Extracting Relevant Information from User's Utterances in Conversational Search and Recommendation

Ali Montazeralghaem
Center for Intelligent Information Retrieval
College of Information and Computer Sciences
University of Massachusetts Amherst
Amherst, MA, USA
montazer@cs.umass.edu

James Allan
Center for Intelligent Information Retrieval
College of Information and Computer Sciences
University of Massachusetts Amherst
Amherst, MA, USA
allan@cs.umass.edu

ABSTRACT

Conversational search and recommendation systems can ask clarifying questions through the conversation and collect valuable information from users. However, an important question remains: how can we extract *relevant information* from the user's utterances and use it in the retrieval or recommendation in the next turn of the conversation? Utilizing relevant information from users' utterances leads the system to better results at the end of the conversation. In this paper, we propose a model based on reinforcement learning, namely RelInCo, which takes the user's utterances and the context of the conversation and classifies each word in the user's utterances as belonging to the relevant or non-relevant class. RelInCo uses two Actors: 1) Arrangement-Actor, which finds the most relevant order of words in user's utterances, and 2) Selector-Actor, which determines which words, in the order provided by the arrangement Actor, can bring the system closer to the target of the conversation. In this way, we can find relevant information in the user's utterance and use it in the conversation. The objective function in our model is designed in such a way that it can maximize any desired retrieval and recommendation metrics (i.e., the ultimate goal of the conversation). We conduct extensive experiments on two public datasets and our results show that the proposed model outperforms state-of-the-art models.

CCS CONCEPTS

• **Information systems → Retrieval models and ranking**; **Retrieval tasks and goals**; **Information extraction**; **Recommender systems**.

KEYWORDS

Conversational Search, Conversational Recommender System, Deep Reinforcement Learning, Relevant Information, User's Utterance, Intelligent Assistants

ACM Reference Format:
Ali Montazeralghaem and James Allan. 2022. Extracting Relevant Information from User's Utterances in Conversational Search and Recommendation. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '22), August 14–18, 2022, Washington, DC, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3534678.3539471

1 INTRODUCTION

By enabling retrieval and recommender systems to dynamically obtain user preferences through conversations with users, conversational search, and recommendation have become increasingly popular in recent years [7, 16, 17, 28, 36, 42].

This process starts with receiving a request from the user and continues with asking clarifying questions or suggesting some possible items or documents by the system. In this way, the system can get valuable feedback from users to accurately determine the users' needs. This process repeats until the search or recommendation is successful, or the user accepts defeat.

Recently, the community has started exploring various settings of this task [1, 5, 16, 36, 43]. For example, Li et al. [16] have collected a publicly available large-scale dataset for conversational recommendations, and explored novel neural architectures, procedures, and approaches for designing conversational recommendation systems. Moon et al. [25] introduced a collection of conversations between two crowdsourcing agents about a specific topic or entity. Zhang et al. [43] proposed a system ask - user respond (SAUR) paradigm for conversational search and recommendation. However, their model is built on aspects of products as questions and their values as answers. Furthermore, their model cannot optimize the ranking measures as the ultimate goal of conversational search and recommendation. Zhou et al. [46] proposed an approach to incorporate both word-oriented and entity-oriented knowledge graphs (KG) in conversational recommender systems to compensate for the semantic gap between natural language expression and item-level user preference. They used knowledge graphs to add sufficient contextual information for accurately understanding users' preferences.

However, most existing works fail to extract *relevant information* from the user's utterances for the retrieval or recommendation in the next turn of the conversation. The reason is that they focus on finding terms in the user's utterances that are semantically similar to the context of the conversation. However, the main objective in the information retrieval and recommender systems is *finding relevant information*. This means that a word might be semantically close to the context of the conversation, but does not improve the rank of the target item or document [23]. This objective is more important in the conversational version of these tasks, since in each turn of the conversation if we cannot extract relevant information from the user's utterance, it will prolong the conversation making it more likely to fail at the end. Even worse, most existing works indiscriminately add all past information to the context of the conversation for the next turns, meaning that non-relevant material is also preserved in the conversation.

In this paper, we propose RelInCo, a reinforcement-based algorithm to extract relevant information from the user's utterance in conversational search and recommendation. The main objective in our model is improving retrieval performance (i.e., *relevancy*) e.g., any desired retrieval and recommendation metrics. Maximizing ranking metrics is a challenging problem since ranking metrics are *non-differentiable*. Reinforcement learning is an effective approach to maximize non-differentiable metrics in various problems [2, 26, 31]. One of the most effective algorithms in reinforcement learning is the Actor-Critic algorithm [14, 38]. In this approach, the Actor takes an action and its Critic informs the actor how good the action was and how it should be improved. In our task, we design a selector-Actor that determines which words in a user's utterance can bring the system closer to the target of the conversation. Therefore, given a user utterance and the context of the conversation, in each step, the selector-Actor takes a word of the user utterance and classifies it as belonging to the relevant or non-relevant class. In other words, the selector-Actor's action is selecting or discarding the word. We update the context of the conversation by the word if the selector-Actor decides to keep the word.

Given that the selector-Actor judges words in the user's utterance sequentially, we need to know which arrangement of the words is most effective in the improving ranking of the target item. Therefore, we design an arrangement-Actor that takes the user's utterance and returns an effective order of words to be used by the selector-Actor. The workflow of RelInCo is shown in Figure 1.

Both Actors in our model are trained using Actor-Critic algorithm via Proximal Policy Optimization (PPO) [33] which improves the supervision of Actors.

We model the reward function as a utility calculator such that it can optimize for different evaluation metrics, such as average precision or normalized discounted cumulative gain (NDCG) [12]. In the ideal case, the utility calculator can be designed by user satisfaction signals to capture relevance.

The core contributions of this work, presented in Section 3, are:

- We present RelInCo, a reinforcement-based model to extract relevant information from users' utterances in conversational search and recommendation. RelInCo appears to be the first attempt at extracting relevant information in conversational search and recommendation.
- RelInCo introduces two Actors, a selector-Actor and an arrangement Actor, which can be trained either simultaneously or in sequence to find the most effective words in a user's conversational search and recommendation process.
- We design an efficient utility calculator as our reward function to capture relevance and guide both Actors and Critics.

