Reinforcement Learning based Recommender Systems: A Survey

M. MEHDI AFSAR, TRAFFORD CRUMP, and BEHROUZ FAR, University of Calgary, Canada

Recommender systems (RSs) have become an inseparable part of our everyday lives. They help us find our favorite items to purchase, our friends on social networks, and our favorite movies to watch. Traditionally, the recommendation problem was considered to be a classification or prediction problem, but it is now widely agreed that formulating it as a sequential decision problem can better reflect the user-system interaction. Therefore, it can be formulated as a **Markov decision process (MDP)** and be solved by **reinforcement learning (RL)** algorithms. Unlike traditional recommendation methods, including collaborative filtering and content-based filtering, RL is able to handle the sequential, dynamic user-system interaction and to take into account the long-term user engagement. Although the idea of using RL for recommendation is not new and has been around for about two decades, it was not very practical, mainly because of scalability problems of traditional RL algorithms. However, a new trend has emerged in the field since the introduction of **deep reinforcement learning (DRL)**, which made it possible to apply RL to the recommendation problem with large state and action spaces. In this paper, a survey on **reinforcement learning based recommender systems (RLRSs)** is presented. Our aim is to present an outlook on the field and to provide the reader with a fairly complete knowledge of key concepts of the field. We first recognize and illustrate that RLRSs can be generally classified into RL- and DRL-based methods. Then, we propose an RLRS framework with four components, i.e., state representation, policy optimization, reward formulation, and environment building, and survey RLRS algorithms accordingly. We highlight emerging topics and depict important trends using various graphs and tables. Finally, we discuss important aspects and challenges that can be addressed in the future.

CCS Concepts: • **Information systems** → **Recommender systems**

Additional Key Words and Phrases: Recommender systems, reinforcement learning

ACM Reference format:
M. Mehdi Afsar, Trafford Crump, and Behrouz Far. 2022. Reinforcement Learning based Recommender Systems: A Survey. *ACM Comput. Surv.* 55, 7, Article 145 (December 2022), 38 pages.
https://doi.org/10.1145/3543846

1 INTRODUCTION

We are living in the *Zettabyte Era* [1]. The massive volume of information available on the web leads to the problem of *information overload*, which makes it difficult for a decision maker to make right decisions. The realization of this in our everyday lives is when we face a long list of items in an online shopping store; the more items in the list, the tougher it becomes to select among them.

Author's address: M. M. Afsar, T. Crump, and B. Far, University of Calgary, 2500 University Dr NW, Calgary, AB, Canada; emails: {mehdi.afsar, tcrump, far}@ucalgary.ca.

0360-0300/2022/12-ART145 $15.00
https://doi.org/10.1145/3543846

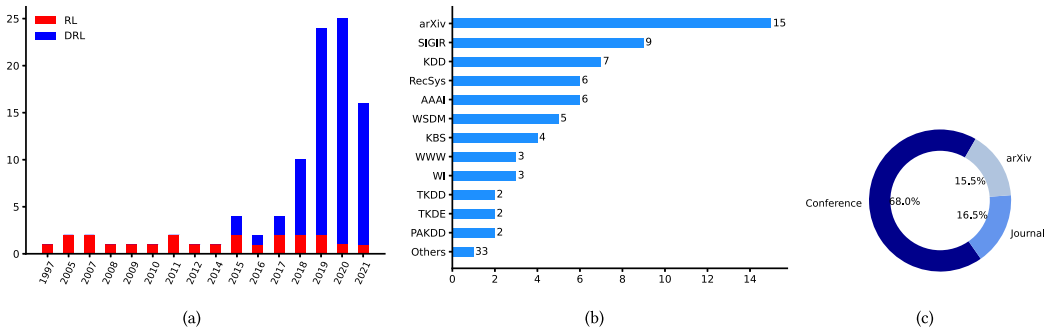


Fig. 1. Publications information of 97 surveyed RLRS papers. (a) Distribution of RLRSs publications per year (until September 2021) separated based on RL and DRL methods. (b) Venue distribution of published RLRSs. To be clear, we filtered venues with only one publication and termed them as 'Others' in the graph. This includes a long list of venues, including AAMAS, ICML, ICDM, and JMLR. (c) The proportion of different types of publications, i.e., conference proceedings, journal articles, and arXiv preprints, of surveyed RLRSs.

Recommender systems (RSs) are software tools and algorithms that have been developed with the idea of helping users find their items of interest, through predicting their preferences or ratings on items [2, 3]. In fact, the idea is to know the users to some extent, i.e., making a *user profile* based on their feedback on items, and to recommending those items that match their profile. Today, RSs are an essential part of most giant companies, like Google, Facebook, Amazon, and Netflix, and employed in a wide range of applications, including entertainment [4–6], e-commerce [7], news [8], e-learning [9], and healthcare [10].

Numerous techniques have been proposed to tackle the recommendation problem; traditional techniques include collaborative filtering, content-based filtering, and hybrid methods. Despite some success in providing relevant recommendations, specifically after the introduction of *matrix factorization* [11], these methods have severe problems, such as *cold start* (i.e., the system cannot provide useful recommendation when the user or item is new), lack of novelty and diversity, scalability, low quality recommendation, and great computational expense [2, 3, 12]. Recently, deep learning [13] has also gained popularity in the RS field due to its ability in finding complex and non-linear relationships between users and items and its cutting edge performance in recommendation [14]. Nonetheless, deep learning models are usually non-interpretable, data hungry (this is specifically problematic as the amount of data, i.e. rating/user feedback, in the RS field is scarce), and computationally expensive [14].

RL is a semi-supervised machine learning field in which the agent optimizes its behavior through interaction with the environment. The milestone in the RL field is the combination of deep learning with traditional RL methods, which is known as **deep reinforcement learning (DRL)** [15, 16]. This made it possible to apply RL in problems with enormous state and action spaces, including self-driving cars [17, 18], robotics [19], industry automation [20], finance [21], healthcare [22, 23], and RSs [24]. The unique ability of an RL agent in learning from a reward from the environment without any training data makes RL specifically a perfect match for the recommendation problem. Today, more and more companies are utilizing the power of RL to recommend better items to their customers. For example, in a study by researchers at Google [25], it is shown that RL can be employed to recommend better video content to YouTube's users. In fact, the use of RL in the RS community is not limited to the industry, but it is becoming a trend in academia as well. Figure 1(a) illustrates this trend.

This trend and this topic motivated us to prepare this survey paper, which aims at providing a comprehensive overview of the state-of-the-art in **reinforcement learning based**

recommender systems (RLRSs). Our main purpose is to depict a high-level picture from the progress in the field since the beginning and show how this trend has been significantly changed with the advent of DRL. At the same time, we provide detailed information about each method in the form of tables so as the reader can easily observe the similarities and differences between methods.

Paper Collection Methodology. To collect relevant papers, we have used a multi-level search process. The focus of this survey paper is specifically on RSs that use an RL algorithm. Accordingly, in order to find relevant papers, we used Google Scholar as the main search engine and searched "reinforcement learning recommender system" keyword. This search resulted in around 33,000 papers. Out of the first 1,000 articles found, we collected 500 papers as the result of our first screening level. Then, to increase the reliability of our paper collection, we also explored related libraries like ACM digital library, IEEE Xplore, SpringerLink, and ScienceDirect with the same keyword and until a point where no more relevant papers were among the results. We found that all related articles identified in these libraries were available in our initial search using Google Scholar. With carefully studying collected articles and excluding irrelevant papers, duplicates, theses, survey and review papers, we selected 97 articles to include in our survey paper. Although we are certain that we did not find all RLRSs through the search process explained, we are confident that we could find the vast majority of relevant publications.

It is noteworthy to mention that we did not include RSs based on *multi-armed bandits*. Bandits are a simplified version of RL. In particular, in bandits, similar to an RL problem, the agent should learn to maximize a numerical reward through interaction with the environment and solving the *exploration vs exploitation dilemma* [26, 27]. However in bandits, different from *full* RL, actions are not permitted to affect the state of the environment and the reward [27]. While bandits have been popular for the recommendation problem [28–31], in light of recent successful applications of DRL and unprecedented interests in full RL by the RS community, we opt to only focus on RSs that use a full RL algorithm. The curious reader is referred to a recent survey on the application of bandits to RSs [32].

Related Work. Plenty of research has been done in the field of RSs and a plethora of survey papers have been published, including RSs [12], collaborative filtering [33, 34], hybrid methods [35], multi-media RSs [36, 37], explainable recommendation [38], and article RSs [39], to name a few. There are also some published survey papers on topics closely related to RLRSs [14, 40–42]. Perhaps the two closest survey papers to ours are [14, 40]. Reference [40] surveys DRL-based information seeking techniques, such as search, recommendation, and online advertising. Authors discuss several RSs, which use multi-armed bandits and DRL for policy optimization. However, the work misses many important RLRSs and fails to provide an in-depth analysis of algorithms reviewed. Zhang et al. [14] provide a comprehensive survey on deep learning based RSs. They consider DRL as a deep learning architectural paradigm and survey a few DRL-based RSs [43–48]. Nonetheless, this classification is not correct and DRL is not a deep learning architecture, but it is an extension to traditional RL algorithms. Other related surveys target sequence-aware [41] and session-based [42] recommendation techniques. Reference [41] surveys sequence-aware RSs. Authors consider RL as a method for sequence learning and review some RL-based RSs [49, 50]. In another related survey [42], RL is considered as a method for session-based RSs [46, 51, 52]. None of previous published survey papers provide a comprehensive overview and in-depth analysis of published RLRSs. To the best of our knowledge, this is the first survey paper that specifically targets RLRSs.

Our contribution. The goal is to provide the reader with a vista toward the field so that they can quickly understand the topic and major trends and algorithms presented so far. This helps

researchers see the big picture, compare algorithms' strengths and weaknesses, and shed some light on ways to advance them in the future. Our main contributions can be summarized as:

- Presenting a framework for RLRSs. We first generally divide RLRSs into RL- and DRL-based methods. Then, we propose a framework with four main components, i.e., state representation, policy optimization, reward formulation, and environment building. This framework can model every RLRS and unify the development process of RLRSs.
- Providing a thorough background on RL. We provide the reader with a fairly complete knowledge on RL/DRL and their various algorithms used by RLRSs.
- Highlighting important trends and emerging topics. Instead of simply summarizing algorithms, our aim is to extract and to illustrate major trends and attempts in each main component of the proposed framework in particular and emerging topics in RLRSs in general.
- Suggesting some open research directions for the future. In order to consolidate our survey paper, we finally present some observations about ongoing research in the RLRS field and propose some open research directions to advance the field.

The remaining of this paper is organized as follows. In Section 2, to help the reader better understand the topic, we discuss some preliminary concepts and provide a solid background on RL. Section 3 presents RLRSs algorithms in a classified manner. Emerging topics are highlighted in section 4. In Section 5, some open research directions are suggested for the future work, and finally, the paper is concluded in Section 6.

2 PRELIMINARIES

In this section, we provide a background on the important concepts discussed throughout this paper. This background provides the reader with useful and concise information about RSs, RL and DRL, why it is necessary to use RL in RSs, problem formulation, and the proposed framework for RLRSs. Table 1 provides the definition of abbreviations used throughout this article.

2.1 Recommender Systems

In everyday life, it is not very uncommon to face situations in which we have to make decisions while we have no *a priori* information about options. In such a case, relying on recommendations from others, who are experienced in that aspect, seems quite necessary [53]. This was the rationale behind the first RS, Tapestry [54], and authors termed it as *collaborative filtering*. Later, this term was broadened to *recommender systems* to reflect two facts [53]: (1) the method may not be based on an implicit collaboration between users, and (2) the method may suggest interesting items, not filter them. By definition, RSs are software tools and algorithms that suggest items that might be of interest to the users [3]. Another important approach toward the recommendation problem is *content-based filtering*, in which the idea is to use item descriptions and devising a method to match them with the user *profile*, a structured representation of user interests [55, 56]. Collaborative filtering usually suffers from data sparsity, scalability, and *gray sheep* (the users with special tastes whose opinions do not agree or disagree with the majority of the users) [33]. Content-based filtering also has some shortcomings, including limited content analysis, serendipity, and new user [56]. Hybrid methods, a combination of the two, can only alleviate part of these problems [3, 33].

2.2 From Reinforcement Learning to Deep Reinforcement Learning

Reinforcement learning (RL) is a machine learning field that studies problems and their solutions in which agents, through interaction with their environment, learn to maximize a numerical reward. According to Sutton and Barto [27], three characteristics distinguish an RL problem: (1) the problem is closed-loop, (2) the learner does not have a tutor to teach it what to do, but it should

Table 1. Abbreviation Definition

Abbreviation	Definition	Abbreviation	Definition
AAR	Average achievable rate	MIMIC	Multiparameter intelligent monitoring in intensive care
ACN	Average click number per capita	MJC	Mean Jaccard coeffcient
ACS	Average click per session	ML	ML MovieLens
ADS	Average depth per session	MOOCs	Massive open online courses
AQ	Average quality	MR	Miss ratio
AR	Average reward	MRR	MRR Mean reciprocal rank
ART	Average return time	MT	Movie tweetings
AT	Average turn	NDCG	Normalized discounted cumulative gain
AWT	Average watched tag per capita	NI	Number of user interactions before success
BC	BookCrossing	NV	Number of violated attributes
BDV	Book details viewed	OU	Ornstein-Uhlenbeck
BP	Number of books purchased	P	Precision
CAC	Combined accuracy and coverage	PA	Predictive ability
CATIE	Clinical antipsychotic trials of intervention effectiveness	PANSS	Positive and negative syndrome scale
CFR	Charging failure rate	PI	Popularity rate
CMU	Carnegie Mellon university	PO	Policy optimization
CNN	Convolutional neural networks	POI	Point-of-interest
CR	Click reduction	POMDP	Partially observable Markov decision process
CTR	Click though rate	PPO	Proximal policy optimization
CVR	Conversion rate	PP30	Passenger pickup in 30 minutes
DDPG	Deep deterministic gradient descent	PR	Popularity rate
DDQN	Double DQN	QE	Queries executed
DHMM	Discriminative hidden Markov model	R	Recall
DQN	Deep Q network	RA	Recommendation acceptance
DRL	Deep reinforcement learning	RC	Recommendation score
Earn	Average earning	RDR	Recommended download rate
EB	Environment building	RF	Reward formulation
ED	Exponential decay score	RL	Reinforcement learning
EMR	Entity matching rate	RLRS	Reinforcement learning based recommender system
ES	Energy saving	RMSE	Root-mean-square error
FC	Fully connected	RP	Ranking percentile
FQI	Fitted Q iteration	RQ	Recommendation quality
GANs	Generative adversarial networks	RS	Recommender system
GI	Gini index	SAD	Session ad revenue
GRU	Gated recurrent units	SDT	Session dwell time
GVI	Gradient value iteration	SL	Session length
HR	Hit ratio	SG	Shortcut gains
HRL	Hierarchical reinforcement learning	SR	State representation
HSP	Historical song playlist	ST	Session time
ILS	Intra-list similarity	SuR	Success rate
k-NN	k-nearest neighbors	TSF	Total saving fee
LFM	Last.fm	TPS	Taste profile subset
LRR	Low rank rate	TWIS	Truncated weighted importance sampling
LSTM	Long short-term memory	UR	User rating
LTR	Listening time ratio	UV	CTR Click ratio to user view
LTV	Life-time value	VCT	Vacant cruising time
MARL	Multi-agent reinforcement learning	WI	Wage improvement
MAP	Mean Average precision	WPF	Weighted proportional fairness
MCP	Mean charging price	WQR	Wrong quite rate
MCTS	Monte Carlo tree search	WT	Waiting time
MCWT	Mean charging wait time	YM	Yahoo music
MDP	Markov decision process	YC	YooChoose
MH	Miss-to-hit		

figure out what to do through trial-and-error, and (3) actions influence not only the short-term results, but also the long-term ones. The most common interface to model an RL problem is the *agent-environment* interface, depicted in Figure 2(a). The learner or decision maker is called *agent* and the *environment* is everything outside the agent. Accordingly, at time step t, the agent sees some representations/information about the environment, called *state*, and based on the current state it takes an *action*. On taking this action, it receives a numerical *reward* from the environment and finds itself in a new state.

