A Review on Reinforced Recommender Systems

Kiarash Geraili[1], M.Akbari[2]

***Abstract*— In this paper, we try to formulate a Reinforced Recommender System or a ranking system that uses the Reinforcement Learning (RL) approaches by illustrating how to determine the requirements of an RL problem, which are states, actions, and reward function. For this purpose in the first section, we talked about the fundamentals of RL and Ranking. Then, we tried to summarize the advantages of Reinforced Recommender Systems over traditional ways. Finally, in the last section, we search for our formulation in some feature works in this area.**

I. Introduction

Through the development of technologies over the past decades, shopping online has become an important part of people's daily life, resulting in E-commerce giants like Amazon and eBay. recommender systems also play a crucial role in mitigating information overload by suggesting user's personalized items or services. Information retrieval intrinsically is modeled as a ranking model where search framework attempts to find a ranked list of documents based on a user query.

The vast majority of traditional recommender systems consider the recommendation procedure as a static process and make recommendations following a fixed strategy. In order to avoid this disadvantage, we try to use RL approaches. This paper exploits reinforced learning to a ranked list of documents towards addressing information retrieval tasks. To accomplish this end, we formalize keystones of Reinforced Ranking (RR) problem and show potential scenarios of an information retrieval task with its reinforced learning models. To do that, in the first section, we introduce RL concepts and, more specifically, Multi armed-bandit and Markov Decision Processes. In the next section, we introduce the ranking problem and its relation to RL, which named with a reinforced recommender system. After covering the fundamental concepts, we formulate the reinforced recommender system based on some featured works on this topic, and finally, in the last section, we trace our work in other papers to evaluate our formulation.

II. Background

A. Reinforcement Learning

Reinforcement learning is a computational approach to understanding and automating goal-directed learning and decision making. It is distinguished from other computational methods by emphasizing learning by an agent from direct interaction with its environment, without relying on exemplary supervision or complete models of the environment [1]. RL arises from learning from interaction with an environment to achieve long-term goals. Reinforcement learning uses the formal framework of Markov decision processes-which we will discuss later-to define the interaction between a learning agent and its environment in terms of *States*, *Actions*, and *Reward*. In supervised learning, we exploit training data to learn from labeled samples provided by an external supervisor. Each example includes a feature vector corresponding to a data item and its corresponding label [1]. Learning is to build a model from data so that it can predict the correct label for unseen data.

Each example is a description of a data item together with a specification of the label of the correct action. The system should take action for a data item, which is often to identify a category to which the data item belongs. The objective of supervised approaches is to allow us to learn generalizable encountered patterns so that it acts correctly in cases not present in the training set.

In interactive problems, it is often impractical to obtain examples of desired behavior that are both correct and representative of all the situations in which the agent has to act. In an uncharted territory where one would expect learning to be most beneficial, an agent must be able to learn from its own experience. The importance of RL in ranking appears here due to the interaction between the user and search engine. More specifically, a search engine acts as an agent that should learn how to recommend items or how to ranked items based on user's behavior. There is another prominent question: why should we use RL approaches instead of Supervised approaches? (we will talk about it in next section)

First of all, the policy defines the learned strategy in which an agent is learning to behave correctly at a given time. Roughly speaking, a policy is a mapping from perceived states of the environment to actions to be taken when in those states. A reward signal defines the goal of a reinforcement learning problem. On each time step, the environment sends to the reinforcement learning agent a single number called the reward. The agent's sole objective is to maximize the total reward it receives over the long run. The reward signal thus defines what good and bad events for the agent are.

According to *Reward Hypothesis*[1]: all of what we mean by goals and purposes can be well thought as the maximization of the expected value cumulative sum of received scalar signal (reward signal). For example, consider we want our agent to learn a pattern to escape from a maze. Defining a (+1) reward for each time that the agent goes out of the maze does not seem reasonable. Because in this definition, the intelligent agent just learns how to escape from the maze. Nevertheless, by defining a (-1) for each time step in which the agent is in the maze, we not only learn the agent to

escape from the maze but also learns to do it as soon as possible. (cause remaining in the maze make the cumulative reward for becoming worst and worst)

Thus the rewards we set up truly indicate what we want to be accomplished. In particular, the reward signal is not the place to impart to the agent prior knowledge about how to achieve what we want it to do. As an illustration, a chess-playing agent should be rewarded only for winning, not for achieving sub-goals such as taking its opponent's pieces or gaining control of the center of the board. If reaching these types of sub-goals were rewarded, then the agent might find a way to achieve them without achieving the real goal.