Our contributions are rounded out by a set of experiments on two public conversational search and recommendation datasets showing the effectiveness of RelInCo 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 and Recommendation

Belkin et al. [3] was one of the first papers to propose an interactive information retrieval system based on scripted conversational interaction for search. Croft and Thompson [8] introduced an intelligent intermediary for information retrieval to enable the system to interact with users during a search session. With the introduction of intelligent conversational systems and the use of neural methods in the natural language process (NLP), this field has exploded in popularity. A Multi-Memory Network (MMN) architecture for conversational search and recommendation was proposed by Zhang et al. [43]. Yang et al [41] proposed an approach to predict the next question in conversations. A Belief Tracker model was developed by Sun et al. [36] to derive facet-value pairs from user utterances during the conversation. They proposed a policy network to decide between asking a question or recommending an item. Recently, Lei et al. [15] showed that combining dialogue and recommendation can significantly increase the performance of these systems. They also developed a policy network for deciding whether to ask a query or provide a recommendation. Zou et al. [48] suggested a question-based recommendation system for eliciting user preferences over descriptive item attributes. Li et al. [16] released a standard conversational recommendation system dataset named REDIAL. They also proposed a hierarchical RNN model to generate utterances and recommendations. Chen et al. [7] and Liao et al. [17] use external Knowledge Graph (KG) to improve conversational recommendation systems performance. Zhou et al. [47] contributed a new conversational recommendation system dataset named TG-ReDial and proposed the task of topic-guided conversational recommendation. They also proposed an effective approach for this task. Moon et al. [25] proposed another dataset for the conversational search and recommendation. Each conversation turn is accompanied by a set of "KG routes" that connect the KG entities and relationships indicated in the dialog. Overall, these systems emphasize accurate retrievals and recommendations, with simple or heuristic solutions implementing the dialogue component. Montazeralghaem et al. [20] introduced a reinforcement-based model for large-scale conversational recommender systems. Most of the existing models are based on an assumption that there is a set of effectively pre-defined candidate questions for each product to be asked. Montazeralghaem and Allan [19] proposed an approach based on reinforcement learning to generate questions from product descriptions in each round of the conversation.

2.2 Deep Reinforcement Learning

Reinforcement learning (RL) is a machine learning framework for optimizing an agent's behavior in relation to the desired reward [37]. RL algorithms have proven an astonishing potential for solving a wide range of complicated tasks, from the game playing [**?**] to robotic manipulation [27] and relevance feedback [22], because of advances in deep learning. One of the most well-known successes of deep RL is Google's DeepMind study on the game of Go [34, 35].

The agent and the environment play the most important roles in RL. The environment is the world that is visible to the agent and the agent can interact with. The agent sees observations in the environment at each step of the interaction and conducts actions based on these observations. According to its acts, the agent is rewarded. The reward is a metric that indicates how excellent or poor the agent's actions are. The agent's ultimate objective is to maximize the total reward.

In this study, we utilize the Actor-Critic framework [14, 38]. In this paradigm, the Actor generates an action based on the current state. The Critic takes this state-action pair and generates an action-value, which is a determination of whether the chosen action is appropriate for the current state. Finally, based on the judgment from the Critic, the Actor updates its' parameters. The Actor-Critic architecture is preferred for our task since it is suitable for large state and action space. The Critic estimates the action-value in each phase in this architecture, which has an intriguing advantage. In our task, we need to estimate a judgment for each action during an episode. In other words, we want to maximize the evaluation measures at the end of an episode. However, we need to estimate a specific value for each action in the episode. So, we need to design a Critic to predict a value for each action. Zhao et al. [44] introduced an Actor-Critic algorithm to generate page-wise recommendations.

Most of the reinforcement learning approaches suffer from high variances. Proximal Policy Optimisation (PPO) is a recent development in Reinforcement Learning [33]. To solve the high variances in reinforcement learning, PPO guarantees that the updated policy isn't too dissimilar from the old policy in order to maintain low variances in training. The Actor-Critic Model, which uses two Deep Neural Networks, one taking action (Actor) and the other handling rewards (Critic) , is the most frequent implementation of PPO. This approach was suggested and implemented by OpenAI, and it performed admirably [33].

3 METHODOLOGY

In this section, we outline the problem setup to extract relevant information from users' utterances for conversational search and recommendation. Then, we lay out our RelInCo approach to solving this problem.

We use the Actor-Critic algorithm to train our model to extract relevant information from users' utterances [14]. We propose two separate Actors in our model: 1) Arrangement-Actor, and 2) Selector-Actor. In each round of the conversation, given the context of the conversation and the user utterance, the selector-Actor learns to classify each word in the user utterance as belonging to the relevant or non-relevant class. In other words, the selector-Actor predicts whether the word is useful for finding the target document or item in a list. The order of words in the user utterance is provided by the arrangement-Actor. This is because the selector-Actor should have a chance to see sequence of words in the user utterance in other ways to decide which word is more relevant and useful. This method prevents the position of words from affecting the final results. If we use the initial order of words, it is possible that the selector-Actor will get good rewards for some first words and add them to the context and discard words that are at the end of the user's utterance, while it is possible that the final words can give better information to the selector-Actor [11, 21, 24].

The Critics evaluates the actions selected by Actors and the Actors' parameters will be updated by these evaluations. Figure 1 is an overview of our approach.

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

3.1 Problem Statement and Motivation

Let $U = \{u_1, u_2, ..., u_m\}$ be a user utterance with m terms issued by a user at a turn of conversation, $C_0 = \{c_1, c_2, ..., c_n\}$ be the context of the conversation without any updates from user utterance, and $P = \{p_1, p_2, ..., p_k\}$ be a set of k items, documents, or entities which are supposed to be ranked by the conversational systems given the user utterance and the context. Using whole terms in the user utterance in the ranking can limit system performance. Sometimes using one term can be enough to find the target item, and using other information can confuse the system in finding it. Since the true value of each term in user utterance is unknown, we cannot use supervised learning to train our system. Also, the true value of each term depends on other terms in the user utterance which makes the problem more difficult. Therefore, we propose RelInCo, a reinforcement learning approach based on the Actor-Critic algorithm to estimate the true value of each term in the user utterance. The selector-Actor takes the context of the conversation and the user utterance in the order provided by arrangement-Actor in each step and decides to select or discard each term in the user utterance. A utility calculator is included to generate a reward for each action of Actors. The Critics take the taken actions, the states, and rewards and learn an action-value for each action. The Actors can be trained by this action-value. In the following sections, we present the technical details of components in Actors and Critics, respectively.