More formally, the RL problem is typically formulated as a **Markov decision process (MDP)** in the form of a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P}, \gamma)$, where $\mathcal{S}$ is the set of all possible states, $\mathcal{A}$ is the set of available actions in all states, $\mathcal{R}$ is the reward function, $\mathcal{P}$ is the transition probability, and γ is the discount factor.

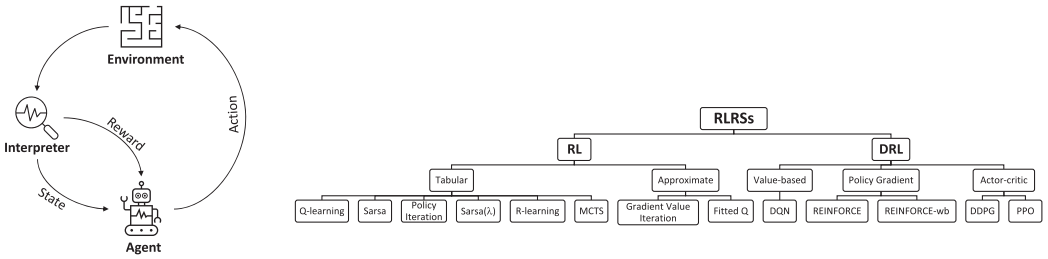


Fig. 2. (a) Agent-environment RL interface, (b) RL algorithms used by RLRSs.

The main elements of an RL system are [27]:

- *Policy:* policy is usually indicated by π and gives the probability of taking action a when the agent is in state s. Regarding the policy, RL algorithms can be generally divided into *on-policy* and *off-policy* methods. In the former, RL methods aim at evaluating or improving the policy they are using to make decisions. In the latter, they improve or evaluate a policy that is different from the one used to generate the data.
- *Reward signal:* upon selecting actions, the environment provides a numerical reward to inform the agent how good or bad are the actions selected.
- *Value function:* the reward signal is merely able to tell what is good immediately, but the value function defines what is good in the long run.
- *Model:* model provides the opportunity to make inferences about the behavior of the environment. For instance, the model can predict next state and next reward in a given state and action [27].

Algorithms. Many algorithms have been proposed to solve an RL problem; they can be generally divided into *tabular* and *approximate* methods [27]. In tabular methods, since the size of action and state spaces is small, value functions can be represented as tables and optimal value function and policy can be found. On the other hand, in approximate methods, since the size of state space is enormous, the goal is to find a good approximate solution with the constraint of limited computational resources. As mentioned earlier, with the foundation of DRL, a substantial change has emerged in the RL field in general. Accordingly, although DRL belongs to the approximate group, we generally divide RL algorithms used by RLRSs into RL-based and DRL-based algorithms as we believe this classification better reflects the recent trend in the RLRS field. It is noteworthy to mention that the distinguishing factor between DRL and traditional RL algorithms is that DRL algorithms use deep learning for function approximation (a more detailed explanation on this is presented shortly in DRL-based Algorithms section). In the following, we briefly review those RL algorithms employed by RLRSs. Figure 2(b) illustrates these algorithms.

(1) RL-based Algorithms. As stated before, RL algorithms could be divided into tabular and approximate methods. Popular tabular methods include *dynamic programming*, *Monte Carlo*, and *temporal difference*. **Dynamic programming** methods assume a perfect model of the environment and use a value function to search for good policies. Two important algorithms from this class are *policy iteration* and *value iteration*. **Policy iteration** algorithm composes of three steps: initialization, policy evaluation, and policy improvement. First the policy is randomly initialized, i.e., a random action $a \in A(s)$ is selected for all $s \in \mathcal{S}$. Then, the value of the states are computed and evaluated using

$$V(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s))[r + \gamma V(s')], \tag{1}$$

where p is the transition probability and s' is the next state. Finally, the policy is updated as follows, $\forall s \in \mathcal{S}$:

$$\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]. \tag{2}$$

As pointed out in [27], a problem with policy iteration algorithm is that it needs policy evaluation in every iteration, which can be computationally prohibitive. **Value iteration** algorithm is a special case of policy iteration algorithm in which policy evaluation is stopped after one *sweep*. More precisely, $V(s)$ is randomly initialized $\forall s \in \mathcal{S}$. Then, it is updated in each step according to

$$V(s) \leftarrow \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]. \tag{3}$$

Temporal difference methods are a combination of dynamic programming and Monte Carlo methods. While they do not need a model from the environment, they can *bootstrap*, which is the ability to update estimates based on other estimates [27]. From this family, Q-learning [57] and Sarsa [58] are very popular. **Q-learning** is a model-free, off-policy algorithm to learn the value of an action in a given state. The main component of Q-learning is the following *Bellman* equation:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\Big[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\Big], \tag{4}$$

where α is the learning rate. **Sarsa** is an online version of Q-learning with the following update rule:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]. \tag{5}$$

Sarsa(λ) employs *eligibility traces*, which unify temporal difference and Monte Carlo methods [27]. In Sarsa(λ), the weight vector is updated on each step using

$$w_{t+1} = w_t + \alpha\delta_t e_t, \tag{6}$$

where δ_t is temporal difference error and defined by

$$\delta_t = r_{t+1} + \gamma q(s_{t+1}, a_{t+1}, w_t) - q(s_t, a_t, w_t), \tag{7}$$

where q is the approximate action value. Also, the eligibility trace e_t is defined as

$$e_t = \gamma\lambda e_{t-1} + \nabla q(s_t, a_t, w_t), \quad 0 \le t \le T, e_{-1} = 0 \tag{8}$$

where $\lambda \in [0, 1]$. More details on Sarsa(λ) can be found in [27]. **R-learning** [59] is an extension of Q-learning for continuing tasks where the interaction between agent and environment continues forever without termination. The main idea in R-learning is the concept of *average reward*; that means, discounted reward setting is not necessary, even problematic, in continuous function approximation and there is no difference between immediate and delayed rewards [27]. That said, Q-learning equation, Equation (4), could be rewritten as [49]

$$Q(s, a) \leftarrow Q(s, a) + \alpha\Big[r - \rho + \max_{a'} Q(s', a') - Q(s, a)\Big], \tag{9}$$

where ρ is the average reward and can be achieved by

$$\rho \leftarrow \rho + \beta\Big[r - \rho + \max_{a'} Q(s, a') - \max_a Q(s, a)\Big], \tag{10}$$

where β is a learning rate.

There are also methods that combine or unify model-based RL (like Dynamic Programming) with model-free RL (e.g., Monte Carlo or temporal difference) [27]. An enhanced version of **Monte Carlo tree search (MCTS)** could be categorized in this family. Although MCTS is a search method in base, it is usually enhanced by a method to accumulate value estimates through Monte Carlo simulations [27]. The basic MCTS is an iterative search tree building algorithm that runs until

reaching a predefined computational budget [60]. It composes of four steps: selection, expansion, simulation, and backpropagation [61]. Starting at the root node, a recursive child selection policy is executed to select the leaf node (*expandable* node). The tree is then expanded by adding one or more child nodes. Next, a simulation of a complete episode is performed from newly added leaf nodes to produce an outcome. Finally, the result of simulation is backpropagated in the tree (through selected nodes). Later in 2016, MCTS was merged with deep learning [62].

In **approximate methods**, a practical approach is to *generalize* from previous experiences (already seen states) to unseen states. *Function approximation* is the type of generalization required in RL and many techniques could be used to approximate the function, including *artificial neural networks*. **Fitted Q** algorithm is an approximate method that is inspired by the idea of *fitted value iteration* proposed by Gordon [63]. An interesting fact about fitted Q framework is that it allows to use any regression algorithm for function approximation. For instance, Ernst et al. [64] use randomized trees for function approximation and call their method **fitted Q iteration (FQI)**. Among approximate solutions, **policy gradient** methods have been very popular, which learn a *parameterized* policy and can select actions without the need of a value function. **REINFORCE** [65] is a Monte Carlo method that uses episode samples in order to update the policy parameter θ. It first randomly initializes θ. Then, it iteratively generates a trajectory following policy $\pi_\theta : S_1, A_1, R_1, \ldots, S_T$. For each step $t = 1, 2, \ldots, T$, it estimates return G_t and updates θ using

$$\theta \leftarrow \theta + \alpha\gamma^t G_t \nabla_\theta \log \pi(A_t|S_t, \theta). \tag{11}$$

A major problem with REINFORCE is that it has high variance in gradient estimation. To overcome this problem, a baseline (state-value function) is added to the update rule:

$$\theta \leftarrow \theta + \alpha\gamma^t (G_t - b(S_t))\nabla_\theta \log \pi(A_t|S_t, \theta). \tag{12}$$

This method is called **REINFORCE-with-baseline** (**REINFORCE-wb**) [27]. While REINFORCE-wb reduces the variance, it is still a Monte Carlo method and has a slow convergence. A better solution is to add bootstrapping to REINFORCE; an idea employed by **actor-critic** method [66]. In particular, instead of a baseline, a critic is used to criticize the policy generated by the actor. That said, the update rule for actor-critic is revised as follows

$$\theta \leftarrow \theta + \alpha\gamma^t (G_t - v(S_t, w))\nabla_\theta \log \pi(A_t|S_t, \theta), \tag{13}$$

where $v(s, w)$ is a state-value function parameterized with w.

(2) DRL-based Algorithms. DRL is an interesting combination of deep learning with RL. In fact, researchers at DeepMind found that this combination can achieve human-level performance in Atari games [67, 68]. **Deep Q-network** (**DQN**) [67] is a creative combination of **convolutional neural networks (CNN)** [13] with Q-learning. More precisely, in DQN, a *Q network* is responsible for action-value approximation, which could be trained to minimize the following loss function:

$$L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(\cdot)}[(y_i - Q(s, a; \theta_i))^2], \tag{14}$$

where $y_i = \mathbb{E}_{s'}[r + \gamma \max_{a'} Q(s', a'; \theta_{i-1})|s, a]$ is the target for iteration i and ρ is a probability distribution over transitions s, a, r, s' collected from the environment. Differentiating $L(\theta)$ in Equation (14) with respect to θ yields the following gradient

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(\cdot)}\Big[\Big(r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) - Q(s, a; \theta_i)\Big)\nabla_{\theta_i} Q(s, a; \theta_i)\Big]. \tag{15}$$

It is computationally beneficial to optimize the gradient in Equation (15) using *stochastic gradient descent* [68]. According to [27], DQN modifies the original Q-learning algorithm in three ways: (1) It uses *experience replay*, first proposed in [69] and a method that keeps agents' experiences over various time steps in a replay memory and uses them to update weights in the training phase.

(2) In order to reduce the complexity in updating weights, current updated weights are kept fixed and fed into a second (duplicate) network whose outputs are used as Q-learning targets. 3) The error term $(r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) - Q(s, a; \theta_i)$ in Equation (15) is clipped such that it remains in the interval [−1, 1]. All these modifications help improve the stability of DQN.

However, DQN has some problems: First, following Q-learning algorithm, DQN overestimates action values under certain circumstances, which makes learning inefficient and can lead to sub-optimal policies [70]. **Double DQN** (**DDQN**) was proposed to alleviate this problem [71]. The difference between DQN and DDQN is that the greedy policy is evaluated using online network, but the target network is used to estimate its value. Thus, y_i is changed as follows

$$y_i = r + \gamma Q\Big(s', \arg\max_{a'} Q(s', a'; \theta_i); \theta_{i-1}\Big). \tag{16}$$

An interesting extension on top of DDQN is **dueling network** [72] whose idea is to have a single Q network with the same *convolutional layers* as DQN, but to have two streams of ***fully connected*** **(FC)** layers, which provide estimates for value and *advantage* functions. This helps better generalize learning between actions. Second, DQN uniformly selects experiences to replay regardless of their significance, which makes the learning process slow and inefficient. Accordingly, *prioritized experience replay* was proposed to solve the problem [73]. The idea is to replay important experiences more often, so as the network training is improved. The importance of each transition is measured proportional to temporal difference error, and two variants, *stochastic prioritization* and *importance sampling*, are proposed to improve it. Finally, DQN is not applicable in continuous spaces, so **deep deterministic policy gradient** (**DDPG**) [74] was proposed, which is a combination of DQN and **deterministic policy gradient (DPG)** [75] in an actor-critic approach. Actor deterministically maps states to a specific action. Critic defines the value of the action taken by actor. In every iteration, critic is updated by

$$L = \frac{1}{N} \sum_i \left(y_i - Q\left(s_i, a_i | \theta^Q\right)\right)^2 \tag{17}$$

and actor is updated by

$$\nabla_{\theta^\mu} J = \frac{1}{N} \sum_i \nabla_a Q(s, a | \theta^Q) | (s = s_i, a = \mu(s_i)) \nabla_{\theta^\mu} \mu(s | \theta^\mu) | s_i, \tag{18}$$

where θ^μ and θ^Q are the parameters of actor and critic networks, respectively.

Finally, **proximal policy optimization** (**PPO**) [76] is another actor-critic algorithm used by RLRSs. In fact, PPO is an improved version of **trust region policy optimization (TRPO)** [77] algorithm, which maximizes a *surrogate* objective

$$\mathbb{E}_t \left[\frac{\pi_\theta(a_t | s_t)}{\pi_{\theta_{\text{old}}}(a_t | s_t)} A_t \right], \tag{19}$$

where A_t is an estimator of advantage function at t. The core idea in PPO is the introduction of *clipped surrogate objective*, as

$$\mathbb{E}_t \left[\min \left(\frac{\pi_\theta(a_t | s_t)}{\pi_{\theta_{\text{old}}}(a_t | s_t)} A_t, \text{clip} \left(\frac{\pi_\theta(a_t | s_t)}{\pi_{\theta_{\text{old}}}(a_t | s_t)}, 1 - \epsilon, 1 + \epsilon \right) A_t \right) \right], \tag{20}$$

where ϵ is a hyperparameter.

RL Challenges. There are some possible challenges when applying RL to any problem. A challenge well-known as *Deadly Triad* states that there is a *hazard* of instability and divergence when combining three elements in RL: function approximation, bootstrapping, and off-policy

training [27]. Another challenge in RL is *sample inefficiency*, specifically in model-free RL algorithms [78]. Current model-free RL algorithms need a considerable amount of agent-environment interaction in order to learn useful states. Moreover, since DRL is based on deep learning, it consequently inherits the famous feature of neural networks, i.e., being *black-box*. It is not obvious how weights and activations are changed, which makes them uninterpretable. The classical problem of exploration vs exploitation is still a challenge in RL and effective exploration is an open research problem. Finally, the problem of reward formulation in RL is a challenge and designing a good reward function is not very clear or straightforward.

2.3 Why Reinforcement Learning for Recommendation?

The nature of user interaction with an RS is sequential [79] and the problem of recommending the best items to a user is not only a prediction problem, but a sequential decision problem [50]. This suggests that the recommendation problem could be modelled as an MDP and be solved by RL algorithms. Three unique features of RL make it a perfect match for the recommendation problem. First, RL is able to handle the *dynamics* of sequential user-system interaction by adjusting actions according to continuous feedback received from the environment. Second, RL is able to take into account the long-term user engagement with the system. Finally, although having user ratings is beneficial, RL, by nature, does not need user ratings and optimizes its policy by sequentially interacting with the environment. All these reasons suggest that it would be beneficial to use RL to provide better recommendations, as proven by online studies [25, 80].

2.4 Problem Formulation

In a recommendation problem, the RS algorithm, through interaction with the user and receiving their implicit/explicit feedback, tries to recommend the best items to the user, in order to achieve the goal it is designed for, which could be increasing profit, user satisfaction, or user fidelity [3]. This is analogous to a typical RL setting, where an agent aims at maximizing a numerical reward through interaction with an environment [27]. Therefore, the RL agent can play the role of the RS algorithm and every thing outside this agent, including the users of the system and items, can be considered as the environment for this agent.

More formally, considering the user and items as the environment and the RS algorithm as the RL agent, MDP formulation can be as follows:

- **State $\mathcal{S}$:** a state $s_t \in \mathcal{S}$ is defined as the user preferences and their past history with the system.
- **Action $\mathcal{A}$:** an action $a_t \in \mathcal{A}$ is to recommend an item to the user at time step t.
- **Reward $\mathcal{R}$:** the RL agent receives reward $r(s_t, a_t) \in \mathcal{R}$ based on the user feedback on the recommendation provided.
- **Transition probability $\mathcal{P}$:** transition probability $p(s'|s, a) \in \mathcal{P}$ is the probability of transition from $s = s_t$ to $s' = s_{t+1}$ if action a is taken by the agent.
- **Discount factor γ:** discount factor $\gamma \in [0, 1]$ is the discount factor for future rewards. With $\gamma = 0$, the agent becomes *myopic*, i.e., it only focuses on immediate reward. On the contrary, if $\gamma = 1$, the agent becomes *farsighted* and focuses more on future rewards [27].