- For instance, it might find a way to take the opponent's pieces even at the cost of losing the game. Consider the way you define the reward is the way you want the agent to achieve a goal, not how you want it made.

Whereas the reward signal indicates what is good in an immediate sense, a value function specifies what is right in the long run. Roughly speaking, the value of a state is the total amount of reward an agent can expect to accumulate over the future, starting from that state. Rewards determine the immediate, intrinsic desirability of environmental states. Still, values indicate the long-term desirability of states after taking into account the states that are likely to follow, and the rewards available in those states. For example, a state might always yield a low immediate reward but still have a high value because it is regularly followed by other states that yield high rewards. Alternatively, the reverse could be true. To make an analogy, rewards are somewhat like pleasure (if high) and pain (if low), whereas values correspond to a more refined and farsighted judgment of how pleased or displeased, we are that our environment is in a particular state.

This framework is intended to be a simple way of representing the essential features of the artificial intelligence problem. These features include a sense of cause and effect, a sense of uncertainty and non-determinism, and the existence of explicit goals. The concepts of value and value function are essential to most of the reinforcement learning methods that we consider. We take the position that value functions are crucial for efficient search in the space of policies. The use of value functions distinguishes reinforcement learning methods from evolutionary methods that search directly in policy space guided by scalar evaluations of entire policies. The fourth and final element of some reinforcement learning systems is a model of the environment. This is something that mimics the behavior of the environment, or more generally, that allows inferences to be made about how the environment will behave.

For example, given a state and action, the model might predict the resultant next state and next reward. Models are used for planning, by which we mean any way of deciding on a course of action by considering possible future situations before they are actually experienced. Methods for solving reinforcement learning problems that use models and planning are called model-based methods, as opposed to more straightforward model-free methods that are explicitly trial-and-error learners viewed as almost the opposite of planning.

1) Markov Decision Process (MDP): Formally MDP describes an environment for an RL problem. At the first step, we can assume that the environment is fully observable, so nothing about the way that environment work is hidden from our sights. When Markovian property holds for our model, We can say that the current state of the environment completely characterizes the process[1]. It is the intuition of a Markov process or Markov chain. Markov Decision Process refers to a Markov process, which also has an action of decision making. Almost all the RL problems can be formulated as an MDP. For example, optimal control primarily deals with continuous MDPs, and partially observable problems can also be converted by MDPs and also a bandit problem (which we will discuss about it later) is a one-state MDP. Markov property is crucial almost in every RL problem. It illustrates that the future of the interaction between agent and environment is independent from the past given the present. Mathematically:

$$P[S_t + 1|S_t] = P[S_t + 1|S_1, S_2, S_3, ..., S_t]$$

In other words, it means that the state captures all relevant information from history. Thus one the state is known, all the history can be thrown away. More wisely, a Markov process is a memory-less random process, i.e. a sequence of random states $S_1, S_2, S_3, \dots$ with the Markov property.

A Markov process (or a Markov chain) is a tuple like ¡S,P¿ in which S is a set of states, and P is a set of transition probabilities. $P_{ss} = P[S_{t+1} = S'|S_t = s]$. Markov property seems to be one of the most important requirements of an RL problem in the way that if an environment has the Markov property, then its one-step dynamics enable us to predict the next state and expected next reward given the current state and action. One can show that, by iterating this equation, one can predict all future states and expected rewards only from a knowledge of the current state as well as would be possible given the complete history up to the current time. It also follows that Markov states provide the best possible basis for choosing actions. That is, the best policy for choosing actions as a function of a Markov state is just as good as the best policy for choosing actions as a function of complete histories.