3.2 Architecture of Selector-Actor

The selector-Actor takes a version of user utterance U' (which is provided by the arrangement-Actor) and the context of the conversation C_0 as inputs and outputs a decision i.e., selection or rejection for each term in the user utterance that maximizes the ultimate goal of the conversation i.e., optimizing the evaluation metrics such as NDCG [12] or any desired objectives. Note that these objectives ideally can be designed based on user satisfaction signals. The evaluation metrics and objectives, which we aim to maximize, are non-differentiable. Therefore, we cannot maximize them in supervised learning. Reinforcement learning has been shown to be an effective approach to solve problems with non-differentiable metrics through policy gradient.

In reinforcement learning, an agent i.e., the Actor performs an action and receives a reward signal from the environment (e.g., utility calculator). The reward is a numerical value that indicates to the agent how good or bad the action was. The agent tries to find out the best actions to do or the best method to operate in the environment in order to complete its mission as efficiently as possible in order to earn more rewards. In this setting, our algorithm would be able to extract relevant information from users' utterances which ultimately maximize the evaluation metrics.

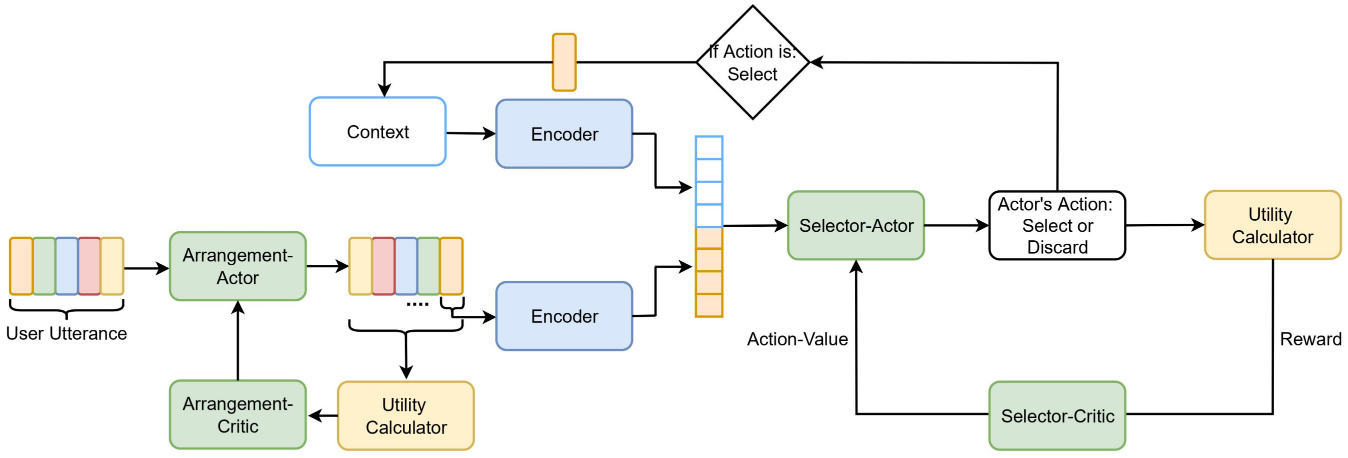

Figure 1: The workflow of RelInCo.

In the following, we describe how we model the selector-Actor, the action of this Actor, and the state.

Selector-Actor's Action: The selector-Actor is supposed to extract relevant words in the user utterance and add them to the context of the conversation. Therefore, by giving the current state, which is the current context and a word from the user utterance, the selector-Actor's action is to select or discard the word. After seeing all words in the user utterance, we have two sets: 1) selected words, and 2) discarded words. The selected words can be considered as relevant information and be useful for making the current state closer to the target item.

State: At the beginning, we get the output of the arrangement-Actor $U' = \{u'_1, u'_2, ..., u'_m\}$. Note that the arrangement-Actor can be trained either simultaneously or before the selector-Actor. At the first, the state is equal to the context of the conversation C_0 and the first word in U':

$$s_0 = (C_0, u'_1). \tag{1}$$

Given the action of the Actor for a word in $u'_t \in U'$, we can update the context of the conversation as follows:

$$C_t = \begin{cases} C_{t-1} + u'_t, & \text{if } a_t = \text{ select} \\ C_{t-1}, & \text{if } a_t = \text{ discard.} \end{cases} \tag{2}$$

Therefore, the state in the following step is $s_t = (C_t, u'_{t+1}), \quad t = \{0, 1, ..., m-1\}$.

Policy Network: A policy is a set of rules that an agent uses to determine which actions to do. The policy network's input is the current state s_t, and the Actor must forecast a probability distribution over possible actions. The Actor can take an action based on this probability distribution, which is selecting or discarding a word. We explain how we design the policy network's input, architecture, and output in the following.

Policy network's input: The selector-Actor takes the state $s_t = (C_t, u'_{t+1})$ at timestep t. To feed the state into the selector-Actor, we need to compute representations for the context and the word in the user utterance. These representations should satisfy two constraints: 1) be context-dependent to find relevant information from the user utterance based on the context of the conversation, and 2) be computationally efficient. To achieve these goals, we use sentence-BERT (SBERT) implementation [30], a variant of the pre-trained BERT network that employs Siamese and Triplet network architectures to generate semantically relevant phrase and word embeddings that can be compared using cosine-similarity (see section 4.1 for more details). Therefore, by using SBERT as our encoder we can generate representations for the context and the word in the user utterance as follows:

$$\vec{C}_t = enc(C_t) = \text{SBERT}(C_t) \tag{3}$$

$$\vec{u}'_{t+1} = enc(u'_{t+1}) = \text{SBERT}(u'_{t+1}), \tag{4}$$

where $\vec{C}_t$ and $\vec{u}'_{t+1} \in R^{1\times D}$ are embedding vectors for the context and the word in the user utterance. One can use other powerful encoders to achieve more accurate representations [39]. In the next step, we concatenate $\vec{C}_t$ and $\vec{u}'_{t+1}$ as an input of the selector-Actor:

$$\vec{s}_t = [\vec{C}_t || \vec{u}'_{t+1}]. \tag{5}$$

Policy network's architecture and output: We create a deep policy network to choose one of the actions based on the input of the selector-Actor. The policy network is designed of a feed-forward neural network with the following layers: 1) input layer z_0, hidden layer $l-1$, and output layer z_l. The input layer is fed by the state representation $z_0 = \vec{s}_t$. Each hidden layer $\vec{z}_i$ is a fully-connected layer $\vec{z}_i = \varphi(W_i.\vec{z_{i-1}} + b_i), \quad 1 \le i \le l-1$, where φ is a non-linear activation function e.g., Rectified Linear Units (ReLU). The output layer $\vec{z}_l$ is also a fully-connected layer, but the activation function for the output layer is a softmax function.