Given $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P}, \gamma)$, the goal of the RL agent is to find a policy π that maximizes the expected, discounted cumulative reward. In other words,

$$\max_{\pi} \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t r(s_t, a_t)\right], \tag{21}$$

where T is the maximum time step in a finite MDP.

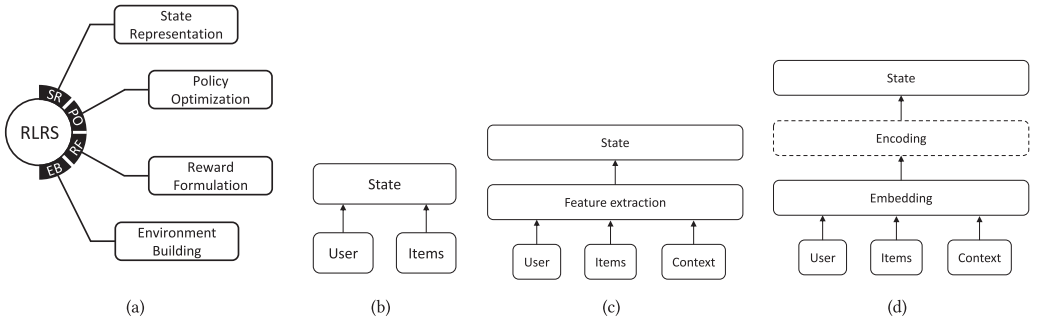


Fig. 3. (a) The proposed RLRS framework, (b) SR1, (c) SR2, (d) SR3. The dashed Encoding module in SR3 means that some models merely use input embeddings as states.

2.5 Proposed RLRS Framework

With carefully studying all RLRSs collected, we found that there are four components common in all of them and believe that a good RLRS should carefully design and address these components. Accordingly, to unify the process of RLRS development, we propose a framework for RLRSs with four key components: (1) State Representation, (2) Policy Optimization, (3) Reward Formulation, and (4) Environment Building. Figure 3(a) depicts this framework. In the following, we explain each component.

State Representation. In the agent-environment RL interface, the state can be *any* information available to the agent. State representation could be as high-level as symbolic descriptions of objects in a room or as low-level as sensor readings [27]. What is important is that defined states should have the *Markov property*. That means, the state signal is not supposed to convey all the information about the environment to the agent, but it should summarize past information such that all relevant information is not missed. A state signal with this property is called Markov. In general, selecting state representation is currently more art than science [27].

In RLRSs, state representation should summarize information about users, items, and the context. We divide state representation in RLRSs into three groups:

(SR1) Treating items as states. When the item space is small, e.g., it includes several web pages in a website, it is possible to treat each item as a state. However, this approach is certainly not scalable when the item space grows large. To tackle the scalability problem in larger items spaces, researchers found that states could indicate a set of items previously rated/consumed by the user. Figure 3(b) depicts this representation.

(SR2) Features from users, items, and context. A popular way for state representation is to extract some features from users, items, and context, as shown in Figure 3(c). User features include demographic information, such as age, race, and gender. Item features may include price, category, and popularity. Context features may include time, platform, and location.

(SR3) Encoded Embeddings. For effective training, deep models in DRL-based RSs need states to be dense, low-dimensional vectors. Figure 3(d) illustrates a general, popular framework for state representation in DRL-based methods. Typically, first user, items, and context features are translated into dense, low-dimensional, continuous vectors called *embeddings*. Then, for better training, this embedding could be encoded using a **recurrent neural network (RNN)** model, which can help the model learn user's sequential preferences [48]. **Gated recurrent units (GRU)** is typically more popular than **long short-term memory (LSTM)** for the RNN module as it has fewer parameters and can achieve the same or better performance [81]. To focus on important parts of the input, some researchers also use an attention layer in the encoding module and add some

weights to the encoded vectors. Finally, the encoded vectors are concatenated to yield the final state.

Policy Optimization. When states are formulated, it is the policy that determines which action to take (i.e., which items to recommend) in each state. For policy optimization, various RL algorithms have been utilized by RLRSs. Before the advent of DRL, RL methods used by RLRSs could be generally classified into tabular and approximate methods. Tabular methods include policy iteration, Q-learning, Sarsa, Sarsa(λ), R-learning, and MCTS. Approximate methods include fitted Q and gradient value iteration. On the other hand, DRL methods could be generally divided into three groups: value-based (DQN), policy gradient (REINFORCE and REINFORCE-wb), and actor-critic (DDPG and PPO) methods. A classification of these algorithms is shown in Figure 2(b).

Reward Formulation. As mentioned earlier, the reward signal from environment reflects how good or bad the agent is performing through selecting actions. Therefore, designing informative reward signal is critical for success/learning of the agent. In fact, in RL, the reward signal is the only way to tell the agent *what* to do, not *how* to do it [27]. In general, defining a proper reward function is a hard problem and it is more of a trial-and-error or engineering process. There is no definite rule to design a good reward function in a specific problem. In RLRSs, we have observed two general trends in designing the reward function: (**R1**) the reward function is a simple, sparse numerical reward, or (**R2**) the reward is a function of one or several observations from the environment.

Environment Building. In general, evaluating RSs is difficult [3, 82]. As a result, building a suitable environment to properly train and evaluate the agent in RLRSs is challenging. To better distinguish between different environment building methods, we generally divide them into three groups: **offline**, **simulation**, and **online**. In offline method, the environment is a static dataset containing the ratings of some users on some items. A common practice in offline methods is to train the agent on the training data (usually 70–80% of the data) and then test it on the remaining data. In simulation studies, usually a user model is built and the algorithm is evaluated while interacting with this user model. This user model could be as simple as a user with some predefined behavior, or it could be more complex and be learnt using available data. In online method, the algorithm is evaluated while interacting with real users and in real-time. This is the best, but most costly method for RLRSs evaluation.

3 REINFORCEMENT LEARNING BASED RECOMMENDER SYSTEMS ALGORITHMS

In this section, we present algorithms in a classified manner. As discussed earlier, first we generally divide RLRSs into RL- and DRL-based methods. Then, we survey algorithms in each category with respect to the RLRS framework.

3.1 RL-based RSs

In this section, we present RL-based RSs; i.e., methods that do not use deep learning for policy optimization. Table 2 provides a quick overview on RL-based methods.

3.1.1 State Representation. As illustrated in Table 2, apart from RPRMS and PHRR, all RL-based methods belong to either SR1 or SR2, which share almost the same proportion (see Figure 5(a)). As mentioned earlier, methods in SR1 use all or a set/tuple of items for state representation. For example, WebWatcher [83], the first RLRS we identified, treats each item (i.e., web page) as a state in a web recommendation scenario. Similarly, References [96, 97] treat each author and learning object as a state in scientific collaborator recommendation and e-learning scenarios, respectively. As stated earlier, while this approach is possible in small state spaces, it is certainly not scalable when the item space grows large. Researchers found that keeping the track of a small set of items already rated/consumed by the user could be informative enough for policy optimization.

Table 2. RL-based Methods

RLRS	Year	SR	PO	RF	EB	Metrics	Dataset	Application*
WebWatcher [83]	1997	SR1	Q-learning	R1	Offline	Accuracy	CMU	Web
Preda et al. [84]	2005	SR1	Sarsa(λ)	R1	Online	RDR	N/A	Web
MDP [50]	2005	SR1	Policy iteration	R2	Online	RC, ED	Mitos	E-commerce
Taghipour et al. [85]	2007	SR1	Q-learning	R2	Offline	Accuracy, Coverage, SG	DePaul	Web
Mahmood et al. [86]	2007	SR2	Policy iteration	R1	Simulation	N/A	N/A	Trip
Taghipour et al. [87]	2008	SR1	Q-learning	R2	Offline	HR, PA, CR, RQ	DePaul	Web
Mahmood et al. [88]	2009	SR2	Policy iteration	R1	Online	26 variables	N/A	Trip
APG [89]	2010	SR1	Q-learning, Sarsa	R2	Simulation, Online	MR, MH, LTR, UR	N/A	Music
Shortreed et al. [90]	2011	SR2	FQI	R2	Offline	PANSS	CATIE	Health
Zhao et al. [91]	2011	SR2	Fitted Q	R2	Simulation	Survival	N/A	Health
RLradio [49]	2012	SR2	R-learning	R1	Online	Listening Time	N/A	Radio channel
Mahmood et al. [92]	2014	SR2	Q-learning	R1	Simulation, Online	BP, ST, BDV, QE	Amazon	Trip
DJ-MC [93]	2014	SR2	MCTS	R2	Simulation, Online	Reward	Million Song	Music
Theocharous et al. [94]	2015	SR2	FQI	R1	Offline	CTR, LTV	N/A	Ad
POMDP-Rec [95]	2016	SR2	Fitted Q	R2	Offline	RMSE	ML1M, YM	
RLWRec [51]	2017	SR1	Q-learning	R2	Offline	CAC	N/A	Music
Zhang et al. [96]	2017	SR1	GVI	R2	Offline	MRR, P, NDCG	ACM	Collaborator
Choi et al. [44]	2018	SR1	Q-learning, Sarsa	R2	Offline	P, R	ML100K, ML1M	
Intayoad et al. [97]	2018	SR1	Sarsa	R1	Offline	RMSE	N/A	E-learning
RPRMS [98]	2019	SR3	Q-learning	R1	Offline	UR	N/A	Music
CAPR [99]	2019	SR2	MCTS	R2	Offline	P, R, F1	Weeplace	POI
PHRR [100]	2020	SR3	MCTS	R2	Offline	HR, F1	Million Song, TPS, HSP	Music
Kokkodis et al. [101]	2021	SR1	Q-learning	R2	Offline	Market Revenue, WI	N/A	Career path

Note: There is no code shared by RL-based methods.
*The application domain is either explicitly mentioned or implicit in respective publications. An empty Application column means the application domain is not clear.

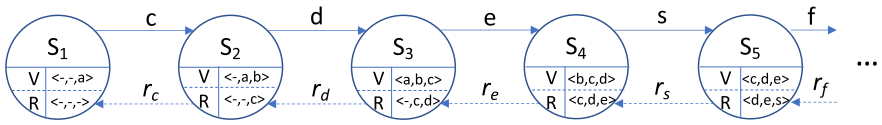
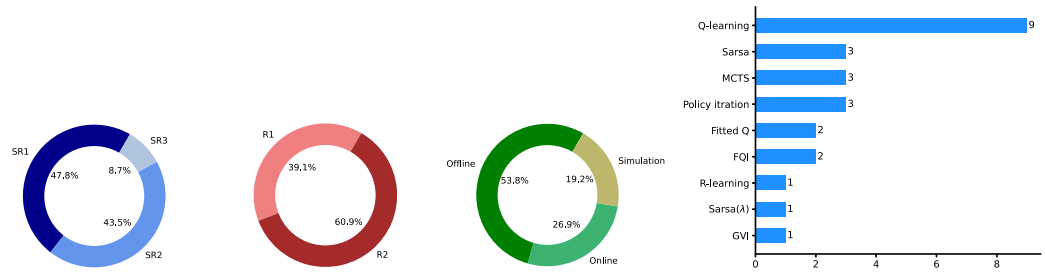


Fig. 4. The concept of sliding window used in [85] for state representation.

Perhaps References [50, 84] are the first RLRSs utilizing this idea, but the idea is better formalized for RLRSs by Reference [85]. Specifically, in a web recommendation application, Taghipour and Kardan [85] borrow the *N-gram* model from the *web usage mining* literature [102] and introduce a *sliding window* to represent states, depicted in Figure 4. In this figure, circles are states, right arrows are actions, and V and R indicate visited and previously recommended pages, respectively. While using this model, the authors assume that knowing the last k pages visited by the user provides enough information to predict their future page requests. It is noteworthy to mention that this set or sliding window in SR1 could indicate any useful information for the purpose of policy optimization, including a set of commercial items [50], concepts in a website [84, 87], emotion classes of songs [89], skills [101], and music songs [51]. In a different setting, Choi et al. [44] formulate the recommendation problem as a *gridworld game* and each grid cell, with its users and items inside, is considered as a state.

Other researchers have proposed to extract some features from user, items, and context and use them for state representation (SR2). Among the first attempts in SR2 is Mahmood et al. works [86, 88, 92] in which a set of variables from user (e.g., the number of times the user has modified his query), agent (e.g., previous action of the agent), and interaction session (e.g., the number of episodes elapsed) are used for state representation. A similar approach is used in RLradio [49] where some variables, containing information about radio channels of user interest and their listening behavior, are defined to represent states. DJ-MC [93] uses an encoding method to represent each song as a vector of song descriptors and each state is the concatenation of k songs vectors in the playlist. Features from weather conditions and time are used in CAPR [99]



(a) State Representation (b) Reward Formulation (c) Environment Building (d) Policy Optimization

Fig. 5. The summary of four components of RLRS framework in RL-based methods.

for state representation in a **point of interest (POI)** recommender. SR2 is specifically popular in healthcare applications in which information about patients are typically recorded by several descriptive features [90, 91]. In a different setting, POMDP-Rec [95] formulates states as belief states using *low-dimensional factor model* [103]. More precisely, with a partially observed user-item matrix, user's behavior observations (O), items' latent features (V), and users' latent interests (U) can be calculated as

$$p(\mathrm{O}|\mathrm{U},\mathrm{V},\sigma^2) = \prod_{i=1}\prod_{j=1}\left(\mathcal{N}\left(\mathrm{O}_{ij}|\mathrm{U}_i^\top \mathrm{V}_j,\sigma^2\right)\right)^{I_{ij}}, \tag{22}$$

$$p\left(\mathrm{U}|\sigma_{\mathrm{U}}^2\right) = \prod_{i=1}\mathcal{N}\left(\mathrm{U}_i|0,\sigma_{\mathrm{U}}^2 I\right), \tag{23}$$

$$p\left(\mathrm{V}|\sigma_{\mathrm{V}}^2\right) = \prod_{j=1}\mathcal{N}\left(\mathrm{V}_j|0,\sigma_{\mathrm{V}}^2 I\right), \tag{24}$$

where $\mathcal{N}(x|\mu,\sigma^2)$ is the probability density function of Gaussian distribution with mean μ and variance σ^2, $I_{ij} \in \{0,1\}$, and $I_{ij} = 1$ means that user i has rated item j. A belief state is then a concatenation of U and V, i.e., $b_{ij} = (\mathrm{U}_i, \mathrm{V}_j)$.

The only works in RL-based methods that lie in SR3 are RPRMS [98] and PHRR [100], both are in the music recommendation domain. In RPRMS, states are a concatenation of songs lyrics embeddings, generated using Word2Vec [104], and audio embeddings, generated using a pre-trained WaveNet model [105]. PHRR uses *weighted matrix factorization* [106] and CNN to embed songs, and similar to DJ-MC, each state is the concatenation of several song vectors.

3.1.2 Policy Optimization. According to Figure 5(d), temporal difference methods, i.e., Q-learning and Sarsa, have been the most popular RL algorithms among RL-based methods [44, 51, 83, 85, 87, 89, 92, 97, 98, 101]. The main reason for this popularity is their simplicity; that is, they are online, model-free, need minimal amount of computation, and can be expressed by a single equation (see Equations (4) and (5)) [27]. Applying Q-learning/Sarsa for policy optimization is quite straightforward and doesn't need any specific modification. Researchers in [85] use a simple trick to have a decreasing learning rate $\alpha = 1/1+\text{visits}(s,a)$ in Equation (4), which helps algorithm convergence. This trick is also used in [51, 89, 92]. A problem with temporal difference methods, like any tabular RL method, is that they lead to the *curse of dimensionality* [107]. To tackle this problem, as discussed earlier, researchers try to manage the state space and keep it small enough.