2) Multi-Armed Bandit (MAB): As an illustration, we want to explain one of the RL scenarios, which is Multi-Armed Bandit (MAB). Here is the k-armed bandit problem: You are faced repeatedly with a choice among k different options that act as your actions. For example, consider that you are facing a slot machine with k armed. After each decision (in our example, each decision is choosing each armed to pull), you receive a numerical reward chosen from a stationary probability distribution, which is unknown. The reward depends on the action you selected (or the arm which you pull). Your objective is to maximize the expected total reward over a while. It is evident that in this RL problem, we have one state. In this problem, if you knew the value

of each action, then it would be trivial to solve the k-armed bandit problem: you would always select the action with the highest value. We denote the action selected on time step t as A_t, and the corresponding reward as R_t. The value then of an arbitrary action a, denoted $q^*(a)$, is the expected reward given that a is selected:

$$q^*(a) \ := \ E[R_t|A_t = a]$$

We assume that you do not know the action values with certainty, although you may have an estimation. We denote the estimated value of action a at time step t as $Q_t(a)$. We want $Q_t(a)$ to be close to $q*(a)$.

If you maintain estimates of the action values, then at any time step, there is at least one action whose estimated value is highest. We call these greedy actions. When you select one of these actions, we say that you are exploiting your current knowledge of the values of the actions. If instead, you select one of the non-greedy actions, then we say you are exploring, because this enables you to improve your estimate of the non-greedy action's value. In this point of view, this example is a simple one, because we are looking for the best strategy just by picking the best actions. In many other RL problems, we are looking for an optimum policy with the use of state-value functions or, in some cases, state-action value function. These two parameters help us find out which states or state-actions are better to maximize rewards. In those problems, as the state space could be so large so the exploration/exploitation could be harder.

As a solution to the MAB problem, simply we can consider that the true value of an action is the mean reward when that action is selected[1]. One natural way to estimate this is by averaging the rewards:

$$Q_t(a) \ = \ \frac{\sum_{i=1}^{t-1} R_i}{Number\ of\ times\ a\ taken\ prior\ to\ t}$$

The most straightforward action selection rule is to select one of the actions with the highest estimated value, that is, one of the greedy actions as defined in the previous section. But it is not necessarily the best policy. Greedy action selection always exploits current knowledge to maximize immediate reward. Still, if we have not had enough experiences from other actions, we would accumulate less long term rewards from this same arm.

An alternative is to behave greedily, with a difference. Every once in a while, say with small probability like ϵ, instead of selecting the best choice, select randomly from among all the actions with equal probability (independently of the action-value estimates). We call methods using this near-greedy action selection rule, ϵ-greedy methods. An advantage of these methods is that, in the limit as the number of steps increases, every action will be sampled an infinite number of times. Thus, ensuring that all the $Q_t(a)$ converge to $q^*(a)$. This implies that the probability of selecting the optimal action converges to greater than 1–ϵ, that is, to near certainty.

There is a well-known alternative named greedy time dependent. In this approach we change the definition of ϵ and we relate it to time like this:

$$\epsilon = \frac{1}{\sqrt{t+1}}$$

This formulation can be reasonable, because at the first value of t is small, and the value of epsilon becomes bigger as we expect. Because at the beginning of the learning algorithm, we do not have enough information about the distribution of the arms. Thus we need to explore more. By the condition of time infinity, of we have a limited number of arms, we can be assured that too much exploration is not reasonable. Therefore, as time goes on, we need less exploration and more exploiting. So that is why we relate epsilon reversely to time.

Back to the usual method, the action-value methods we have discussed so far all estimate action values as sample averages of observed rewards. We now turn to the question of how these averages can be computed in a computationally efficient manner, in particular, with constant memory and constant per-time-step computation. To simplify notation, we concentrate on a single action.

Let R_i now denote the reward received after the i'th selection of this action, and let Q_n denote the estimate of its action value after it has been selected $n-1$ times, which we can now write simply as:

$$Q_n := \frac{R_1 + R_2 + ... + R_{n-1}}{n-1}$$

As in this way of calculation, we need to save each step; we can incrementally calculate Q_n to avoid complexity:

$$Q_{n+1} \ = \ Q_n + \frac{1}{n}[R_n \ - \ Q_n]$$

or in the other word:

$$NewEstimate \leftarrow OldEstimate + StepSize[Target - OldEstimate]$$

The expression $[Target \ - \ OldEstimate]$ is an error in the estimate. It is reduced by taking a step toward Target. The target is presumed to indicate a desirable direction in which to move, though it may be noisy. In the case above, for example, target is the n'th reward. Note that the step-size parameter $(StepSize)$ used in the incremental method described above changes from time step to time step.