The softmax function returns a probability distribution across all potential outcomes (i.e., actions which can be selected or discarded in this task). As a result, if the policy network's parameters are represented by θ_π, the probability to select a word at timestep t given the current state is estimated as $\pi_{\theta_\pi}({a_t}^1|s_t) = \frac{e^{z_{l_1}}}{e^{z_{l_1}} + e^{z_{l_2}}}$. As a result, the probability to discard the word is $\pi_{\theta_\pi}({a_t}^2|s_t) = \frac{e^{z_{l_2}}}{e^{z_{l_1}} + e^{z_{l_2}}}$.

Reward: At timestep t, the Actor takes an action a_t given the state s_t. Given the action and the state, we need to generate a reward $R(a_t, s_t)$ to guide the selector-Actor. We wait until an episode finishes i.e., the selector-Actor extracts relevant words in the user's utterance U', then we compute the reward by utility calculator as follows:

$$R(a_t, s_t) = \begin{cases} \text{Utility}(s_t, P, C_t, C_0) & \text{If } t = m \\ 0 & \text{Otherwise} \end{cases} \tag{6}$$

where Utility(s_t, P, C_t, C_0) measures the reward depending on the evaluation function or any specified objective which is computed by the utility calculator (section 3.4).

3.3 Architecture of Arrangement-Actor

The arrangement-Actor takes the user utterance $U = \{u_1, u_2, ..., u_m\}$ and returns a new order of its words $U' = \{u'_1, u'_2, ..., u'_m\}$. The arrangement-Actor is supposed to return the order of words in the user utterance which maximizes an evaluation metric or objective which is computed by the utility calculator.

In the following, we describe how we model the arrangement-Actor, the action of this Actor, and the state.

Arrangement-Actor's Action: The arrangement-Actor is supposed to return a new order of words in the user utterance. Therefore, in each step, by giving the current state, the arrangement-Actor's action is to finds the most relevant order of words (Forward or Backward) in the user utterance as follows:

$$a_t = \begin{cases} 0 & \text{Backward i.e., } \{u_m, u_{m-1}, ..., u_1\} \\ 1 & \text{Forward i.e., } \{u_1, u_2, ..., u_m\} \end{cases} \tag{7}$$

Note that the arrangement-Actor's action should be able to choose any order of words in the user utterance e.g., $\{u_2, u_{m-1}, ..., u_1\}$. However, the number of possible actions in this situation can be order of factorial and it is not feasible in the real world. So, we simplified it to choosing between backward and forward order of words in the user utterance.

State: At the beginning, we get the original version of the user utterance U. In the first step, the state is equal to all words in U:

$$s_0 = \{u_1, u_2, ..., u_m\}. \tag{8}$$

Given the action of the arrangement-Actor, we update as follows:

$$s_t = \begin{cases} \{u_m, u_{m-1}, ..., u_1\} & \text{if } a_t = 0 \\ \{u_1, u_2, ..., u_m\} & \text{if } a_t = 1. \end{cases} \tag{9}$$

Policy Network: The policy network's input is the current state s_t, and the arrangement-Actor must forecast a probability distribution over possible actions. Note that the number of actions is equal to 2 i.e., backward or forward. We need to design a policy network that is flexible to the size of the input.

In the following, we explain how we design the policy network's input, architecture, and output.

Policy network's input: The arrangement-Actor takes the state s_t at timestep t. To feed the state into the arrangement-Actor, we need to compute a representation for the state which is the words in the user's utterance. To achieve this goal, same as the selector-Actor, we use sentence-BERT (SBERT) implementation to generate representations for all words in s_t as follows:

$$\{\vec{u}_i = enc(u_i) = \text{SBERT}(u_i) | u_i \in s_t\} \tag{10}$$

where $\vec{u}_i \in R^{1\times D}$ is an embedding vector. Therefore, we can write the state representation as follows:

$$\vec{s}_t = \begin{cases} \{\vec{u_m}, \vec{u_{m-1}}, ..., \vec{u_1}\} & \text{if } a_t = 0 \\ \{\vec{u_1}, \vec{u_2}, ..., \vec{u_m}\} & \text{if } a_t = 1. \end{cases} \tag{11}$$

Policy network's architecture and output: The policy network for the arrangement-Actor should be able to handle the different sizes of the input. In order to achieve this goal, we use a Recurrent Neural Network (RNN) with Gated Recurrent Units (GRU). The internal states of GRU are defined as follows:

$$\begin{aligned} \vec{z_t} &= \sigma(W_z \vec{u_t} + U_z \vec{h_{t-1}}) \\ \vec{r_t} &= \sigma(W_r \vec{u_t} + U_r \vec{h_{t-1}}) \\ \vec{h_t} &= (1 - \vec{z_t}).\vec{h_{t-1}} + \vec{z_t}.\vec{\hat{h}_t} \\ \vec{\hat{h}_t} &= \tanh[W\vec{u_t} + U(\vec{r_t}.\vec{h_{t-1}})]. \end{aligned} \tag{12}$$

We use the final hidden state $\vec{h_t}$ as the representation of the current state. Given $\vec{h_t}$, we use a feed-forward neural network and the softmax function to generate a probability distribution across two actions. As a result, if the policy network's parameters are represented by θ_η, the probability to choose an action at timestep t given the current state is estimated as:

$$\pi_{\theta_\eta}(a_t = 0|s_t) = \frac{e^{z_{l_0}}}{e^{z_{l_0}} + e^{z_{l_1}}}, \pi_{\theta_\eta}(a_t = 1|s_t) = \frac{e^{z_{l_1}}}{e^{z_{l_0}} + e^{z_{l_1}}} \tag{13}$$

where $\vec{z_l}$ is the output of the feed-forward neural network.

Reward: To generate $R(a_t, s_t)$ to guide the arrangement-Actor, we compute the reward by the utility calculator same as Eq.6.