Among the tabular methods, dynamic programming methods are usually impractical due to their great computational expense and the need to perfect knowledge about the environment. While these algorithms are polynomial in the number of states, performing even one iteration of policy or value iteration methods is often infeasible [108]. To make it practical, Reference [50] uses a

couple of features in their state space and makes some approximations. For instance, one feature of state space in [50] is directionality; the authors argue that a short state cannot follow a long state or the probability of occurring loops in their MDP is not very high. Moreover, Reference [86] keeps the number of policy iteration run to a limited number.

MCTS is a decision-time planning algorithm that benefits from online, incremental, sample-based value estimation and policy improvement [27] and has been employed by [93, 99, 100]. In order to facilitate the agent learning, in case the song space is very large or search time is limited, DJ-MC [93] clusters songs according to song types and then applies MCTS to clustered songs. PHRR [100] adopts similar schemes in policy optimization and song clustering. Similar to AlphaGo [62], CAPR [99] uses **UCT (Upper Confidence Bound applied to Trees)** [109] to solve the exploration vs exploitation trade-off in the selection step of MCTS (see Section 2.2).

Preda and Popescu [84] use Sarsa(λ) with *tile coding* [27] linear approximation for policy optimization. To be able to apply Sarsa(λ), the work transforms epistemic information into arrays of real numbers. Moling et al. [49] define the problem of optimal radio channel recommendation as a continuous task and then employ R-learning [59] to solve it.

On the other hand, some RL-based RSs have used approximate methods for policy optimization, including fitted Q [90, 91, 94, 95] and gradient value iteration [96]. Fitted Q is a flexible framework that can fit any approximation architecture to Q-function [64]. Accordingly, any batch-mode supervised regression algorithms can be used to approximate the Q-function, which can scale well to high dimensional spaces [27]. However, one problem with this method is that it could have a high computational and memory overhead with the increase in the number of *four-tuples* (x_t, u_t, r_t, x_{t+1}), where x_t indicates the system state at time t, u_t the control action taken, r_t the immediate reward, and x_{t+1} the next state of the system [64]. This algorithm has been used by several RLRSs [90, 91, 94, 95]. To fit the Q function in these methods, linear regression [90, 94], support vector regression [91], and neural networks [95] are used. Finally, Zhang et al. [96] introduce a gradient descent version of value iteration algorithm for collaborator recommendation in a multi-agent RL setting.

3.1.3 Reward Formulation. Figure 5(b) shows that R2, with 60% proportion, has been more popular than R1, with 40%, among RL-based methods. In R1, different numerical values have been used for the immediate reward. For instance, Mahmood et al. use +1 in terminal state and a negative number otherwise [86], + 5 for adding a product to travel plan, +1 for showing result page, 0 otherwise [88], and +100 for buying a book, -30 for user quit, and 0 otherwise [92]. On the other hand in R2, researchers have proposed to use different observations from the environment to formulate the reward, including net profit [50], overall survival time [91], some clinical scores (i.e., PANSS) [90], and Jaccard distance between two states [44].

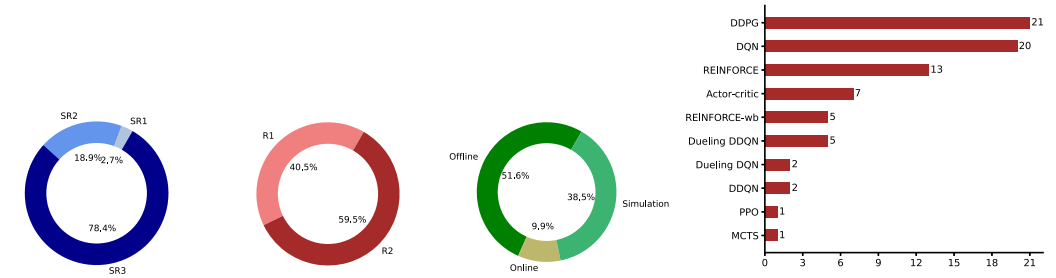
3.1.4 Environment Building. It is observable from Figure 3(c) that the dominant environment building method in RL-based RSs is offline. This makes sense as training the agent and testing the performance on an available dataset is the easiest and safest option. Two popular datasets used in RL-based RSs are MovieLens [110] and Million Song dataset [111].

Another environment building method is simulation, which is a safe method to fine-tune important model parameters before system deployment or an online study. Simulation could be as simple as assuming constant users with predefined preference patterns [89, 91], or it could be more complex and learn user behavior through available data [86, 92, 93]. For example, in a supervised learning task, Mahmood et al. [86] define a user behavior model and use it in a travel RS, called NutKing, to learn transition probabilities. The aim is to know how the simulated user reacts to a certain action of the system.

Table 3. DRL-based Methods

RLRS	Year	SR	PO	RF	EB	Metrics	Dataset	Application	Code
Slate-MDP [112]	2015	SR1	DQN	R1	Simulation	Reward	N/A		
Wolpertinger [24]	2015	SR1	DDPG	R1	Simulation	Reward	N/A		
Nemati et al. [113]	2016	SR2	DQN	R2	Offline	Reward	MIMIC	Healthcare	
CEI [114]	2017	SR3	REINFORCE-wb	R1	Simulation	P, F1, BLEU, Reward	ML1M, MT		
Raghu et al. [115]	2017	SR2	Dueling DDQN	R2	Offline	Mortality Rate	MIMIC	Healthcare	✓
DEERS [48]	2018	SR3	DQN	R1	Offline, Simulation	MAP, NDCG	JD	E-commerce	
Robust DQN [43]	2018	SR2	DDQN	R2	Online	CTR, UV CTR	Taobao	E-commerce	
SRL-RNN [116]	2018	SR3	DDPG	R1	Offline	Mortality Rate, MJS	MIMIC	Healthcare	✓
DRN [47]	2018	SR2	Dueling DDQN	R2	Offline, Online	CTR, NDCG, P, ILS	N/A	News	
Deep Page [46]	2018	SR3	DDPG	R1	Offline, Simulation	P, R, F1, NDCG, MAP	N/A	E-commerce	
CRM [117]	2018	SR3	REINFORCE	R2	Simulation, Online	Reward, SuR, AT, WQR, LRR	Yelp	E-commerce	
Munemasa et al. [45]	2018	SR2	DDPG	R2	Offline	MRR, R	Ekiten	Store	
CapDRL [118]	2019	SR3	DDPG	R1	Offline	P, NDCG	ML100K, ML1M	Movie	
FeedRec [119]	2019	SR2	DQN	R2	Offline, Simulation	ACS, ADS, ART	N/A	Feed streaming	
DRCGR [120]	2019	SR3	DQN	R1	Offline	NDCG, MAP	N/A	E-commerce	
LIRD [40]	2019	SR3	DDPG	R1	Simulation	MAP, NDCG	N/A	E-commerce	
SlateQ [80]	2019	SR2	DQN	R2	Simulation, Online	Reward, AQ	N/A		
Yu et al. [121]	2019	SR3	Actor-critic	R2	Simulation	SuR, NI, NV, RP	UT-Zappos50K	E-commerce	
Tsumita [122]	2019	SR3	DQN	R1	Simulation	AT, SuR	PDS	Restaurant	
Zhang et al. [123]	2019	SR3	REINFORCE	R2	Offline	HR, NDCG	XuetangX	E-learning	✓
DRESS [124]	2019	SR3	PPO	R1	Offline	CVR, CTR, TWIS	JD	E-commerce	
Liu et al. [125]	2019	SR2	Dueling DDQN	R2	Simulation	AAR	N/A	Content	
Yuyan et al. [126]	2019	SR3	DQN	R1	Offline	RMSE	ML100K, ML1M	Movie	
CROMA [127]	2019	SR3	DDPG	R2	Offline	P, R, F1, MRR	Twitter	Tweet	✓
REINFORCE [25]	2019	SR3	REINFORCE	R1	Simulation, Online	ViewTime	N/A		
PCR [128]	2019	SR3	REINFORCE-wb	R1	Simulation	SuR, NI, NV	UT-Zappos50K	E-commerce	
UDQN [129]	2019	SR2	DQN	R1	Offline	Reward	ML100K, ML1M, YM		
IRecGAN [130]	2019	SR3	REINFORCE	R1	Simulation	P	CIKM		✓
Den et al. [131]	2019	SR2	Mixed	R2	Simulation	Reward	N/A		
KGRE-Rec [132]	2019	SR3	REINFORCE-wb	R2	Offline	P, R, HR, NDCG	Amazon	E-commerce	✓
TPGR [133]	2019	SR3	REINFORCE	R2	Simulation	Reward, P, R, F1	ML10M, Netflix		✓
Div-FMCTS [134]	2019	SR3	MCTS	R2	Offline	F1, NDCG	ML		✓
Cascading DQN [135]	2019	SR3	DQN	R2	Simulation	CTR, Reward	ML, LFM, Yelp, YC, AFN		
Ekar [136]	2019	SR3	REINFORCE	R1	Offline	HR, NDCG	ML1M,LFM, DBbook2014		
Pseudo Dyna-Q [137]	2020	SR3	DQN	R2	Simulation	Click, Diversity, Horizon	Taobao, RetailRocket	E-commerce	✓
SADQN [138]	2020	SR3	DQN	R1	Offline	HR, NDCG	LFM, Ciao, Epinions		
Zhao et al. [139]	2020	SR3	Duleing DQN	R2	Offline	SDT, SL, SAD	TikTok	Ad	
Ji et al. [140]	2020	SR2	REINFORCE	R2	Simulation	Earn, VCT, WT, PP30	NY, SF data	Taxi route	
recEnergy [141]	2020	SR2	DQN	R2	Online	RA, ES	N/A	Energy optimization	
GCQN [142]	2020	SR3	DQN	R1	Offline	Reward	ML1M, LFM, Pinterest		
MaHRL [143]	2020	SR3	DDPG	R2	Offline, Simulation	MAP, NDCG, Reward	N/A	E-commerce	
DRR [144, 145]	2018, 2020	SR3	DDPG	R2	Offline, Simulation	P, NDCG, Reward	ML1M, ML100K, YM, Jester		
FairRec [146]	2020	SR3	DDPG	R2	Offline	CVR, WPF	ML100K, Kiva		
EAR [147]	2020	SR3	REINFORCE	R1	Simulation	SuR, AT	Yelp, LFM		✓
MASSA [148]	2020	SR3	DDPG	R2	Offline, Simulation	P, NDCG	Taobao	E-commerce	
DeepChain [149]	2020	SR3	DDPG	R1	Offline, Simulation	MAP, NDCG, Reward	JD	E-commerce	
KGPolicy [150]	2020	SR3	REINFORCE-wb	R2	Offline	R, NDCG	Amazon, LFM, Yelp		✓
KGRL [151]	2020	SR3	DDPG	R2	Offline	P, R, NDCG	6 datasets		
CRSAL [152]	2020	SR3	Actor-critic	R2	Offline	BLEU, ROUGE, EMR, SuR	DSTC2, CR676, MultiWOZ	Restaurant	
CPR [153]	2020	SR3	DQN	R1	Simulation	SuR, AT	Yelp, LFM		✓
Xin et al. [154]	2020	SR3	DDQN, Actor-critic	R1	Offline	HR, NDCG	YC, RetaiRocket		
ADAC [155]	2020	SR3	Actor-critic	R1	Offline	P, R, NDCG, HR	Amazon	E-commerce	
KERL [156]	2020	SR3	REINFORCE	R2	Offline	HR, NDCG	Amazon, LFM		
DRprofiling [157]	2020	SR3	REINFORCE	R2	Offline	P, R, HR	ML, LFM		
KGQR [158]	2020	SR3	DQN	R2	Simulation	Reward, P, R	Data-Crossing, ML20M		
Singh et al. [159]	2020	SR3	REINFORCE	R2	Simulation	Reward, Risk	ML1M		
EDRR [160]	2020	SR3	DQN, DDPG	R2	Offline, Simulation	P, NDCG, MAP, Reward	ML1M, Jester		
SRR [161]	2020	SR3	DQN, DDPG	R2	Offline, Simulation	P, NDCG, Reward	ML1M, ML100K, BC, Jester		
D^2RLIR [162]	2021	SR3	DDPG	R2	Offline	NDCG, P, Diversity	ML1M		
DRGR [163]	2021	SR3	DDPG	R1	Simulation	R, NDCG	ML		✓
FCPO [164]	2021	SR3	DDPG	R1	Offline	R, F1, NDCG, GI, PR	ML1M, ML100K		✓
DHCRS [165]	2021	SR3	DQN	R2	Offline	HR, NDCG	ML1M, 10M, 20M, Netflix		
DARL [166]	2021	SR3	REINFORCE	R2	Offline	HR, NDCG	MOOCCourse, MOOCCube	E-learning	
MKRLN [167]	2021	SR3	REINFORCE-wb	R1	Offline	P, NDCG	Book, MOvie, KKBOX		
MASTER [168]	2021	SR2	DDPG	R2	Offline	MCWT, MCP, TSF, CFR	Beijing, Shanghai Data	Charging spot	
AnchorKG [169]	2021	SR3	Actor-critic	R2	Offline	P, R, NDCG, HR	MIND, BingNews	News	✓
UNICORN [170]	2021	SR3	Dueling DDQN	R1	Simulation	SuR, AT, NDCG	LFM, Taobao, Yelp		
HRL-Rec [171]	2021	SR3	DDPG	R2	Offline, Online	CTR, ACN, AWT	WeChat		✓
GoalRec [172]	2021	SR3	Dueling DDQN	R2	Simulation, Online	CTR, Reward	ML25M, Taobao, YC		
DEAR [173]	2021	SR3	Dueling DQN	R2	Offline	Reward	Douyin	Ad	
VPQ [174]	2021	SR3	Actor-critic	R1	Offline	HR, NDCG	RetailRocket, YC		
SDAC [175]	2021	SR2	Actor-critic	R2	Offline	HR, NDCG	YC, Kaggle		
URL [176]	2021	SR3	REINFORCE	R1	Offline, Online	MAP	N/A		

Online study is the most effective, but costly environment building method for RLRSs. In an early, valuable attempt [50], the performance of the proposed MDP-based RS is evaluated in a two-year online study conducted on an online book store. The study has had a good daily exposure to users, almost 5,000–6,000 different users daily, with a reasonable number of items to recommend (over 15,000), compared to other online studies performed by RL-based methods with only five [89], 13 [92], 47 [93], 469 [88], and 500 [84] users.



(a) State Representation (b) Reward Formulation (c) Environment Building (d) Policy Optimization

Fig. 6. The summary of four components of RLRS framework in DRL-based methods.

3.2 DRL-based RSs

In this section, we study DRL-based methods; those RSs that use a deep learning model for policy optimization. Table 3 provides a quick overview of these methods.

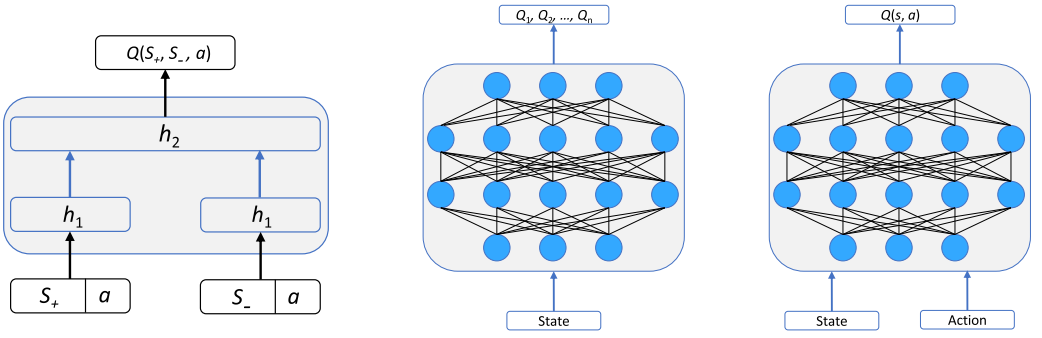
3.2.1 State Representation. As depicted in Figure 6(a), SR3 is the dominant state representation scheme for DRL-based RSs. As stated earlier, this is because deep models are trained more effectively on dense, low-dimensional vectors. Nonetheless, researchers have taken one step further and tried to make the general framework of SR3 (see Figure 3(d)) more effective. Typically, in RLRSs, items *positively* rated by the user are considered as the preferences of the user. However, in DEERS [48], authors discuss that the proportion of negative feedback, e.g., skipped items, could be much larger than the positive one, so they propose to have two states: positive and negative states. Figure 7(a) illustrates this modification. In particular, the input is divided into items with positive and negative feedbacks, are passed through embedding and RNN layers, and fed into Q network where they are concatenated. This technique has also inspired other researchers [120, 165]. Instead of RNN layer, DRCGR [120] uses a convolution layer (with both horizontal and vertical kernels) to encode the embeddings of positive feedbacks. On the other hand, a **generative adversarial network (GAN)** module is trained to generate negative samples. Deep Page [46] also extends the SR3 framework by adding a CNN module between the embedding and RNN layers, in order to learn item spatial display scheme in a page-wise recommendation scenario. Before passing the item embeddings through the CNN module, a page layer is used to convert item embeddings into a 2D grid/matrix for 2D CNN processing. Moreover, authors in [135] propose to use a *position weighting* scheme for state embedding. Formally, if W is a matrix with historical steps as rows and importance weight of positions as columns, the embedding of a state s^t can be defined as

$$s^t = h(F^{t-m:t-1}) = vec\Big[\sigma(F^{t-m:t-1}W + B)\Big], \tag{25}$$

where F is the feature vector of the history with m steps, B is a bias matrix, $\sigma(\cdot)$ is a nonlinear activation, and $vec[\cdot]$ concatenates the matrix columns. The authors claim that this method for state embedding is more efficient for optimization than LSTM. Finally, in D^{2}RLIR [162], a *positional encoding* is added to state embeddings so that the model understands the chronological order of items.