B. What is Ranking?

A ranking is a relationship between a set of items such that, for any two items, the first is either 'ranked higher than', 'ranked lower than' or 'ranked equal to' the second. In mathematics, this is known as a weak order or total preorder of objects. It is not necessarily a total order of objects because two different objects can have the same ranking. In search engines, once the search engine has predicted the user's underlying intent, its first task is to retrieve a pool of documents that are relevant to that intent. This process is very complicated and typically spans many different servers and indices. A primary way to do this is to retrieve a list of

all documents that exist which contain the words or match the core concept of the query.

Google results are a good example of this concept. When you search a word or a phrase in Google, the search engine find the best results based on your word/phrase, and then it ranks them (or sort them) based on a measure, then as a final result, it suggests those ranked items to you.

The ranking task in information retrieval is performed by using a ranking model $f(q, d)$ to sort the documents, where q denotes a query, and d denotes a document. In Language Model for IR (LMIR), $f(q, d)$ is represented as a conditional probability distribution $P(q|d)$. The probability models can be calculated with the words appearing in the query and document, and thus no training is needed. On the other hand, Learning to rank or machine-learned ranking (MLR) is the application of machine learning, typically supervised, semi-supervised or reinforcement learning, in the construction of ranking models for information retrieval systems.

Training data consists of lists of items with some partial order specified between items in each list. This order is typically induced by giving a numerical or ordinal score or a binary judgment (e.g., "relevant" or "not relevant") for each item. The ranking model's purpose is to rank, i.e., produce a permutation of items in new, unseen lists in a way that is "similar" to rankings in the training data in some sense.

III. Reinforced Recommender System

Given the explosive growth of information available on the web, users are often greeted with more than countless products, movies or restaurants. Thus, recommender systems are an intuitive line of defense against consumer over-choice. As such, personalization is an essential strategy for facilitating a grassroots user experience. These systems have been playing a vital role in various information access systems to boost business and facilitate the decision-making process and are pervasive across numerous web domains such as e-commerce and/or media websites. In general, recommendation lists are generated based on user preferences, item features, user-item past interactions, and other additional information. Such as temporal (e.g., sequence-aware recommender) and spatial (e.g., POI recommender) data. Recommendation models are mainly categorized into collaborative filtering, content-based recommender system and hybrid recommender system based on the types of input data.

Recommender systems are useful alternatives to search algorithms since it helps users discover items they might not have found otherwise. Of note, recommender systems are often implemented using search engines indexing non-traditional data[2]. Various techniques are proposed, such as content-based collaborative filtering, matrix factorization, logistic regression, factorization machines, neural networks, and MAB. However, most of the previous studies suffer from two limitations[3]:

1) Considering the recommendation as a static procedure and ignoring the dynamic interactive nature between users and the recommender systems, it is essential because in RL approaches, and we focused on the interaction. In a practical manner, we see that a user's behavior in one session of interaction with the recommender agent affects documents ranking in the next session. This can be formulated as an MDP and optimized using reinforcement learning. Recent works on reinforcement learning to rank, model the ranking process as an MDP, where, at each time-step, an agent selects one document given its current observation (ranking position and unranked documents.) Therefore it would be more reasonable to model the recommendation as a sequential decision-making process.
2) Focusing on the immediate feedback of recommended items and neglecting the long-term rewards. The aforementioned studies are trained by maximizing the immediate rewards of recommendations, which merely concentrates on whether the recommended items are clicked or consumed, but ignores the long-term contributions that the items can make. As we talked about RL, goal of the agent in this approach is to maximize the accumulative reward in a period of time. So it is obvious why this feature could affect our recommendation based on an RL approach.

As an instance, consider a user who requests for news to read, two possible pieces of news may lead to the same immediate reward, i.e., the user will click and read the two pieces of news with equal probability, where one is about a thunderstorm alert, and the other is about a basketball player Kobe Bryant. In this example, after reading the news about thunderstorms, the user probably is not willing to read news about this issue anymore; while on the other hand, the user will possibly read more about NBA or basketball after reading the news about Kobe. The fact suggests that recommending the news about Kobe will introduce more long term rewards. Hence, when recommending items to users, both the immediate and long-term rewards should be taken into consideration.

