rate or review? Boost phrase-level sentiment labeling with review-level sentiment classification. In *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval*. 1027–1030.

[37] 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.