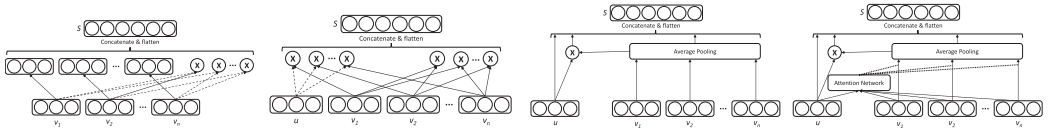
In DRR [144], an individual module called *state representation module* is proposed for the purpose of state formulation. Authors propose three structures to model the interactions between user and items. The first structure, DRR-p, simply concatenates the embeddings of items and their pairwise products, as depicted in Figure 8(a). More formally, if $H = \{v_1, v_2, \ldots, v_n\}$ is the positive interaction history of the user and

$$P = \{w_i v_i \otimes w_j v_j | i, j = 1, 2, \ldots, n\} \tag{26}$$



(a) (b) (c)

Fig. 7. (a) The architecture of DEERS [48], (b) Q network A1 architecture, and (c) Q network A2 architecture.



(a) DRR-p (b) DRR-u (c) DRR-ave (d) DRR-att

Fig. 8. State representation module in DRR [144, 145].

is the weighted pairwise product between items, then state S is defined as the concatenation of H and P, i.e., $S = (H, P)$. In the second structure, DRR-u, the user embedding is also incorporated (shown in Figure 8(b)). That means, with

$$K = \{u \otimes w_i v_i | i = 1, 2, \ldots, n\}, \tag{27}$$

$S = (K, P)$. In the last structure illustrated in Figure 8(c), DRR-ave, an average pooling layer is introduced to eliminate the items' *position bias* in the recommended list. In particular, if

$$G = \{ave(w_i v_i) | i = 1, \ldots, n\}, \tag{28}$$

$S = (u, u \otimes G, G)$. In [145], the authors extend DRR-ave and add an attention network to generate user-dependent weights for each item, as depicted in Figure 8(d). In another work [160], the same authors study the effect of updating the state representation module using a supervised learning signal, and through experimental studies, they show that the recommendation performance could be improved.

Around 20% of DRL-based RSs belong to SR2. For instance, DRN [47] uses user and context features for the purpose of state representation in news recommendation. User features extracted in DRN include the features of the news clicked by the user in different time frames, like one hour, six hours, 24 hours, one week, and one year. These news features include headline provider, ranking, entity name, category, and topic category. Context features used also describe the time context of the news request, including time and weekday. Similar to RL-based methods, SR2 is the popular state representation method in healthcare applications [113, 115]. Nemati et al. [113] use a **partially-observable MDP (POMDP)** formulation in a clinical application. They formulate states as belief states using **discriminative hidden Markov model (DHMM)**.

We found that only two works [24, 112] lie in SR1. Perhaps the reason these works use a simple state representation method is that they are representative works not specifically designed for RSs, but developed to target specific challenges in applying DRL to domains like RSs.

Table 4. DQN-based RSs

RLRS	Year	Algorithm	Architecture	Experience replay	Exploration
Slate-MDP [112]	2015	DQN	A2	Uniform	ϵ-greedy
Nemati et al. [113]	2016	DQN	A1	Uniform*	ϵ-greedy*
Raghu et al. [115]	2017	Dueling DDQN	A1	Prioritized	ϵ-greedy*
DEERS [48]	2018	DQN	A2	Prioritized	ϵ-greedy*
Robust DQN [43]	2018	DDQN	A2	Stratified	ϵ-greedy*
DRN [47]	2018	Dueling DDQN	A2	Uniform	UBGD
FeedRec [119]	2019	DQN	A2	Uniform	Decayed ϵ-greedy
DRCGR [120]	2019	DQN	A2	Uniform	ϵ-greedy
SlateQ [80]	2019	DQN	A2	Uniform	ϵ-greedy*
Tsumita [122]	2019	DQN	A1	Uniform*	ϵ-greedy*
Liu et al. [125]	2019	Dueling DDQN	A1	Uniform	Decayed ϵ-greedy
Yuyan et al. [126]	2019	DQN	A1	Uniform	ϵ-greedy
UDQN [129]	2019	DQN	A1	Uniform	ϵ-greedy
Cascading DQN [135]	2019	DQN	A2	Uniform	ϵ-greedy
Pseudo Dyna-Q [137]	2020	DQN	A2	Uniform	Decayed ϵ-greedy
Zhao et al. [139]	2020	Duleing DQN	A2	Uniform	ϵ-greedy*
recEnergy [141]	2020	DQN	A1	Uniform	Boltzmann exploration
SADQN [138]	2020	DQN	A2	Not using	ϵ-greedy
CPR [153]	2020	DQN	A1	Uniform	ϵ-greedy*
Xin et al. [154]	2020	DDQN	A2	Uniform	ϵ-greedy*
KGQR [158]	2020	Dueling DDQN	A2	Uniform	ϵ-greedy
EDRR [160]	2020	DQN	A1	Uniform*	ϵ-greedy*
GCQN [142]	2020	DQN	A2	Uniform*	ϵ-greedy
SRR [161]	2020	DQN	A1	Uniform*	ϵ-greedy*
DHCRS [165]	2021	DQN	A1	Uniform	ϵ-greedy*
UNICORN [170]	2021	Dueling DDQN	A2	Prioritized	ϵ-greedy
GoalRec [172]	2021	DQN	A2	Hindsight	Decayed ϵ-greedy
DEAR [173]	2021	Dueling DQN	Hybrid	Uniform	ϵ-greedy*

*There is no indication about this element in respective papers and we have assumed uniform/ϵ-greedy because all these algorithms are based on DQN.

3.2.2 Policy Optimization. After defining states, the role of policy π is to map states to actions. Policy optimization algorithms used by DRL-based RSs could be generally divided into value-based, policy gradient, and actor-critic methods.

Value-based methods. Apart from MCTS used in [134], DQN and its extensions, i.e., DDQN, dueling DQN, and dueling DDQN, are the ruling value-based methods. Basically, there are three main elements in DQN: (1) Q network architecture, (2) experience replay, and (3) exploration. We survey DQN-based methods according to these elements and Table 4 summarizes DQN-based methods.

(1) Q network Architecture. Figure 7 depicts two possible architectures of Q network used by DQN-based RLRSs. The original architecture (A1), introduced in [67, 68], receives the state and emits the Q value of all actions, indicated by $Q_1, \ldots, Q_n$ in Figure 7(b). While A1 woks fine when the action space is small, its applicability to the RS domain with a large, and even huge (in the order of millions), action space is questionable. Another possible architecture (A2) is to receive the pair of state and action, and then to emit the Q value of the pair, i.e., $Q(s, a)$ (depicted in Figure 7(c)). Although A2 solves A1's problem, a problem with A2 is that the time complexity of the model could be high.

Despite the original DQN where CNN is used for Q network to process the image data, Q network in RLRSs is typically composed of several FC layers, as the input, i.e., states or actions, are in the form of 1D vectors. For example, as stated before, DEERS [48] uses two types of states as the input into Q network: positive and negative states, depicted in 7(a). Q network is a five-layer FC network where the first three layers are separate for positive and negative states, and then the last

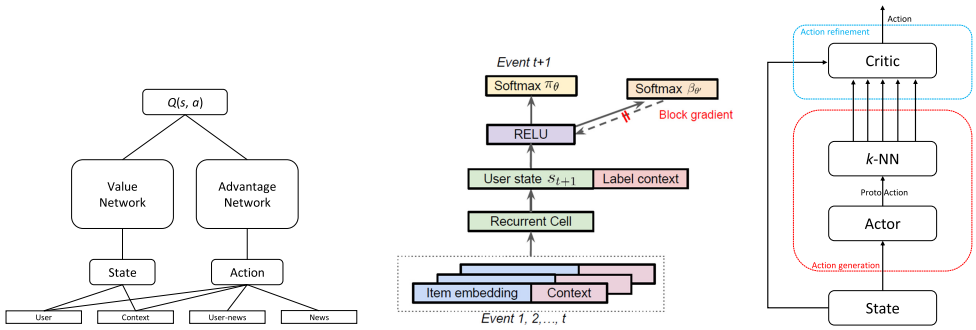


Fig. 9. (a) Dueling architecture used in DRN [47], (b) The neural architecture of policy π_θ [25], (c) Wolpertinger architecture [24].

two layers connect both states, emitting the Q value of a given state and action pair. To take this dual-state architecture into account, the original loss function of DQN in Equation (14) is modified as

$$L(\theta_i) = \mathbb{E}_{s,a\sim\rho(\cdot)}\Big[\big(y_i - Q(s_+, s_-, a; \theta_i)\big)^2\Big], \tag{29}$$

where $y_i = \mathbb{E}_{s'}[r + \gamma \max_{a'} Q(s'_+, s'_-, a'; \theta_{i-1})|s_+, s_-, a]$. Consequently, the gradient of the loss function becomes

$$\nabla_{\theta_i} L(\theta_i) = \mathbb{E}_{s,a\sim\rho(\cdot)}\Big[\big(r + \gamma \max_{a'} Q(s'_+, s'_-, a'; \theta_{i-1}) - Q(s_+, s_-, a; \theta_i)\big)\nabla_{\theta_i} Q(s_+, s_-, a; \theta_i)\Big]. \tag{30}$$

Other researchers have employed DQN's extensions. For instance, DRN [47] adopts dueling DDQN for policy optimization in news recommendation. In particular, the authors argue that while the reward of taking an action is impacted by all features, i.e., user, news, context, and user-news features, there is a reward that is impacted by merely user and context features. Accordingly, the Q function is divided into value function $V(s)$ and advantage function $A(s, a)$. As depicted in Figure 9(a), while $V(s)$ is fed with state features, the input into $A(s, a)$ is comprised of state and action features.

DEAR [173] studies the problem of advertising along with recommendation. It combines the two architectures of DQN Q network, i.e., A1 and A2, and the resulting architecture generates the Q value of a list of candidate ads if inserted in the recommendation list. In other words, the input is similar to A2 architecture, i.e., state and action, and the output is the same as A1, which is a list containing the Q values of all state-action pairs.

(2) Experience Replay. According to Table 4, the vast majority (22 out of 28) of DQN-based RSs use the original uniform sampling to replay collected experiences. Also, only three of them use prioritized experience replay [48, 115, 170]. Authors in [43] propose to use *stratified sampling* replay instead of uniform sampling to address the variance of sampling in dynamic environments. Stratified sampling is a sampling technique from a population in which the entire population is partitioned into several groups (called strata) and then samples are randomly selected from these strata [177]. They propose to use some stable features from customers, like gender, age, and geography, as strata.

GoalRec [172] uses *hindsight* experience replay [178]. The main idea in hindsight replay is to learn from an undesirable outcome as much as from a desirable outcome. Since the goal has no effect on the dynamics of the environment, a failed trajectory is re-labelled as a successful one, as if the state in the trajectory is the actual goal. This considerably improves sample efficiency.

In contrast to existing DQN-based methods, SADQN [138] does not use experience replay for training. Instead, in each episode of the training phase, a user is sampled from the user set and the agent is trained on the available interactions until it is converged. Using experiments, authors claim that the experience replay in fact diminishes the performance of SADQN.

(3) Exploration. Although exploration is an important factor in learning of the agent, many DQN-based methods have seemingly overlooked it, as there is no specific indication about this in respective publications. Apart from simple exploration techniques like ϵ-greedy, DRN [47] proposes to use an exploration approach similar to *dueling bandit gradient descent* algorithm [179]. In particular, there is a separate network for exploration called *explore network* and its parameters can be obtained using a disturbance to the parameters of current network with parameters W

$$\Delta W = \alpha \cdot \text{rand}(-1, 1) \cdot W, \tag{31}$$

where α is the explore coefficient. Then, the agent generates a merged list of recommendations using probabilistically interleaving between items found by current network and explore network.

In recEnergy [141], to balance the exploration vs exploitation trade-off, *Boltzmann exploration* [27] is used. More precisely, the output Q values of actions from Q network are passed through a softmax equation as

$$P(a) = \frac{\exp\frac{Q(a)}{\tau}}{\sum_{i=1}^{n} \exp\frac{Q(i)}{\tau}}, \tag{32}$$

where τ is a temperature and is decayed over time. This method guarantees that the model explores more often initially, and then it starts to exploit actions with larger Q values more frequently.

Policy Gradient methods. In contrast with value-based methods, policy gradient methods learn a parameterized policy without the need to a value function. REINFORCE is a Monte Carlo, stochastic gradient method that directly updates the policy weights. The major problems of REINFORCE algorithm are high variance and slow learning. These problems come from the Monte Carlo nature of REINFORCE, as it selects samples randomly and updates are made when the episode is completed.

In a valuable work, Reference [25] adapts REINFORCE algorithm to a neural candidate generator with a very large action space. In particular, in an online RL setting, the estimator of the policy gradient can be expressed as

$$\sum_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{|\tau|} R_t \Delta_\theta \log \pi_\theta(a_t|s_t) \right], \tag{33}$$

where π_θ is the parametrized policy, $\tau = (s_0, a_0, s_1, \ldots)$, and R_t is the cumulative reward. Since in the RS setting, unlike classical RL problems, the online or real time interaction between the agent and environment is infeasible and usually only logged feedback is available, applying the policy gradient in Equation (33) is biased and needs correction. The off-policy-corrected policy gradient estimator is then:

$$\sum_{\tau \sim \beta} \frac{\pi_\theta(\tau)}{\beta(\tau)} \left[\sum_{t=0}^{|\tau|} R_t \Delta_\theta \log \pi_\theta(a_t|s_t) \right], \tag{34}$$

where β is the behavior policy and

$$\frac{\pi_\theta(\tau)}{\beta(\tau)} = \frac{\rho(s_0) \prod_{t=0}^{|\tau|} P(s_{t+1}|s_t, a_t) \pi(a_t|s_t)}{\rho(s_0) \prod_{t=0}^{|\tau|} P(s_{t+1}|s_t, a_t) \beta(a_t|s_t)} = \prod_{t=0}^{|\tau|} \frac{\pi(a_t|s_t)}{\beta(a_t|s_t)} \tag{35}$$

is the importance weight. Since this correction generates a huge variance for the estimator due to the chained products, authors use first-order approximation for importance weights, leading to

the following biased estimator with a lower variance for the estimator:

$$\sum_{\tau\sim\beta}\left[\sum_{t=0}^{|\tau|}\frac{\pi_\theta(a_t|s_t)}{\beta(a_t|s_t)}R_t\Delta_\theta\log\pi_\theta(a_t|s_t)\right]. \tag{36}$$

Figure 9(b) illustrates the neural architecture of the parametrized policy π_θ in Equation (36).