A. Problem Description

Because of the dynamic characteristics of the ranking problem, the way in which we can use RL could be different in one case to another. In this paper, by illustrating the mentioned idea, we want to explain how we can formulate a reinforced recommender system. Therefore, we found two basic approaches to use RL in a ranking problem. The first one is to see the problem as a multi-agent system like the "MarlRank" paper, which we do not want to talk about in this paper. The second approach is to see the whole problem as one agent system. In this approach, we look at "RL in E-Commerce", "Reinforcement learning approaches to movies recommendation" as instances to explain our ideas about differences and similarities between these approaches. For the last part of our paper, we want to trace our ideas about using one agent RL approach in ranking problems according to other papers.

For this purpose, we need to define the requirements of any RL problem, which are Reward, actions, and states. First of all, for defining Reward based on the reward hypothesis,

we should consider that the final duty of our agent is to rank our some documents (or in some cases like some commercial products) due to a specific goal. This part of the formulation -defining our final goal- is significant because, in the RL approach, this part leads us to determine the Reward accurately.

1) State: We talk about that our final goal that is to rank the top k documents considering user's rating by defining Reward and actions. As ranking documents matter for us, so the simple idea for defining the states refers to the interaction between the user (by query) and the documents. In a simple way, the first step of defining the state is one state situation, which we know as a multi-armed bandit problem.

In a multi-state reinforced recommender system, each state could be a situation in which the user is interacting with documents, and we already had ranked the documents. Thus, in this point of view, the definition of the states should be related to history (which we will discuss how the Markov property holds.) A simple way of formulating each state could be using a binary flag for each document. By considering that we have N documents at all, in each state, we have k documents with a score of +1 and $N - k$ documents with 0 score (which we know as a hot vector for documents.) It is trivial that by considering this way of defining the states, we have exactly 2^N, which is like a brute force which is not computationally efficient. But as we want to formulate the problem, generally it seems reasonable. In this kind of definition, not only should we recommend items but also we should arrange them with an order, therefore using a hot vector could not give us an ordered sequence of documents. Thus, we should consider any combination of the documents. This approach also produces $N!$ states.

Although the last approach provides a general formulation for us to rank the documents, considering all the traditional approaches for ranking, we should notice that this way of formulating the problem is not so efficient because it is like the brute-force approach. Instead, we can see each state historical based on the browsing history of each user. Notice that we want to formulate the ranking problem (not solve it!) based on RL approaches, so the definition of the state could be a little bit different based on the way in which we want to solve the problem. Thus, based on our final definition, state space is a set Like S, which contained a history of the items that the user browse. Therefore we formulate the history as a sequence like $s_h = \{i_1, i_2, ..., i_t\}$ in which i_t is the last item that user browse by time step t. this definition also is flexible based on the problem. For example, we can divide the states and label them into some groups.

For example, in [4], the definition of states depends on the page history. In this case, each item is divided into item pages which the user browse. The authors divided the whole state space into three parts. Two of them are the terminal states (one for buying products and one for leaving a search session) and the final group for those states which do not belong to the first two groups.

In another example like [5] the State space is a set like $s = \{i_1, ..., i_N\} \in S$ is defined as the previous N items that a user clicked/ordered recently. The items in state s are sorted in chronological order. As you can see, this formula is the same as we define with a difference of picking N items instead of all the history. So in this example, as we see a new item, we throw out the first item (because we should keep the number of browsing items to N).

2) Rewards: So in the first step of this formulation in the one agent RL based ranking problem, our final goal is to rank top documents based on the user's interests by considering the interaction between the user and the documents, Therefore, in almost all the cases our definition of Reward depends on the way in which user interacts with the documents using a specific numerical measure. Therefore We have a function like r. The domain of this function depends on the user's preferences. Based on our problem, this function could have other arguments, and the range of his function always is a numerical variable. Which by maximizing this numerical variable, we want to gain a goal. A practical example of defining Reward in a ranking problem is this: Assume that we have two business plans, one is to make money by selling commercial products and the other one by attracting clients to its website (assume that this latest one wants to make money by advertisement). So the best reward definition in the first approach is probably to consider a positive reward just for buying each product. However, in the second approach, it does not seem reasonable. Because in some cases, as the agent learned to recommend products to sell it, the agent may not consider that we want the client to stay on the page-whether the client bought anything or not. So it just recommends the product based on selling to achieve a reward. In this state, maybe the best reward definition would be considered a positive number for each time step that a client would spend time on this website.