3.4 Utility Calculator

Utility calculator takes the context of the conversation, the word in the user utterance, and the descriptions of some non-relevant items and relevant items and returns a reward as a signal that helps the Actors to generate optimal actions. Figure 2 illustrates the utility calculator workflow. The reward is supposed to be computed based on the rank of the target item. In this way, the Actors try to get more rewards, i.e., improving the rank of the target items in the ranking list, by selecting relevant information in the users' utterances. However, in most cases, there are lots of non-relevant items in a collection. So, instead of using all non-relevant items, we select N non-relevant items by negative sampling $P_{N,p_u} = \{p_1, p_2, ..., p_N\} \subset P \setminus p_u$ where p_u is the relevant item. Negative sampling is proposed by Mikolov et al. [18] and has now been extensively used for machine learning and information retrieval [1, 29]. For each item we collect a short description by using Wikipedia and DBpedia [6] (see section 4.3.1 for more details) to find more relevant words in the user's utterance. In the next step, we convert each description to its representation by SBERT encoder $\vec{p_i} = enc(p_i) = \text{SBERT}(p_i)$.

To evaluate the word in the user utterance u'_{t+1} in finding the relevant item, we include this word to the context $C'_t = C_t + u'_{t+1}$ and compute its representation $\vec{C'_t} = enc(C'_t) = \text{SBERT}(C'_t)$. Given the context of the conversation without any updates $\vec{C_0}$, we first rank relevant item and non-relevant items by cosine similarity between items and context representations:

$$\text{score}(C_0, p_i) = \cos(\vec{C_0}, \vec{p_i}). \tag{14}$$

Likewise, we can compute the score of each item and the updated context C'_t, score$(C'_t, p_i) = \cos(\vec{C'_t}, \vec{p_i})$. By generating these scores we can rank the items given the context and the updated context and compute an evaluation function eval(C, P_{N,p_u}, p_u) e.g., NDCG [12]. In the following, we show it abridged eval(C).

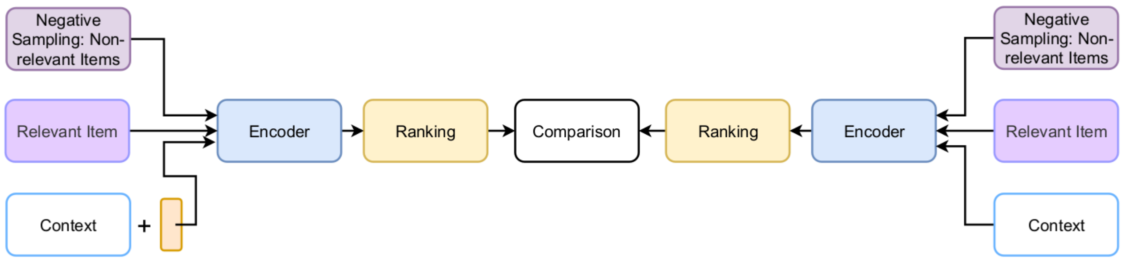

Figure 2: An illustration of Utility Calculator.

Finally, by comparing the evaluation function for the context and the updated context we can compute a signal as follows:

$$\text{Utility}(s_t, P, C_t, C_0) = \text{eval}(C_t') - \text{eval}(C_0). \tag{15}$$

If the utility is greater than zero, this means that the Actors find relevant information from the user's utterance which can improve the performance. Otherwise i.e., the utility is less than zero, and the Actors should avoid these actions.

3.5 Architecture of Critics

In the Actor-Critic algorithm, the Critic takes the state, the taken action, and the reward and learns an action-value function $Q(s, a)$ to evaluate the action in the current state. In other words, instead of assigning a constant reward for each action in a state, we train it with a Critic. The action-value can be used by the Actor to update its parameters. Because the state spaces are so large in our task, the action-value function $Q(s, a)$ can be more effective in a non-linear fashion. Therefore, we build our Critics with deep neural network and GRU to estimate $Q(s, a)$ same as the Actors (see section 4.1 for more details). The Critics network's parameters are represented by θ_μ and θ_ρ for selector-Critic and arrangement-Critic, respectively.

In the following sections, we'll go over how the Critics' parameters will be updated.

3.6 Training and Test Procedure

3.6.1 ***Critics Training Procedure.*** The following is the loss function for the Critic, which is known as Temporal Difference (TD) learning and is derived from Bellman equation [4]:

$$\mathcal{L}(\theta_\mu) = \mathbb{E}_{s_t,a_t,s_{t+1},R}[(R + \gamma Q_{\theta_{\mu'}}(s_{t+1}, \pi_{\theta_{\pi'}}(s_{t+1})) - Q_{\theta_\mu}(s_t, a_t))^2], \tag{16}$$

where θ_μ, γ, and R denote all parameters in the selector-Critic, discount factor, and reward, respectively. The parameters of the selector-Critic from the previous iteration are shown in $\theta_{\mu'}$ and will be fixed in the loss function $\mathcal{L}(\theta_\mu)$. The derivations of the loss function with respect to parameters θ_μ is $\theta_\mu \longleftarrow \theta_{\mu'} - \alpha \nabla_{\theta_\mu} \mathcal{L}(\theta_\mu)$ where α is the learning rate for the selector-Critic training. The parameters of the arrangement-Critic are updated in the same way.

3.6.2 ***Actors Training Procedure.*** The loss function forces the Actor to predict the true action for each word as follows:

$$\mathcal{L}(\theta_\pi) = \log \pi_\theta(a|s), \tag{17}$$

where $\pi_\theta(a|s)$ is a probability of an action taken by the Actors given the state. However, this loss function might have high variance in the training. To improve this problem we use Proximal Policy Optimization (PPO) [33]. PPO guarantees that the updated policy isn't too dissimilar from the old policy in order to maintain a low level of training variation. This approach is proposed and implemented by OpenAI and showed remarkable performance. By using PPO, we replace the loss function as follows:

$$\mathcal{L}_{CLIP}(\theta_\pi) = \min(r_t(\theta), clip(r_t(\theta), 1 - \epsilon, 1 + \epsilon)), \tag{18}$$

where $r_t(\theta)$ is the probability ratio between the action under the current policy and the action under the previous policy as follows:

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}, \tag{19}$$

and ϵ is a hyperparameter that shows the limit of the range within which the update is allowed. In other words, by using this loss function we make sure that the old policy is not too different from the new policy.