As discussed in Section 2.2, REINFORCE-wb adds a baseline to REINFORCE's update rule in order to decrease the variance (see Equation (12)). Several RLRSs have used this approach [114, 128, 132, 150, 167]. Specifically, the baseline in these methods is a value network [114, 132, 167], a constraint [128], and average reward [150]. However, it is not clear how other REINFORCE-based RSs [117, 123, 136, 140, 147, 157, 159, 166] tackle the variance problem.

Following SeqGAN [180], IRecGAN [130] employs GANs to develop a model-based RL recommender. In particular, the generator is responsible to generate recommendations and to model user behavior, and the discriminator is used to rescale the generated rewards. Using both generated and offline data, REINFORCE is used to optimize the recommendation policy. Similar to SeqGAN, to reduce the variance, IRecGAN uses MCTS with roll-out policy, i.e., sampling N sequences from interaction between the recommender and user model and then averaging the estimations.

Actor-Critic Methods. DDPG is the base method used in almost all actor-critic based RLRSs. DDPG uses an actor-critic architecture to combine DPG and DQN. Actor, also called policy network, is responsible to generate actions, and critic, a DQN module, is responsible to evaluate the action taken. The original DDPG uses either several FC layers or convolutional plus FC layers when the input is pixel. The output layer of actor is a `tanh` layer to bound actions. For exploration, DDPG uses a temporally correlated noise, **Ornstein-Uhlenbeck (OU)** process [181], that is suitable for physical environments with momentum. Also, similar to DQN, experience replay with uniform sampling is used.

Wolpertinger [24] is the first actor-critic method based on DDPG to handle large discrete action spaces, with a recommendation case study. The idea is to provide a method that has sub-linear complexity w.r.t. action space and generalizable over actions. As depicted in Figure 9(c), Wolpertinger consists of two parts: action generation and action refinement. In the first part, proto-actions are generated by the actor in continuous space and then are mapped to discrete space using k-nearest neighbor (k-NN) method. More precisely, the proto-action $\hat{a}$ is generated by actor as

$$\hat{a} = f_\theta(s). \tag{37}$$

This proto-action is not likely to be a valid action so $\hat{a}$ is mapped to an element in $\mathcal{A}$ as

$$g_k(\hat{a}) = \arg\min_{a\in\mathcal{A}}{}^{k}\ |a-\hat{a}|_2. \tag{38}$$

In the second part, outlier actions are filtered using a critic, which selects the best action that has the maximum Q value. In other words,

$$\pi_\theta(s) = \arg\max_{a\in g_k}\ Q_\theta(s,a). \tag{39}$$

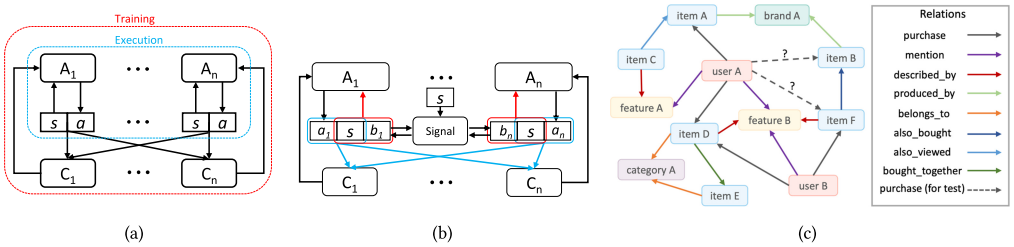
Wolpertinger is trained using DDPG. For exploration, for the recommendation task, Wolpertinger uses a guided ϵ-greedy exploration technique. In particular, the exploration is restricted to a likely good set of actions provided by the environment simulator.

The vast majority of actor-critic methods are based on DDPG [24, 40, 45, 46, 116, 118, 127, 143–146, 148, 149, 151, 160–164, 168, 171]. Table 5 summarizes these methods. As depicted, only DRR uses prioritized experience replay; the remaining algorithms either use uniform sampling or there is no clue about this in respective publications. Another worthwhile observation from Table 5

Table 5. DDPG-based RSs

RLRS	Year	Experience replay	Exploration
Wolpertinger [24]	2015	Uniform	Guided ϵ-greedy
SRL-RNN [116]	2018	Uniform	N/A
Deep Page [46]	2018	Uniform	N/A
DRR [144, 145]	2018, 2020	Prioritized	ϵ-greedy
Munemasa et al. [45]	2018	Uniform	N/A
CapDRL [118]	2019	Uniform*	N/A
LIRD [40]	2019	Uniform	N/A
CROMA [127]	2019	Uniform	N/A
MaHRL [143]	2020	Uniform	N/A
FairRec [146]	2020	Uniform*	N/A
DeepChain [149]	2020	Uniform	N/A
KGRL [151]	2020	Uniform	N/A
EDRR [160]	2020	Uniform*	N/A
SRR [161]	2020	Uniform*	N/A
MASSA [148]	2021	Uniform	Entropy-regularized
FCPO [164]	2021	Uniform	N/A
HRL-Rec [171]	2021	Uniform*	ϵ-greedy
MASTER [168]	2021	Uniform	N/A
D^{2}RLIR [162]	2021	Uniform	N/A
DRGR [163]	2021	Uniform*	OU noise

*There is no indication about this element in respective papers and we have assumed uniform because all these algorithms are based on DDPG.



(a) (b) (c)

Fig. 10. (a) MADDPG architecture, (b) MASSA architecture, (c) Graph-based reasoning example [132].

is that the vast majority of algorithms do not talk about exploration. Of five algorithms with a described exploration method, three of them, i.e., Wolpertinger, DRR, and HRL-Recused, are based on ϵ-greedy. Similar to DDPG, DRGR uses OU process to encourage better exploration for the actor. However, as stated earlier, OU noise is suitable for physical processes. Finally, MASSA introduces a novel entropy-regularized method for exploration, a method similar to soft actor-critic [182].

Actor-critic seems as a popular architecture for **multi-agent RL (MARL)**. Centralized learning/training with decentralized execution [183, 184] is a suitable framework for a multi-agent setting and adopted by CROMA, MASSA, DeepChain, and MASTER. For instance, Figure 10(a) depicts the architecture of MADDPG [184], which utilizes a centralized training and decentralized execution framework. MASSA builds upon this architecture and adds a signal network to the MADDPG architecture, depicted in Figure 10(b), which is responsible to ease the cooperation between decentralized actors.

There are a couple of actor-critic based methods that use adversarial training for a better policy learning [152, 155]. For instance, CRSAL [152] extends soft actor-critic [182] with adversarial learning, by adding a discriminator inside the critic to distinguish between dialogues generated by the policy network and real users. In a path reasoning scenario over knowledge graph, ADAC [155] uses adversarial imitation learning [185] and defines two-path and meta-path discriminators to distinguish expert paths from paths generated by the actor.

In contrast to other actor-critic methods, DRESS [124] uses PPO and SDAC [175] proposes a stochastic discrete actor-critic. Authors in SDAC propose a general offline framework for RLRSs. They first formulate the recommendation problem as a probabilistic generative model. Then, a stochastic actor-critic algorithm is proposed to optimize the recommendation policy.

3.2.3 Reward Formulation. As depicted in Figure 6(b), the majority (60%) of DRL-based RSs belong to R2. A common pattern frequently used by RLRSs in R2 is to formulate the reward as a function, or a simple combination, of several factors or metrics [47, 113, 115, 119, 121, 127, 131, 133, 141, 150, 152, 156–158, 160, 161, 169, 173, 175]. For example, in a news recommendation scenario, reward in DRN [47] is a function of user click and user activeness. The rationale behind factoring in the user activeness is that a good recommendation should motivate the user to use or interact with the system again. Authors use *survival models* [186] to model user return [187] and user activeness. FeedRec [119] formulates the reward as a weighted sum of instant metrics, including user click and purchase, and delayed metrics, like browsing depth and dwell time. Authors consider user clicks as instant metric, and browsing depth and return time as delayed metrics. Yu et al. [121] design an advantage function composed of visual, attribute, and history matching rewards, in order to tackle the multi-modal recommendation problem.

Robust DQN [43] proposes to use *approximate regretted reward* to improve reward estimation. The idea is to use two different rewards, i.e., current and optimal rewards, and then calculate the regret as the final reward. Since calculating the optimal reward in the real is not possible, they propose to use an alternative, benchmark reward, which is the average reward achieved by applying the model to a subset of users.

A simple but effective scheme for multi-objective optimization in RLRSs is to formulate the reward as a multi-objective function [134, 146, 159, 162]. Singh et al. [159], for example, use this idea for a safe RLRS. The format of reward in their work is as follows

$$R_{mo} = R_t - \mathrm{C}_{\mathrm{risk}}, \tag{40}$$

where $\mathrm{C}_{\mathrm{risk}}$ is a health risk constraint. This reward function balances between reward maximization and health constraint preservation. A similar formulation is used to balance the accuracy trade-off with diversity [134, 162] and fairness [146].

In hierarchical RL, two reward functions, i.e., for low-level and high-level agents, should be defined [114, 123, 143, 165, 166, 171]. For example, in HRL-Rec [171], click times on the recommended channel is considered as the low-level agent's reward, while the reward for the high-level agent is composed of four factors, including click times, dwell time, list-level diversity, and item novelty.

In KGRE-Rec [132], a delayed reward function is used. Authors discuss that it is impossible to define a sparse, binary reward when there is no pre-known good/targeted item in their recommendation problem. Instead, the agent is encouraged to find good paths in the graph, those that lead to an item of user interest with high probability. Thus, the agent is received a reward only in a terminal state. The same idea can be seen in MASTER [168], where a *lazy reward* is given to the agent when a charging request is successful. However, the idea of rewarding the agent only in the terminal state is not always practical. For example, Liu et al. discuss that since there is no

well-defined terminal state in AnchorKG [169], the reward function should be composed of immediate and terminal rewards.

Another reward formulation method used in R2 is to define the reward as a *distance* between the recommended item and a target item [45, 140, 151]. In KGRL [151], for instance, the reward is based on the distance between the predicted item and target item in the graph

$$r = \frac{100}{\sqrt{d(v_p, v_t) + \epsilon}} \cdot W_{pt}, \tag{41}$$

where $d(v_p, v_t)$ is the distance between predicted item p and target item t, ϵ is a regulizer, and W_{pt} is the sum of weights of the shortest path from v_p to v_t. Distance d is calculated using Dijkstra's algorithm.

On the other hand, perhaps the simplest method of reward definition for the designer is to empirically use several real values for different goals in the system, which is usually used in R1. For example, Zhao et al. use a similar pattern of numerical reward in their proposals [40, 46, 48, 149] and reward three behaviors of users, namely skip, click, and order, with some numbers, e.g., 0, 1, and 5, respectively. The same pattern can be observed in other RLRSs belonging to R1 [114, 116, 118, 120, 126, 138, 142].

Similarly, in a conversational RS scenario, EAR [147] defines a sparse reward function with four pre-defined values, i.e., strongly positive reward when the recommendation is successful (r_s), a positive reward if the user gives positive feedback on asked attribute (r_a), a strong negative reward for user quit (r_q), and a slight negative for every conversation turn (r_p). The total reward is the sum of these rewards and in the experiments they use values $r_s = 1$, $r_a = 0.1$, $r_q = -0.3$, and $r_p = -0.1$. A similar reward function is used in other conversational RSs [122, 153, 170].

Delayed reward is also used in R1 by some graph-based RLRSs [136, 155, 167], where agent is only rewarded when it reaches the terminal state. For example in Ekar [136], the agent is rewarded with +1 if it reaches an item in the terminal state with which the user has interacted, 0 if it reaches an item but the user has not interacted with, and −1 if the entity reached is not an item in the graph.

3.2.4 Environment Building. As shown in Figure 6(c), more than half of DRL-based RSs use an offline method for environment building. Almost 40% of methods use a simulator, and only in 10% online study is used. Compared to RL-based methods, while a similar proportion use the offline method, the uptake of simulation and online schemes has been doubled and diminished by almost 60%, respectively. This graph shows that conducting an online study has become more difficult or costly, and simulation is getting more and more popular among the RLRS community.

Among those conducting a simulation study, SlateQ [80] introduces an open-source RLRS simulation environment, called RecSim [188], which gives the researcher flexibility to evaluate their algorithms in different settings. Cascading DQN [135] uses GANs to simulate a real user and estimate the reward function from logged data. More precisely, the GAN training is formulated as

$$\min_{\theta} \max_{\alpha} \left(\mathbb{E}_{\phi_\alpha} \left[\sum_{t=1}^{T} r_\theta(s^t_{true}, a^t) \right] - R(\phi_\alpha)/\eta \right) - \sum_{t=1}^{T} r_\theta \left(s^t_{true}, a^t_{true} \right), \tag{42}$$

where η is a regularization term, *true* means real data, ϕ represents the generator and generates user's next action, and r is the discriminator trying to differentiate between generated actions and real actions.

In DEERS [48], a user simulator, with the same architecture as DEERS, is trained on user logs. However, the output layer of the simulator is a softmax layer to predict the user feedback

(immediate reward) based on the input (pair of state and recommended item). Authors claim that the simulator is 90% precise in predicting user feedback. The same approach for simulation study has been used by other RLRSs [46, 119, 121, 128, 137, 143, 145, 148, 149, 160, 161, 172]. For instance, a similar idea is used in [119], but the simulator (S Network) provides different feedbacks, including user response, dwell time, revisited time, and a binary indicator if the user is leaving or not. In Pseudo Dyna-Q [137], a world model (user simulator) is trained by minimizing an error between online and offline rewards. *Truncated importance sampling* [189] is used to alleviate the bias in the offline data.

Another popular simulation method is to develop a simulator based on collaborative filtering [40, 125, 163]. To be specific, LIRD [40] builds a memory with (s, a, r) tuples seen in the logs dataset and uses a similarity method, based on cosine similarity, to find the closest state-action pair to the current state and action recommended. DRR [125] and DRGR [163] use the same intuition but based on probabilistic matrix factorization [103] and matrix factorization, respectively.

Building a simulator for conversational RSs is more challenging than that for typical recommendation scenarios mentioned above, as there are a small number of public datasets available to have both user rating and natural language/user chat for that item-rating pair. CRM [117] tackles this problem by creating simulated users based on Yelp [190] data and a dialogue corpus, collected using *crowd sourcing workers*. The simulated users have three behaviors: answering the agent question, finding the target item in a list, and leaving the dialogue. The same scheme has been used in other conversational RLRSs [122, 147, 153, 170].

Generally speaking, performing a good online study has become more challenging in modern RSs with huge user and item spaces, as the risk of implementing a non-optimal RS is very high. As discussed before, this is the most probable reason of a considerable decrease in the popularity of online study among DRL-based methods compared to RL-based methods. Perhaps two of the best online studies among RLRSs are conducted in [25] and [80] and performed on YouTube.

4 EMERGING TOPICS

Having reviewed RLRSs, we have recognized that there are a couple of trends that are being formed among DRL-based RSs and have the potential to become mature in the course of time. In this section, we briefly review these emerging topics.

Multi-agent RL. Multi-Agent RL (MARL) is a generalization of a single-agent RL and is formulated as a *Markov/stochastic game* [191, 192]. MARL enables RLRSs to target several or complex tasks by dividing them into sub-tasks and each agent can handle one of them. For example, instead of optimizing a single strategy for all scenarios in an e-commerce application of RSs (like entrance page, item recommendation, and checking out purchases), there could be several RS agents each of which responsible for a specific scenario and the final policy is jointly optimized between them [149]. From a *game theory* prospective, MARL methods can be generally divided into three groups: fully cooperative, fully competitive, and a mix of the two [192].

Recently, several RLRSs have employed MARL to tackle the problems of scholarly collaborator recommendation [96], mention recommendation in Twitter [127], page-wise recommendation [148], whole-chain recommendation [149], and charging spot recommendation [168]. As stated earlier, actor-critic with centralized training and decentralized execution has been a popular framework for DRL-based RLRSs employing MARL [127, 148, 149, 168]. In a cooperative setting [127, 148, 149, 168], a challenge is to determine the role of each *player* in the overall's team success. CROMA [127], with two actors and a centralized critic, tackles this problem by a differentiated advantage scheme using reverse operation. Specifically, each actor agent can estimate its particular advantage by subtracting the overall Q value of the joint action, computed by the

centralized critic, from the Q value of a reverse action. A similar architecture is used in DeepChain [149] to jointly optimize the overall reward of a session. It is not, however, clear how DeepChain solves the aforementioned problem, i.e., shared reward for two actors, which is critical for their effective training. In MASSA [148], a MARL with separate actor and critic agents is used to tackle a multi-module, page-wise recommendation. A game theory concept called *correlated equilibrium* [193] in the format of a *signal network* is used to handle the communication between agents. MASTER [168] considers each charging spot for electric vehicles as a distributed agent and uses a centralized critic to coordinate these agents. A couple of techniques, including *bidding game* and multiple critics, are employed to address challenges like cooperation between agents, future competition between requests, and multi-objective optimization. In a different, competitive scenario, authors in [96] use MARL to recommend scientific collaborator. Each author looking for a collaborator is deemed as an agent and learn an optimal policy using gradient value iteration algorithm.