- For the first case in Movie Recommendation, we see the problem as a MAB. In this case[6], our reward specified by user's behavior, which is to rate each movie. So the reward function is r_t, which is equal to the rating that at each time step user chose. In this case, using RL approach has the advantage of personalizing the recommendation list for every individual based on their rating to the movies. Therefore, in this case, this type of definition of the Reward seems reasonable.
- In another case like [4] , the user makes some operations (e.g., click items, buy some certain item or just request a new page of the same query on the page, and we specified a numerical reward for such an activity. In the online Learning To Rank (LTR) domain, user clicks are commonly adopted as a reward metric to guide learning algorithms. However, in E-commerce scenarios, successful transactions between users (who search items) and sellers (whose items are ranked by the search engine) are more important than user clicks. Thus, our reward setting is designed to encourage more successful transactions. So The agent will receive a positive reward from the environment only when its ranking action leads to a successful transaction. In all other cases, the reward

is zero. So the reward function is something like $r_{s'_{,a,h_t}}$ in which the h_t is the item page history at the time step t and a is the action and s' is the next state. notice that the expected deal price of any item page history is most probably unknown beforehand. In practice, the actual deal price of a transaction can be directly used as the reward signal.

3) Action: The definition of the actions depends on the reward functions. It is trivial that we want to maximize the cumulative reward in the long-term by taking appropriate actions. In case that we have the optimal policy, the result of our policy action selection should be picking up the final best k items to rank them as a result and exploit the current situation. On the other hand, in each step, our agent should take an action that can maximize the cumulative reward and with a probability like ϵ suggesting other documents to explore among all other possible documents (in $\epsilon greedy$ method).

- In the first case, the agent's action is to rank movies based on the estimation of the reward function to maximize future rewards[6]. Based on what we said in MAB section, this estimation is equal to an expectation that is resulted by averaging the cumulative rewards in all samples.
- In [4] example, as the user inputs a query in the blank of the search engine, the search engine ranks the items related to the query, and after that based on considering the reward it displays the top K items (e.g., K = 10) in a page. So the action is to displaying top K items based on maximizing the reward function after ending each search session. The key factor here is the search session. For example in a website like Amazon ranking items given a query is a multi-step decision-making problem. Therefore, the search engine should take a ranking action whenever a user requests an item page. A typical search session between the search engine and a mobile app user in Amazon. In the beginning, the user inputs a query "Cola" into the blank of the search engine and clicks the "Search" button. The search engine then takes a ranking action and shows the top items related to "Cola" on a page. The user browses the displayed items and clicks some of them for the details. When no items interest the user, or the user wants to check more items for comparisons, the user requests a new item page. The search engine again takes a ranking action and displays another page. After a certain number of such ranking rounds, the search session will finally end when the user purchases items or just leaves the search session. Thus we can easily see that the result of the reward function how effects on the way that search engine should act (or how to pick the actions to maximize the reward.)

4) Markov Property: The final step of defining the state space is to prove that this type of definition hold the Markov property. The Markov property means that a state is able to summarize past sensations compactly in such a way that all relevant information is retained. Formally, the Markov property refers to that for any state-action sequence $s_0, a_0, s_1, a_1, s_2, ..., s_{t1}, a_{t1}, s_t$ experienced an MDP, it holds that $Pr(s_t|s_0, a_0, s_1, a_1, ..., s_{t-1}, a_{t-1}) = Pr(s_t|s_{t-1}, a_{t-1})$. That is to say, the occurring of the current state st is only conditional on the last state-action pair (s_{t-1}, a_{t-1}) rather than the whole sequence.