Finally, we utilize REINFORCE algorithm [40] to update parameters in the selector-Actor as follows:

$$\nabla_{\theta_\pi} J(\theta_\pi) \approx \mathbb{E}_{\pi_\theta}[\nabla_{\theta_\pi} \mathcal{L}_{CLIP}(\theta_\pi) Q_{\theta_\mu}(s, a)] \tag{20}$$

where $Q_{\theta_\mu}(s, a)$ denotes the action value which is the output of the selector-Critic. Now, we can update the parameters in the selector-Actor by $\theta_\pi \longleftarrow \theta_{\pi'} - \lambda \nabla_{\theta_\pi} \mathcal{L}(\theta_\pi)$ where λ is the learning rate of the selector-Actor. The parameters of the arrangement-Actor are updated in the same way.

3.6.3 ***The Test Procedure.*** At test time, we simply use the Actors θ_η and θ_π to generate an order of words in the user's utterance and select or discard them. The arrangement-Actor takes the user's utterance and returns a new order of its words. Then, we feed this sequence as a new version of the user's utterance in the selector-Actor. The selector-Actor takes the context of the conversation and the new version of the user's utterance. The selector-Actor returns selected and discarded words. We update the context of the conversation by selecting words as relevant information. In the end, we used the updated context to retrieve items.

Table 1: Basic statistics of the experimental datasets.

Dataset	#Dialog	#Utterance	Domains
REDIAL	10,006	182,150	Movie
OpenDialKG	13,802	91,209	Movie, Book

Table 2: Comparison of proposed model RelInCo (trained to maximize NDCG@100 as eval function in Eq.15) and baselines. The superscript ▲ indicates that the improvements over all baselines are statistically significant.

Dataset	Metric	ReDial	KBRD	KGSF	RelInCo
REDIAL	NDCG@1	0.0127	0.0389	0.0347	**0.0415**▲
	NDCG@10	0.0326	0.1004	0.0975	**0.1136**▲
	NDCG@50	0.0586	0.1396	0.1398	**0.1531**▲
OpenDialKG	NDCG@1	0.0117	0.2271	0.2220	**0.2410**▲
	NDCG@10	0.0230	0.3370	0.3287	**0.3427**
	NDCG@50	0.0403	0.3556	0.3465	**0.3698**▲

4 EXPERIMENTS

4.1 Experimental and parameter settings

We utilize *CRSLab* [45] for our experiments, which is an open-source toolkit for conversational systems that includes a number of state-of-the-art models and datasets. We used Tensorflow to implement and train our model[1]. Our code is publicly available[2]. The network parameters in the Actors and Critics models is trained using the Adam optimizer [13], which uses the back-propagation algorithm [32]. The learning rate for the Actors and Critics were selected from $[1e-3, 5e-4, 1e-4, 5e-5]$, and $[1e-2, 5e-3, 1e-3, 5e-4]$, respectively. For the selector-Actor and selector-Critic networks we used 3 or 2 hidden layers, respectively. The layer size was selected from $\{400, 200, 100, 50\}$. We used Rectified Linear Units (ReLU) activation function for hidden layers and Softmax for the last layer of the selector-Actor. For arrangement-Actor and arrangement-Critic, the number of units in GRU is selected from $\{10, 100, 200\}$. Because we want Critics to learn faster than Actors, the learning rate for Critics is higher than Actors. The discount factor in Eq.16 was set to 0.99 since it has been shown this value works well in the reinforcement learning [14]. The number of non-relevant samples for each sample is set to 100. ϵ in Eq.18 is set to 0.2 which is proposed by the original paper [33]. For our encoder (i.e., SBERT) we used all-MiniLM-L6-v2 which is trained on a large and diverse dataset of over 1 billion training pairs [30]. Note that in the inference time we take the top 100 items or entities using KGSF as a candidate set for our model and re-rank them by our model. Re-ranking the top items or documents, as Diaz [9] demonstrated, is as successful as obtaining from the whole collection at a far cheaper cost.

4.2 Evaluation Measures

For all experiments, we report NDCG@{1, 10, 50} [12] which are used as retrieval and recommendation metrics in *CRSLab*. Note that in our training we used NDCG@100 as eval function in Eq.15. Statistically significant differences of performance are determined using two-tailed paired t-test at 95% confidence level ($p\_value < 0.05$).

4.3 Datasets

We use two commonly-used human-annotated conversational systems datasets 1) OpenDialKG [25], and 2) REDIAL [16].

[1]https://www.tensorflow.org/

[2]https://drive.google.com/drive/folders/1QV2Ypz_nEOmge5Ba0jcu4CrhJa18CMd5?usp=sharing

OpenDialKG [25] is an Open-ended Dialog with knowledge graph (KG) corpus which includes two different tasks: 1) Recommendation, and 2) Chit-chat. We use the first task in our work. OpenDialKG contains 13, 802 human-to-human role-playing dialogs with 91, 209 utterances. For the recommendation task, we have movies and books domains in this dataset.

REDIAL is a REcommendations through DIALog dataset which is a conversational recommendation dataset collected by [16]. REDIAL is constructed through Amazon Mechanical Turk (AMT). In seeker-recommender pairs, the AMT workers generate conversations for recommendations on movies. It contains 10, 006 conversations consisting of 182, 150 utterances related to 51, 699 movies.

Table 1 shows the basic statistics of these two datasets. The datasets are split into training, validation, and test sets using a ratio of 8:1:1.

4.3.1 Collecting Item Descriptions.

To find more relevant information in the users' utterances, we need to enrich the descriptions of the items or entities. Therefore, we collect more information about each item or entity from Wikipedia and DBpedia [6].

For each item in the REDIAL dataset, we collect the first 100 words of its abstract from DBpedia. We clean it by removing non-alphabet chars from descriptions.

For entities in the OpenDialKG, we use Wikipedia to collect the first 100 words of its summary and clean it. We release this information to help other researchers in this task.

4.4 Baselines

The following methods are baseline models for the experiments:

- ReDial [16]: ReDial has been proposed with REDIAL dataset. The recommendation module is built on auto-encoder [10] and a sentiment analysis module.
- KBRD [7]: KBRD or Knowledge-Based Recommender Dialog system which is integration the recommender system and the dialog generation system. This model utilizes DBpedia [6] to enhance the semantics of contextual items or entities.
- KGSF [46]: KGSF is a novel KG-based semantic fusion approach for CRS. KGSF employs a KG-enhanced recommendation component for making accurate recommendations. To improve data representations in CRSs, KGSF uses both word-oriented and entity-oriented knowledge graphs (KG), as well as Mutual Information Maximization to align the word-level and entity-level semantic spaces.