Hierarchical and meta-controller RL. Hierarchical RL (HRL) was initially sought to address the scalability problem in traditional RL algorithms [194]. In HRL, however, it is possible to define multiple layers of policies, each of which can be trained to provide higher levels of temporal and behavioral abstractions, leading to the ability of solving more complex tasks [195, 196]. Recommendation is not an exception and several researchers have utilized HRL in the RS domain [114, 123, 143, 165, 166, 171]. Generally speaking, all these RLRSs define a HRL with two levels of hierarchies where a high-level agent defines a high-level/abstract goal and a low-level agent tries to satisfy that goal. CEI [114] builds a conversational RS on a deep HRL method [197], which uses ideas from a popular and traditional HRL framework, called *options* [198]. CEI uses a meta-controller that selects a goal (chitchat or recommendation) in a given state and a controller makes an action following a goal-specific policy to satisfy the defined goal. Zhang et al. [123] employ HRL for course recommendation in massive open online courses (MOOCs). The key idea is to develop a profile reviser using HRL, which removes noisy courses from users profiles. This is decomposed into two high-level and low-level tasks: given a user profile and a target course, should the profile be revised (high-level) and if yes, which courses in the profile should be removed (low-level). DARL [166] improves Zhang et al.'s RS by making the recommendation unit more adaptive. That means, they equip the basic recommendation module in Zhang's work with an attention mechanism to take dynamic users' interest in diverse courses into account. HRL-Rec [171] uses HRL in an integrated recommendation scenario. A low-level agent generates a list of channels, and a high-level agent recommends a list of items with the channel constraint selected by low-level agent. Moreover, MaHRL [143] tackles the sparse conversion metric in e-commerce by using HRL. More precisely, there is a high-level agent responsible to track long-term sparse conversion interest by setting multiple abstract goals for the low-level agent, while the low-level agent follows these goals and tries to catch short-term click interest. Finally, DHCRS [165] tries to tackle the large action space in RSs through using a two-level HRL, where a high-level DQN selects categories of items and a low-level DQN selects an item in the category to recommend.

In an emerging topic, a group of researchers have used RL as a *meta-controller* module in conversational RSs. That means, instead of using RL to optimize the recommendation policy, similar to HRL, these methods use RL to select either recommending items or asking questions from users to refine recommendations. But different from HRL, there is only one level using RL and the recommendation unit uses other techniques, like supervised learning, to generate the recommendations. This is the common theme in a couple of RLRSs [117, 122, 131, 147, 152, 153, 170]. For instance, CRM [117] is composed of three main parts: a belief tracker, a recommender, and a policy network (RL module). The belief tracker unit is responsible to extract facet-value pairs (some constraints) from user utterances and convert them to beliefs using an LSTM network. Factorization

machine [199] is used in the recommender to generate a set of recommendations. Finally, a neural policy network, optimized by REINFORCE, is used to manage the conversational system, i.e., to decide either to ask for more information from the user or to recommend the items.

Knowledge graph based RLRSs. Incorporating knowledge graphs into RSs can boost recommendation accuracy and explainability [38]. Utilizing knowledge graphs provide RLRSs with different useful information, which can address sample inefficiency in DRL. Recently, many researchers started to use this idea and boost recommendation performance and explainability [132, 136, 150, 151, 155–158, 167, 169, 170]. For example, the idea in KGRE-Rec [132] is to not only recommend a set of items, but also the paths in the knowledge graph to show the reason why the method has made these recommendations. An example of this graph reasoning is depicted in Figure 10(c). For a given user A, the algorithm should find items B and F with their reasoning paths in the graph, like {User $A \rightarrow$ Item $A \rightarrow$ Brand $A \rightarrow$ Item B} and {User $A \rightarrow$ Feature $B \rightarrow$ Item F}. Obviously, graph based techniques face the scalability problem as the number of nodes and links can significantly grow, proportional to the number of users and items. To address this problem, KGRE-Rec proposes a *user-conditional action pruning* strategy, which uses a scoring function to only keep important edges conditioned on the starting user.

Supervised RL. The key feature that distinguishes RL from supervised learning is whether the training data serves as an evaluation signal, like numerical reward, or as an error signal [200]. However, these methods, RL and supervised learning, can be combined to improve policy learning when both signals are available. Wang et al. [116] use this idea to dynamically recommend treatment options to patients. The idea is while the model should maximize the expected return, it should minimize the difference from doctors' prescriptions. In particular, in an actor-critic architecture, the actor is responsible to recommend the best prescription by optimizing the following objective function:

$$J(\theta) = (1 - \alpha)J_{RL}(\theta) + \alpha(-J_{SL}(\theta)), \tag{43}$$

where $J_{RL}(\theta)$ and $J_{SL}(\theta)$ are the objective function of RL and supervised learning tasks, respectively, and α is a weight factor. Similarly, Liu et al. [160, 161] leverage supervised learning to guide the RL module in learning better policies. More precisely, in [160], a supervised learning signal helps generate better embeddings for state representation, and in [161], a supervised learning model is trained to guide the RL policy to focus on short-term reward and to generate top-aware recommendations.

Imitation Learning and Auxiliary Tasks. In addition to the above emerging topics, there are some topics, although less popular compared to the ones discussed above, we think that they have the potential to become emerging topics in the future. These topics include adversarial RL/training, safe RL, self-supervised learning, and imitation learning.

Adversarial training using GANs is an interesting emerging topic used in [130, 152, 155]. As mentioned earlier in Actor-Critic Methods section, CRSAL [152] and ADAC [155] use adversarial training integrated with actor-critic architecture for better agent training. Moreover, as discussed in policy gradient methods, IRecGAN [130] proposes a model-based RL based on GANs for the purpose of variance reduction and sample efficiency.

In safe RL, it is important for the agent to respect some safety constraints, along with maximizing the long term reward [201]. In Reference [159], an RS based on multi-objective safe RL is proposed to improve the long term well-being of users. In particular, the agent simultaneously tries to maximize user engagement and the health of worst-case user.

Self-supervised learning (SSL) empowers the model to utilize labels available freely with the data. In [154], a framework is introduced to augment RLRSs with SSL. More precisely, authors propose a framework with two heads: RL and SSL. While the RL head is used as a regularizer to tune recommendations, the SSL head provides negative samples to update parameters.

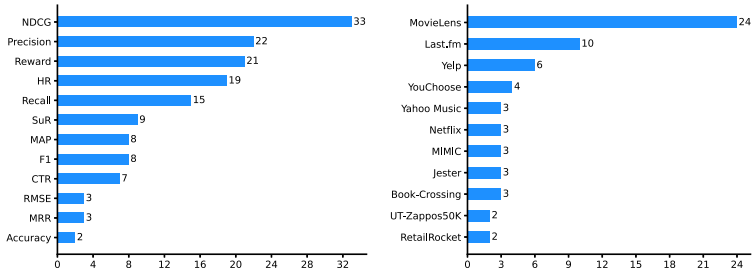
In imitation learning, the agent is trained to perform a task from demonstrations [202]. Zhang et al. [124] combine imagination (model-based RL) and imitation learning to recommend personalized search stories. They argue that the goal of imitation learning is to imitate the policy of a recommender agent from which the logging data has been collected. Also, some fictional sessions are imagined by the agent and saved in a separate memory, and are used to fine-tune the agent training.

5 OPEN RESEARCH DIRECTIONS

Slate Recommendation. RL algorithms have been originally developed to select a single action, e.g., the action with the highest Q value, in each time step from different actions around [203]. However, in the RS field, similar to many sequential decision support systems [203], it is wise to recommend a slate or list of items and let the user involved in the decision making process choose the best action, as the final goal is typically user satisfaction and recommendation acceptance. Despite some efforts [25, 40, 80, 112, 135], current RL algorithms cannot handle this problem. There are only two studies [80, 112] in the RLRS field that deeply investigate this problem. Slate-MDP [112] tries to solve this problem by searching the policy space for each slot in the slate individually. SlateQ [80] proposes to calculate the combination of the action set and consider each combination as an action. Slate-MDP cannot guarantee any optimality, and SlateQ is only applicable in two-stage RSs and fails to scale to single stage RSs with large action spaces. More attention is necessary in this aspect and more studies with solid theoretical foundations should be conducted in the future.

Explainability. Explainable recommendation is the ability of an RS to not only provide a recommendation, but also to address why a specific recommendation has been made [38]. Explanation about recommendations made could improve user experience, boost their trust in the system, and help them make better decisions [204–206]. Explainable methods could be generally divided into two groups: model-intrinsic or model-agnostic [207]. In the former, explanation is part of the recommendation process, while in the latter, the explanation is provided after the recommendation is made. An intrinsic explanation method could be the method we reviewed earlier [132]. On the other hand, as a model-agnostic example [208], RL is used to provide explanation for different recommendation methods. In particular, the method uses *couple agents*; one is responsible to generate explanations and another one predicts if the explanation generated is good enough for the user. One interesting application of explainable recommendation is in debugging the failed RS [208]. That is, through explanations provided, we can track the source of problems in our system and to see which parts are not working properly. Although there have been some efforts in RLRSs to provide explainable recommendations [132, 136, 155, 167, 169], there is still a lack in this aspect and more attention is required in the future.

Design. All RLRSs reviewed employ RL/DRL algorithms that have been originally developed in domains other than RSs, like games [62, 67, 74]. These methods are typically designed based on physics or Gaussian processes, not based on the complex and dynamic nature of a human. While sticking to available, cutting-edge RL algorithms and adapting them for RLRSs is wise, sometimes thinking out of the box could make a substantial improvement in the field. For example, instead of the usual MDP-based RL algorithms, Reference [209] uses *evolution strategies* [210] to optimize the recommendation policy, or Reference [211] borrows ideas from a different literature and adapts them to the recommendation problem. Relevant to this, as surveyed in [14], there are many deep learning models developed for RSs. Because deep learning and DRL are closely related, perhaps wisely combining these models with traditional RL algorithms could outperform existing DRL algorithms. Last but not least, as illustrated earlier, some RL algorithms like Q-learning have been more popular among RLRSs than other RL algorithms. Nonetheless, there is no clue or justification



(a) Popular metrics distribution (b) Popular datasets distribution

Fig. 11. Evaluation metrics and datasets used more often by RLRSs.

behind the use of a specific RL algorithm for an RS application. Therefore, this would be a great study to possibly find a relationship between the RL algorithm and the RS application.

Environment and Evaluation. Figure 11(a) depicts the most popular metrics by RLRSs. As shown, there is no metric specifically developed for RLRSs and almost all metrics are borrowed from the Information Retrieval field. Although RSs and Information Retrieval fields are very close, they are essentially different fields. Reward is also among the popular metrics used by RLRSs, which is a common metric in the RL literature. This analysis shows that there is a lack of metrics specifically designed for RLRSs. On the other hand, Figure 11(b) illustrates the most popular datasets used to evaluate RLRSs. MovieLens is the most popular dataset by far. A large number of datasets used for evaluation are not defined or public (see Tables 2 and 3). This limits the application and design of RLRSs to only a few applications, like entertainment. Therefore, it is important to collect and share more datasets in various domains to better evaluate RLRSs. Another aspect in RLRSs evaluation that needs further improvement in the future is to have a stronger and more unified simulation environment, something that can play the role of a benchmark in RLRSs' evaluation. As stated earlier, online evaluation is the natural method to evaluate RLRSs; however, it is difficult and costly to conduct a proper online study. On the other hand, offline environment, i.e., a dataset, is static and biased. Therefore, this clearly shows the importance of developing a strong, general-purpose simulator for RLRSs, something like OpenAI Gym [212] in the RL literature. Although some environment simulators have recently been developed for RLRSs [213–217], this trend should be continued and fortified.

Reproducibility. The effect of reproducibility on the advancement of a field is undeniable. For example, the field of image synthesis using GANs has seen astonishing results in a short period of time [218–220], and undoubtedly, an effective factor has been the common practice of sharing implementation codes, datasets, and research results. As illustrated in Tables 2 and 3, we cannot see this trend in the RLRS research community and only about 16% of researchers have shared their implementation codes. It would be helpful and can significantly accelerate the field's progress if researchers accurately present the value of important parameters and hyperparameters used in their experiments, to perform statistical significance testing for results presented, to disclose which random seeds have been used to repeat experiments, and to share their implementation codes and datasets (if datasets are not already public).

6 CONCLUSION

In this paper, we presented a comprehensive survey on state-of-the-art in RLRSs. We highlighted the important role of DRL in changing the research direction in the RLRS field, and accordingly, classified the algorithms into two general groups, i.e., RL- and DRL-based methods. Then, we

proposed a framework for RLRSs with four components, i.e., state representation, policy optimization, reward formulation, and environment building, and surveyed algorithms accordingly. Although many RLRSs have been proposed recently, we believe that the research on RLRSs is still in its infancy and needs plenty of advancements. Both RL and RSs are hot and ongoing research areas and are of specific interest to giant companies and businesses, so we can expect to witness new and exciting models to emerge each year. In the end, we hope this survey can assist researchers in understanding the key concepts and help advance the field in the future.

ACKNOWLEDGMENTS

We wish to thank the anonymous reviewers for their constructional comments on the first versions of this paper.

REFERENCES

[1] Cisco Visual Networking Index. 2013. The Zettabyte Era–Trends and Analysis. *Cisco White Paper* (2013).
[2] Dietmar Jannach, Markus Zanker, Alexander Felfernig, and Gerhard Friedrich. 2010. *Recommender Systems: An Introduction*. Cambridge University Press.
[3] Francesco Ricci, Lior Rokach, and Bracha Shapira. 2011. Introduction to recommender systems handbook. In *Recommender Systems Handbook*. 1–35.
[4] George Lekakos and Petros Caravelas. 2008. A hybrid approach for movie recommendation. *Multimedia Tools and Applications* 36, 1 (2008), 55–70.
[5] Hung-Chen Chen and Arbee L. P. Chen. 2001. A music recommendation system based on music data grouping and user interests. In *CIKM'01*. 231–238.
[6] Xuan Zhu, Yuan-Yuan Shi, Hyoung-Gook Kim, and Ki-Wan Eom. 2006. An integrated music recommendation system. *IEEE Transactions on Consumer Electronics* 52, 3 (2006), 917–925.
[7] J. Ben Schafer, Joseph Konstan, and John Riedl. 1999. Recommender systems in e-commerce. In *EC'99*. 158–166.
[8] Mozhgan Karimi, Dietmar Jannach, and Michael Jugovac. 2018. News recommender systems–survey and roads ahead. *Information Processing & Management* 54, 6 (2018), 1203–1227.
[9] Aleksandra Klašnja-Milićević, Mirjana Ivanović, and Alexandros Nanopoulos. 2015. Recommender systems in e-learning environments: A survey of the state-of-the-art and possible extensions. *Artificial Intelligence Review* 44, 4 (2015), 571–604.
[10] Emre Sezgin and Sevgi Özkan. 2013. A systematic literature review on health recommender systems. In *EHB'13*. 1–4.
[11] Netflix Update: Try This at Home. https://sifter.org/~simon/journal/20061211.html. ([n. d.]).
[12] Jesús Bobadilla, Fernando Ortega, Antonio Hernando, and Abraham Gutiérrez. 2013. Recommender systems survey. *Knowledge-based Systems* 46 (2013), 109–132.
[13] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. 2016. *Deep Learning*. MIT press Cambridge.
[14] Shuai Zhang, Lina Yao, Aixin Sun, and Yi Tay. 2019. Deep learning based recommender system: A survey and new perspectives. *ACM Computing Surveys (CSUR)* 52, 1 (2019), 1–38.
[15] Yuxi Li. 2017. Deep reinforcement learning: An overview. *arXiv preprint arXiv:1701.07274* (2017).
[16] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. 2018. Deep reinforcement learning that matters. In *AAAI'18*.
[17] Ahmad EL Sallab, Mohammed Abdou, Etienne Perot, and Senthil Yogamani. 2017. Deep reinforcement learning framework for autonomous driving. *Electronic Imaging* 2017, 19 (2017), 70–76.
[18] Changxi You, Jianbo Lu, Dimitar Filev, and Panagiotis Tsiotras. 2019. Advanced planning for autonomous vehicles using reinforcement learning and deep inverse reinforcement learning. *Robotics and Autonomous Systems* 114 (2019), 1–18.
[19] Jens Kober, J. Andrew Bagnell, and Jan Peters. 2013. Reinforcement learning in robotics: A survey. *International Journal of Robotics Research* 32, 11 (2013), 1238–1274.
[20] Richard Meyes, Hasan Tercan, Simon Roggendorf, Thomas Thiele, Christian Büscher, Markus Obdenbusch, Christian Brecher, Sabina Jeschke, and Tobias Meisen. 2017. Motion planning for industrial robots using reinforcement learning. *CIRP'17* 63 (2017), 107–112.
[21] Zhengyao Jiang, Dixing Xu, and Jinjun Liang. 2017. A deep reinforcement learning framework for the financial portfolio management problem. *arXiv preprint arXiv:1706.10059* (2017).
[22] Arthur Guez, Robert D. Vincent, Massimo Avoli, and Joelle Pineau. 2008. Adaptive treatment of epilepsy via batch-mode reinforcement learning. In *AAAI'08*. 1671–1678.