B. Problem Formulation

Formulation of a reinforced recommender system is a tuple like $< R, A, S, P >$

- **State:** state space is a set like S in which for all $s_h \in S$, $s_h = i_1, i_2, ..., i_t$ is a state. for each time step like t, i_t is the last item that user browse by time step t. We usually divided the state space into different classes, like terminal states, or abandoned states an so on.
- **Reward:** is a function like $R(s, a_t, s') = \epsilon$ which relates the current state s into a transition state like s' based on an action like a using a numerical variable like ϵ. Value of ϵ based on reward hypothesis and our goal of recommendation system is positive in case that we satisfied the user's preferences (which the action taken due to this approach makes our state to be in a terminal one). For example in a case that user buy a product from our web-site. Usually in other cases we consider ϵ equal to zero.

$$R(s, a, s') \begin{cases} \epsilon > 0 & \text{if the user's interaction was successful} \\ \epsilon = 0 & \text{otherwise} \end{cases} \tag{1}$$

- **Action:** The action space is a set Like A in which for all $a_t \in A$, a_t is an action in time step t. a_t refers to recommending a set of k documents.
- **P:** State transition probability $p(s'|s, a)$ is defined as the probability that the state transits from s to s_0 when action a is executed. We assume that the state transition is deterministic: we always remove the first item i_1 from s and add the action a at the bottom of s, i.e., s_0 = $i_2, i_3, ..., i_t$.

IV. Related Works

In this section, we focus on three papers to find out similarities between our formulation and other works to evaluate our formulation.

In this[3] paper, the authors propose a novel recommendation framework based on deep reinforcement learning, called DRR. The DRR framework treats recommendation as a sequential decision making procedure and adopts an "Actor-Critic" reinforcement learning scheme to model the interactions between the users and recommender systems, which can consider both the dynamic adaptation and long-term rewards. In this paper, we aim to see how they formulate the problem, not how they solve it. So, based on the paper, the authors suggest a list of items sequentially over the time-steps, by maximizing the cumulative rewards of the whole recommendation procedure. More specifically, the recommended procedure is modeled by an MDP, as follows.

State: At time-step t, the state can be defined as follows: $s_t = f(H_t)$ where $f()$ stands for the state representation module, $H_t = \{i_1, ..., i_n\}$ denotes the embeddings of the latest positive interaction history, and item $i_t \in \mathbb{R}^{1\times k}$ is a k-dimensional vector. When the recommender agent recommends an item it, if the user provides positive feedback, then in the next time step, the state is updated to $s_{t+1} = f(H_{t+1})$, where $H_{t+1} = \{i_2, ...i_n\}$ otherwise, $H_{t+1} = H_t$. The reasons to define the state in such a manner are two folds: 1.a superior recommender system should cater to the users' taste, i.e., what items the users like; 2.the latest records represent the users' recent interests more precisely.

Action: An action a is a continuous parameter vector denoted as $a \in \mathbb{R}^{1\times k}$. Each item $i_t \in \mathbb{R}^{1\times k}$ has a ranking score, which is defined as the inner product of the action and the item embedding, i.e., $i_t.a^T$¿. Then the top-ranked ones will be recommended.

Reward: Given the recommendation based on the action a and the user state s, the user will provide her feedback, i.e., click, not click, or rating, etc.In timestep t, the recommender receives immediate reward $R(s; a)$ which is a function of $rate(i, j)$. The recommender agent recommends an item j to user i, (denoted as action a in state s), and the rating rate i;j comes from the interaction logs if user i actually rates item j, or from a predicted value by the simulator otherwise. According to the user's feedback.

For the second part, we peruse this [7] paper. The authors formulate the recommendation task as sequential interactions between a recommender system (agent) and users (environment E). They use the Markov Decision Process (MDP) to model it. It consists of a sequence of states, actions, and rewards. Typically, MDP involves four elements (S, A, P, R), and below we introduce how they set them:

State: We define the state $s = \{i_1, ..., i_N\}$ S as a sequence of N items that a user browsed and user's corresponding feedback for each item. The items in s are chronologically sorted.

Action: An action $a \in A$ from the recommender system perspective is defined as recommending a set of items to a user. Without loss of generality, we suppose that each time the recommender system suggests one item to the user, but it is straightforward to extend this setting to recommending more items.

Reward: When the system takes an action a based on the state s, the user will browse the recommended item and provide her feedback on the item. In this paper, we assume a user could skip, click, and purchase the recommended items. Then the recommender system will receive a reward $r(s; a)$ solely according to the type of feedback.