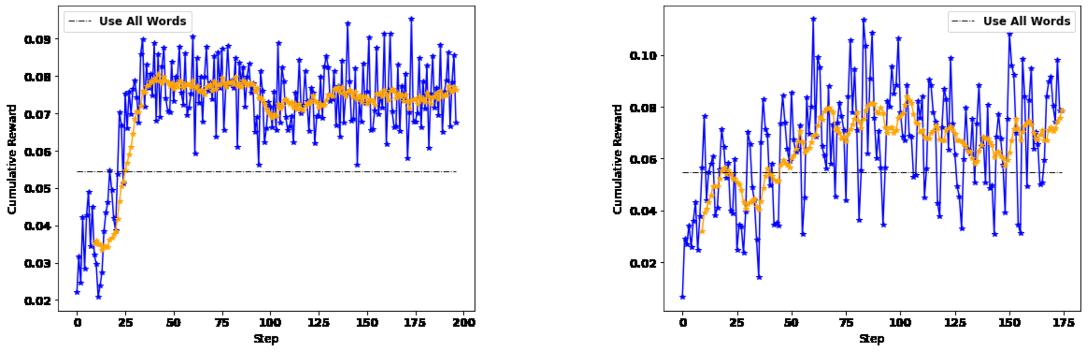

Figure 3: The cumulative reward over training for selector-Actor (left), and arrangement-Actor (right).

4.5 Results and Discussion

4.5.1 ***Comparison with the Baselines.*** In the first experiment, we evaluate our model against baselines. Note that in our experiments, we try to maximize NDCG@100 as eval function in Eq.15. The results for our model and baselines are reported in Table 2. The first observation from Table 2 is that the results of KBRD and KGSF are better than ReDial in the two datasets. The reason is that both of these models try to use the knowledge graph in this task. In other words, they use other information to understand user needs. But we argue that it is important to extract relevant information before using other knowledge to understand users' utterances. According to the Tabel 2, our model significantly outperforms other baselines in most cases. The reason is that our model uses other information (such as baseline models) and at the same time maximizes evaluation metrics to find more relevant information in users' utterances.

4.5.2 ***Analysis of the Rewards.*** In this section, we analyze the rewards earned by both Actors. We want to make sure that both Actors are learning to get more rewards which means that they are improving the retrieval performance.

In the training, we first let the arrangement-Actor train for 3 or 4 steps. Then we freeze the parameters of the arrangement-Actor and start to train the selector-Actor. The reason is that the output of the arrangement-Actor is the input of the selector-Actor. Therefore, we need to make sure that the arrangement-Actor generates better order of words in the user's utterance compared to the original one.

Then, we check that the selector-Actor can extract relevant information from the user's utterance.

To show the learning curve of both Actors in RelInCo, we depicted the cumulative reward that each Actor has earned in the training in Figure 3. In this experiment, we just report the results of the REDIAL dataset for the sake of space. However, we had similar observations for another dataset. According to this figure, both Actors earn more rewards after some steps. The arrangement-Actor finds more effective orders of words in the users' utterances since we show the difference between the performance of the system with the original context and updated context by all words in the user's utterance (without reordering) in both figures. This means that if we reorder words in the user's utterance by arrangement-Actor, we can achieve better performance. Also, we can see that the selector-Actor is more stable compared to the arrangement-Actor since the selector-Actor learns to discard non-relevant information and just use relevant information.

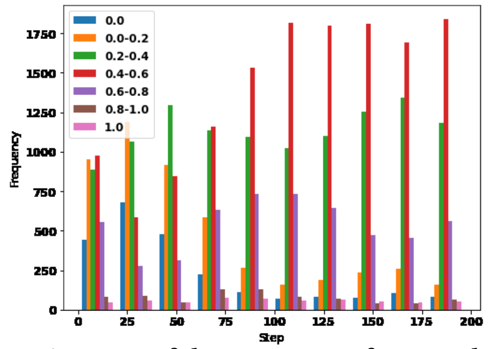

Figure 4: Histogram of the percentage of terms selected by the selector-Actor at each stage of the training.

4.5.3 ***Analysis of the Number of Words Selected by Selector-Actor.*** In this section, we analyze the percentage of words of the users' utterances selected by the selector-Actor in the training. The result of this experiment is shown in Figure 4. At the first steps, the selector-Actor has not selected even one word from the user's utterance for some of the training data (the blue bar that shows zero percent of words are selected). By increasing the training time, the selector-Actor learns to select at least some words to get more rewards. Also, it is obvious from the figure that the selector-Actor discards about 50% of the words and selects others as relevant information.

5 CONCLUSIONS AND FUTURE WORK

We proposed RelInCo, a reinforcement-based algorithm to extract relevant information from the user's utterance in conversational search and recommendation. RelInCo uses two Actors to reorder and select words from the user's utterance and use in the retrieval in the next turn of the conversation. RelInCo is able to maximize any evaluation metrics or objectives and can easily be replaced by any user satisfaction signals. To reduce the high variances in the training, we use Proximal Policy Optimisation which was proposed recently in reinforcement learning. Our results show that our model is able to increase retrieval performance while discarding some of the words in the user's utterance. This can show extracting relevant information from the user's utterance in conversational search and recommendation is essential.

We used on-policy reinforcement learning in our model. However, the training time and sampling are not efficient in the on-policy approach. This can be a bigger problem in information retrieval and recommender systems. Finding more efficient sampling in this area can be an interesting future work. We used SBERT as our encoder in RelInCo, and these representations of words are input for our Actors and affect the results. So, finding more rich representations for words and the context of the conversation would help to increase the performance. In this study, we use relevant information for finding the target item or entity. However, it is important to use this information in generating question in the conversation which we leave it to future work.

6 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 sponsor.

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] Dzmitry Bahdanau, Philemon Brakel, Kelvin Xu, Anirudh Goyal, Ryan Lowe, Joelle Pineau, Aaron Courville, and Yoshua Bengio. 2016. An actor-critic algorithm for sequence prediction. *arXiv preprint arXiv:1607.07086* (2016).

[3] Nicholas J Belkin, Colleen Cool, Adelheit Stein, and Ulrich Thiel. 1995. Cases, scripts, and information-seeking strategies: On the design of interactive information retrieval systems. *Expert systems with applications* 9, 3 (1995), 379–395.

[4] Richard Bellman. 1966. Dynamic programming. *Science* 153, 3731 (1966), 34–37.

[5] Keping Bi, Qingyao Ai, Yongfeng Zhang, and W Bruce Croft. 2019. Conversational product search based on negative feedback. In *CIKM'19*. 359–368.

[6] Christian Bizer, Jens Lehmann, Georgi Kobilarov, Sören Auer, Christian Becker, Richard Cyganiak, and Sebastian Hellmann. 2009. Dbpedia-a crystallization point for the web of data. *Journal of web semantics* 7, 3 (2009), 154–165.

[7] Qibin Chen, Junyang Lin, Yichang Zhang, Ming Ding, Yukuo Cen, Hongxia Yang, and Jie Tang. 2019. Towards knowledge-based recommender dialog system. *arXiv preprint arXiv:1908.05391* (2019).

[8] W Bruce Croft and Roger H Thompson. 1987. I3R: A new approach to the design of document retrieval systems. *Journal of the american society for information science* 38, 6 (1987), 389–404.

[9] Fernando Diaz. 2015. Condensed list relevance models. In *ICTIR'15*. 313–316.

[10] Junhua He, Hankz Hankui Zhuo, and Jarvan Law. 2017. Distributed-representation based hybrid recommender system with short item descriptions. *arXiv preprint arXiv:1703.04854* (2017).

[11] Ayyoob Imani, Amir Vakili, Ali Montazer, and Azadeh Shakery. 2019. An Axiomatic Study of Query Terms Order in Ad-hoc Retrieval. In *ECIR'19*. Springer, 196–202.

[12] Kalervo Järvelin and Jaana Kekäläinen. 2002. Cumulated gain-based evaluation of IR techniques. *TOIS'02* 20, 4 (2002), 422–446.

[13] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[14] Vijay R Konda and John N Tsitsiklis. 2000. Actor-critic algorithms. In *NIPS'00*. 1008–1014.

[15] 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 *WSDM'20*. 304–312.

[16] Raymond Li, Samira Ebrahimi Kahou, Hannes Schulz, Vincent Michalski, Laurent Charlin, and Chris Pal. 2018. Towards deep conversational recommendations. *NIPS'18* 31 (2018).

[17] Lizi Liao, Ryuichi Takanobu, Yunshan Ma, Xun Yang, Minlie Huang, and Tat-Seng Chua. 2019. Deep conversational recommender in travel. *arXiv preprint arXiv:1907.00710* (2019).

[18] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. *NIPS'13* 26 (2013).

[19] Ali Montazeralghaem and James Allan. 2022. Learning Relevant Questions for Conversational Product Search using Deep Reinforcement Learning. In *WSDM'22*. 746–754.

[20] Ali Montazeralghaem, James Allan, and Philip S Thomas. 2021. Large-scale Interactive Conversational Recommendation System using Actor-Critic Framework. In *RecSys'21*. 220–229.

[21] Ali Montazeralghaem, Razieh Rahimi, and James Allan. 2020. Relevance Ranking Based on Query-Aware Context Analysis. *ECIR'20* 12035 (2020), 446.

[22] Ali Montazeralghaem, Hamed Zamani, and James Allan. 2020. A reinforcement learning framework for relevance feedback. In *SIGIR'20*. 59–68.

[23] Ali Montazeralghaem, Hamed Zamani, and Azadeh Shakery. 2016. Axiomatic analysis for improving the log-logistic feedback model. In *SIGIR'16*. 765–768.

[24] Ali Montazeralghaem, Hamed Zamani, and Azadeh Shakery. 2017. Term proximity constraints for pseudo-relevance feedback. In *SIGIR'17*. 1085–1088.

[25] Seungwhan Moon, Pararth Shah, Anuj Kumar, and Rajen Subba. 2019. Opendialkg: Explainable conversational reasoning with attention-based walks over knowledge graphs. In *ACL'19*. 845–854.

[26] Romain Paulus, Caiming Xiong, and Richard Socher. 2017. A deep reinforced model for abstractive summarization. *arXiv preprint arXiv:1705.04304* (2017).

[27] 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).

[28] Filip Radlinski and Nick Craswell. 2017. A theoretical framework for conversational search. In *CHIIR'17*. 117–126.

[29] Razieh Rahimi, Ali Montazeralghaem, and James Allan. 2019. Listwise neural ranking models. In *ICTIR'19*. 101–104.

[30] Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084* (2019).

[31] Steven J Rennie, Etienne Marcheret, Youssef Mroueh, Jerret Ross, and Vaibhava Goel. 2017. Self-critical sequence training for image captioning. In *CVPR'17*. 7008–7024.

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

[33] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).

[34] 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.

[35] 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.

[36] Yueming Sun and Yi Zhang. 2018. Conversational recommender system. In *SIGIR'18*. 235–244.

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

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

[39] Lakshmi Vikraman, Ali Montazeralghaem, Helia Hashemi, W Bruce Croft, and James Allan. 2021. Passage Similarity and Diversification in Non-factoid Question Answering. In *ICTIR'21*. 271–280.

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

[41] 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).

[42] Xiaoying Zhang, Hong Xie, Hang Li, and John CS Lui. 2019. Toward building conversational recommender systems: A contextual bandit approach. *arXiv preprint arXiv:1906.01219* (2019).

[43] 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.

[44] Xiangyu Zhao, Long Xia, Liang Zhang, Zhuoye Ding, Dawei Yin, and Jiliang Tang. 2018. Deep reinforcement learning for page-wise recommendations. In *RecSys'18*. 95–103.

[45] Kun Zhou, Xiaolei Wang, Yuanhang Zhou, Chenzhan Shang, Yuan Cheng, Wayne Xin Zhao, Yaliang Li, and Ji-Rong Wen. 2021. CRSLab: An Open-Source Toolkit for Building Conversational Recommender System. *arXiv preprint arXiv:2101.00939* (2021).

[46] Kun Zhou, Wayne Xin Zhao, Shuqing Bian, Yuanhang Zhou, Ji-Rong Wen, and Jingsong Yu. 2020. Improving conversational recommender systems via knowledge graph based semantic fusion. In *SIGKDD'20*. 1006–1014.

[47] Kun Zhou, Yuanhang Zhou, Wayne Xin Zhao, Xiaoke Wang, and Ji-Rong Wen. 2020. Towards topic-guided conversational recommender system. *arXiv preprint arXiv:2010.04125* (2020).

[48] Jie Zou, Yifan Chen, and Evangelos Kanoulas. 2020. Towards question-based recommender systems. In *SIGIR'20*. 881–890.