For the last one, in [4] In the first part of this paper, the authors propose to use reinforcement learning (RL) for ranking control in E-commerce searching scenarios. Their contributions are as follows. They formally define the concept of the search session Markov decision process (SSMDP) to formulate the multi-step ranking problem in E-commerce searching scenarios. a search session between a user and the search engine is a multi-step ranking problem as follows:

1-The user inputs a query in the blank of the search engine

2-The search engine ranks the items related to the query and displays the top K items (e.g., K = 10) in a page

3-The user makes some operations (e.g., click items, buy some specific item or just request a new page of the same query) on the page

4-When a new page is requested, the search engine re-ranks the rest of the items and display the top K items.

These four steps will repeat until the user buys some items or just leaves the search session.

State: For defining states, instead of using histories of item sets, authors work with item pages. Each item page is a set of items. The states of the environment are an indication of user status in the corresponding item page histories. It consists of three different categories: continuation, abandonment, or transaction. For any item page history h_t in a search session, let $B(h_t)$ denote the conversion event that a user purchases an item after observing h_t. For any item page history h_t in a search session, we are in an abandon state when a user leaves the search session after observing h_t. For any item page history h_t in a search session, let $C(h_t)$ denote the continuation event that a user continues searching after observing h_t. This type of definition helps the author to prove Markov property and define transition functions easier. More important, it has an advantage to defining reward function.

Action: In this scenario, The action space A can be set differently (e.g., discrete or continuous) according to specific ranking tasks. Action space contains all possible ranking functions of the search engine.

Reward: In a search session MDPM, the reward function$R$ is a quantitative evaluation of the action performed in each state. Specifically, for any non-terminal state like s, any action like a, and any other state like s, $R(s, a, s)$ is the expected value of the immediate rewards that numerically characterize the user feedback when action a is taken in s, and the state is changed to s. Lastly we should translate this feedback to numerical value for the learning algorithm. Then for the non-terminal state $C(h_t)$ and any state s, the reward function is set as follows:

$$R(C(h_t), a, s') \begin{cases} m(h_{t+1}) & \text{if } s' = B(h_{t+1}) \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

where $B(h_{t+1})$ is the terminal state which represents the conversion event of h_{t+1}.The agent will recieve a positive reward from the environment only when its ranking action leads to a successful transaction. In all other cases, the reward is zero.

V. Conclusion

For a single agent RL based ranking (known as reinforced recommender system model), then introduce a way of formulating the problem by defining the requirements of an RL problem, which are the rewards, actions, and states. For reaching this goal, first, we describe how we can see the ranking problem using RL approaches and then by using other featured works on this topic. In the last part, in order

to evaluate our formulation. Then, We illustrate that this formulation had been used in some other important papers.

REFERENCES

[1] Sutton,R. S., Barto, A.G.(2018) "Reinforcement Learning: An Introduction" MIT Press, Cambridge, MA

[2] Francesco Ricci and Lior Rokach and Bracha Shapira (2011), "Introduction to Recommender Systems Handbook" Recommender Systems Handbook, Springer

[3] Feng Liu, Ruiming Tangy, Xutao Li, Weinan Zhangz Yunming Ye, Haokun Chenz, Huifeng Guoy and Yuzhou Zhangy. "Deep Reinforcement Learning based Recommendation with Explicit User-Item Interactions Modeling" Shenzhen Graduate School, Harbin Institute of Technology, Shenzhen, 518055, China

[4] Yujing Hu, Qing Da, Anxiang Zeng, Yang Yu, Yinghui Xu (2018) "Reinforcement Learning to Rank in E-Commerce Search Engine: Formalization, Analysis, and Application" Alibaba Group, National Key Laboratory for Novel Software Technology, Artificial Intelligence Department

[5] Xiangyu Zhao, Liang Zhang, Zhuoye Ding, Long Xia, Jiliang Tang, Dawei Yin. "Recommendations with Negative Feedback via Pairwise Deep Reinforcement Learning" Michigan State University, Data Science Lab JD.com

[6] Antoine Carpentier, Pierre G'erard and Julian Schembri. "Reinforcement learning approaches to movies recommendation" Vrije Universiteit Brussel, Universit' e libre de Bruxelles

[7] Xiangyu Zhao, Long Xia, Lixin Zou, Dawei Yin, Jiliang Tang. "Simulating User Feedback for Reinforcement Learning Based Recommendations" Michigan State University, Tsinghua University, Data Science Lab JD.com