[23] Omer Gottesman, Fredrik Johansson, Matthieu Komorowski, Aldo Faisal, David Sontag, Finale Doshi-Velez, and Leo Anthony Celi. 2019. Guidelines for reinforcement learning in healthcare. *Nature Medicine* 25, 1 (2019), 16–18.
[24] Gabriel Dulac-Arnold, Richard Evans, Hado van Hasselt, Peter Sunehag, Timothy Lillicrap, Jonathan Hunt, Timothy Mann, Theophane Weber, Thomas Degris, and Ben Coppin. 2015. Deep reinforcement learning in large discrete action spaces. *arXiv preprint arXiv:1512.07679* (2015).
[25] Minmin Chen, Alex Beutel, Paul Covington, Sagar Jain, Francois Belletti, and Ed H. Chi. 2019. Top-k off-policy correction for a REINFORCE recommender system. In *WSDM'19*. 456–464.
[26] Aleksandrs Slivkins. 2019. Introduction to multi-armed bandits. *arXiv preprint arXiv:1904.07272* (2019).
[27] Richard S. Sutton and Andrew G. Barto. 2018. *Reinforcement Learning: An Introduction*. MIT press.
[28] Lihong Li, Wei Chu, John Langford, and Robert E. Schapire. 2010. A contextual-bandit approach to personalized news article recommendation. In *WWW'10*. 661–670.
[29] Shuai Li, Alexandros Karatzoglou, and Claudio Gentile. 2016. Collaborative filtering bandits. In *SIGIR'16*. 539–548.
[30] Charu C. Aggarwal. 2016. *Recommender Systems*. Springer.
[31] M. Mehdi Afsar, Trafford Crump, and Behrouz Far. 2021. An exploration on-demand article recommender system for cancer patients information provisioning. In *FLAIRS'21*, Vol. 34.
[32] Gangan Elena, Kudus Milos, and Ilyushin Eugene. 2021. Survey of multiarmed bandit algorithms applied to recommendation systems. *International Journal of Open Information Technologies* 9, 4 (2021), 12–27.
[33] Xiaoyuan Su and Taghi M. Khoshgoftaar. 2009. A survey of collaborative filtering techniques. *Advances in Artificial Intelligence* (2009).
[34] Yue Shi, Martha Larson, and Alan Hanjalic. 2014. Collaborative filtering beyond the user-item matrix: A survey of the state of the art and future challenges. *ACM Computing Surveys (CSUR)* 47, 1 (2014), 1–45.
[35] Robin Burke. 2002. Hybrid recommender systems: Survey and experiments. *User Modeling and User-adapted Interaction* 12, 4 (2002), 331–370.
[36] Feng Xia, Nana Yaw Asabere, Ahmedin Mohammed Ahmed, Jing Li, and Xiangjie Kong. 2013. Mobile multimedia recommendation in smart communities: A survey. *IEEE Access* 1 (2013), 606–624.
[37] Yashar Deldjoo, Markus Schedl, Paolo Cremonesi, and Gabriella Pasi. 2020. Recommender systems leveraging multimedia content. *ACM Computing Surveys (CSUR)* 53, 5 (2020), 1–38.
[38] Yongfeng Zhang and Xu Chen. 2018. Explainable recommendation: A survey and new perspectives. *arXiv preprint arXiv:1804.11192* (2018).
[39] Joeran Beel, Bela Gipp, Stefan Langer, and Corinna Breitinger. 2016. Paper recommender systems: A literature survey. *International Journal on Digital Libraries* 17, 4 (2016), 305–338.
[40] Xiangyu Zhao, Long Xia, Jiliang Tang, and Dawei Yin. 2019. Deep reinforcement learning for search, recommendation, and online advertising: A survey. *ACM SIGWEB Newsletter* (2019), 1–15.
[41] Massimo Quadrana, Paolo Cremonesi, and Dietmar Jannach. 2018. Sequence-aware recommender systems. *ACM Computing Surveys (CSUR)* 51, 4 (2018), 1–36.
[42] Shoujin Wang, Longbing Cao, Yan Wang, Quan Z. Sheng, Mehmet A. Orgun, and Defu Lian. 2021. A survey on session-based recommender systems. *ACM Computing Surveys (CSUR)* 54, 7 (2021), 1–38.
[43] Shi-Yong Chen, Yang Yu, Qing Da, Jun Tan, Hai-Kuan Huang, and Hai-Hong Tang. 2018. Stabilizing reinforcement learning in dynamic environment with application to online recommendation. In *SIGKDD'18*. 1187–1196.
[44] Sungwoon Choi, Heonseok Ha, Uiwon Hwang, Chanju Kim, Jung-Woo Ha, and Sungroh Yoon. 2018. Reinforcement learning based recommender system using biclustering technique. *arXiv preprint arXiv:1801.05532* (2018).
[45] Isshu Munemasa, Yuta Tomomatsu, Kunioki Hayashi, and Tomohiro Takagi. 2018. Deep reinforcement learning for recommender systems. In *ICOIACT'18*. 226–233.
[46] 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.
[47] Guanjie Zheng, Fuzheng Zhang, Zihan Zheng, Yang Xiang, Nicholas Jing Yuan, Xing Xie, and Zhenhui Li. 2018. DRN: A deep reinforcement learning framework for news recommendation. In *WWW'18*. 167–176.
[48] Xiangyu Zhao, Liang Zhang, Zhuoye Ding, Long Xia, Jiliang Tang, and Dawei Yin. 2018. Recommendations with negative feedback via pairwise deep reinforcement learning. In *SIGKDD'18*. 1040–1048.
[49] Omar Moling, Linas Baltrunas, and Francesco Ricci. 2012. Optimal radio channel recommendations with explicit and implicit feedback. In *RecSys*. 75–82.
[50] Guy Shani, David Heckerman, and Ronen I. Brafman. 2005. An MDP-based recommender system. *Journal of Machine Learning Research* 6 (2005), 1265–1295.
[51] Binbin Hu, Chuan Shi, and Jian Liu. 2017. Playlist recommendation based on reinforcement learning. In *ICIS'17*. 172–182.
[52] Xiangyu Zhao, Liang Zhang, Long Xia, Zhuoye Ding, Dawei Yin, and Jiliang Tang. 2017. Deep reinforcement learning for list-wise recommendations. *arXiv preprint arXiv:1801.00209* (2017).

[53] Paul Resnick and Hal R. Varian. 1997. Recommender systems. *Commun. ACM* 40, 3 (1997), 56–58.
[54] David Goldberg, David Nichols, Brian M. Oki, and Douglas Terry. 1992. Using collaborative filtering to weave an information tapestry. *Commun. ACM* 35, 12 (1992), 61–70.
[55] Michael J. Pazzani and Daniel Billsus. 2007. Content-based recommendation systems. In *The Adaptive Web*. Springer, 325–341.
[56] Pasquale Lops, Marco De Gemmis, and Giovanni Semeraro. 2011. Content-based recommender systems: State of the art and trends. In *Recommender Systems Handbook*. 73–105.
[57] Christopher Watkins. 1989. *Learning from Delayed Rewards*. University of Cambridge.
[58] Gavin A. Rummery and Mahesan Niranjan. 1994. *On-line Q-learning Using Connectionist Systems*. Vol. 37. University of Cambridge.
[59] Anton Schwartz. 1993. A reinforcement learning method for maximizing undiscounted rewards. In *ICML'93*. 298–305.
[60] Cameron B. Browne, Edward Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. 2012. A survey of Monte Carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games* 4, 1 (2012), 1–43.
[61] Guillaume Chaslot, Sander Bakkes, Istvan Szita, and Pieter Spronck. 2008. Monte-Carlo tree search: A new framework for game AI. *AIIDE* 8 (2008), 216–217.
[62] David Silver et al. 2016. Mastering the game of go with deep neural networks and tree search. *Nature* 529, 7587 (2016), 484–489.
[63] Geoffrey J. Gordon. 1999. *Approximate Solutions to Markov Decision Processes*. Carnegie Mellon University.
[64] Damien Ernst, Pierre Geurts, and Louis Wehenkel. 2005. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research* 6 (2005), 503–556.
[65] Ronald J. Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning* 8, 3-4 (1992), 229–256.
[66] Andrew G. Barto, Richard S. Sutton, and Charles W. Anderson. 1983. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics* 5 (1983), 834–846.
[67] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing Atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* (2013).
[68] Volodymyr Mnih et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529–533.
[69] Long-Ji Lin. 1992. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning* 8, 3-4 (1992), 293–321.
[70] Sebastian Thrun and Anton Schwartz. 1993. Issues in using function approximation for reinforcement learning. In *Connectionist Models Summer School Hillsdale*.
[71] Hado Van Hasselt, Arthur Guez, and David Silver. 2015. Deep reinforcement learning with double Q-learning. *arXiv preprint arXiv:1509.06461* (2015).
[72] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. 2016. Dueling network architectures for deep reinforcement learning. In *ICML'16*. 1995–2003.
[73] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. 2015. Prioritized experience replay. *arXiv preprint arXiv:1511.05952* (2015).
[74] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2015. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971* (2015).
[75] David Silver et al. 2014. Deterministic policy gradient algorithms. In *ICML'14*.
[76] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).
[77] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. 2015. Trust region policy optimization. In *ICML'15*. 1889–1897.
[78] Mohammad Mehdi Afsar, Trafford Crump, and Behrouz Far. 2022. Sample efficiency in deep reinforcement learning based recommender systems with imitation learning. In *Canadian Conference on Artificial Intelligence*.
[79] A. Zimdars, D. M. Chickering, and C. Meek. 2001. Using temporal data for making recommendations. In *UAI'01*. 580–588.
[80] Eugene Ie et al. 2019. Reinforcement learning for slate-based recommender systems: A tractable decomposition and practical methodology. *arXiv preprint arXiv:1905.12767* (2019).
[81] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555* (2014).
[82] Joeran Beel and Stefan Langer. 2015. A comparison of offline evaluations, online evaluations, and user studies in the context of research-paper recommender systems. In *TPDL'15*. 153–168.

[83] Thorsten Joachims, Dayne Freitag, Tom Mitchell, et al. 1997. WebWatcher: A tour guide for the World Wide Web. In *IJCAI'97*. 770–777.
[84] Mircea Preda and Dan Popescu. 2005. Personalized web recommendations: Supporting epistemic information about end-users. In *WI'05*. 692–695.
[85] Nima Taghipour, Ahmad Kardan, and Saeed Shiry Ghidary. 2007. Usage-based web recommendations: A reinforcement learning approach. In *RecSys'07*. 113–120.
[86] Tariq Mahmood and Francesco Ricci. 2007. Learning and adaptivity in interactive recommender systems. In *EC'07*. 75–84.
[87] Nima Taghipour and Ahmad Kardan. 2008. A hybrid web recommender system based on Q-learning. In *SAC'08*. 1164–1168.
[88] Tariq Mahmood and Francesco Ricci. 2009. Improving recommender systems with adaptive conversational strategies. In *HT'09*. 73–82.
[89] Chung-Yi Chi, Richard Tzong-Han Tsai, Jeng-You Lai, and Jane Yung-jen Hsu. 2010. A reinforcement learning approach to emotion-based automatic playlist generation. In *TAAI'10*. 60–65.
[90] Susan M. Shortreed, Eric Laber, Daniel J. Lizotte, T. Scott Stroup, Joelle Pineau, and Susan A. Murphy. 2011. Informing sequential clinical decision-making through reinforcement learning: An empirical study. *Machine Learning* 84, 1-2 (2011), 109–136.
[91] Yufan Zhao, Donglin Zeng, Mark A. Socinski, and Michael R. Kosorok. 2011. Reinforcement learning strategies for clinical trials in nonsmall cell lung cancer. *Biometrics* 67, 4 (2011), 1422–1433.
[92] Tariq Mahmood, Ghulam Mujtaba, and Adriano Venturini. 2014. Dynamic personalization in conversational recommender systems. *Information Systems and E-Business Management* 12, 2 (2014), 213–238.
[93] Elad Liebman, Maytal Saar-Tsechansky, and Peter Stone. 2014. DJ-MC: A reinforcement-learning agent for music playlist recommendation. *arXiv preprint arXiv:1401.1880* (2014).
[94] Georgios Theocharous, Philip S. Thomas, and Mohammad Ghavamzadeh. 2015. Personalized ad recommendation systems for life-time value optimization with guarantees. In *IJCAI'15*.
[95] Zhongqi Lu and Qiang Yang. 2016. Partially observable Markov decision process for recommender systems. *arXiv preprint arXiv:1608.07793* (2016).
[96] Yang Zhang, Chenwei Zhang, and Xiaozhong Liu. 2017. Dynamic scholarly collaborator recommendation via competitive multi-agent reinforcement learning. In *RecSys'17*. 331–335.
[97] Wacharawan Intayoad, Chayapol Kamyod, and Punnarumol Temdee. 2018. Reinforcement learning for online learning recommendation system. In *GWS*. 167–170.
[98] Jia-Wei Chang, Ching-Yi Chiou, Jia-Yi Liao, Ying-Kai Hung, Chien-Che Huang, Kuan-Cheng Lin, and Ying-Hung Pu. 2019. Music recommender using deep embedding-based features and behavior-based reinforcement learning. *Multimedia Tools and Applications* (2019), 1–28.
[99] Jing Chen and Wenjun Jiang. 2019. Context-aware personalized POI sequence recommendation. In *iSCI'19*. Springer, 197–210.
[100] Yu Wang. 2020. A hybrid recommendation for music based on reinforcement learning. In *PAKDD'20*. 91–103.
[101] Marios Kokkodis and Panagiotis G. Ipeirotis. 2021. Demand-aware career path recommendations: A reinforcement learning approach. *Management Science* 67, 7 (2021), 4362–4383.
[102] Bamshad Mobasher, Robert Cooley, and Jaideep Srivastava. 2000. Automatic personalization based on web usage mining. *Commun. ACM* 43, 8 (2000), 142–151.
[103] Andriy Mnih and Russ R. Salakhutdinov. 2008. Probabilistic matrix factorization. In *NIPS'08*. 1257–1264.
[104] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S. Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *NIPS'13*. 3111–3119.
[105] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. 2016. WaveNet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499* (2016).
[106] Hui Li, Tsz Nam Chan, Man Lung Yiu, and Nikos Mamoulis. 2017. FEXIPRO: Fast and exact inner product retrieval in recommender systems. In *SIGMOD'17*. 835–850.
[107] Richard E. Bellman. 2015. *Adaptive Control Processes*. Princeton University Press.
[108] Andrew G. Barto. 1995. Reinforcement learning and dynamic programming. In *Analysis, Design and Evaluation of Man-Machine Systems*. 407–412.
[109] Levente Kocsis and Csaba Szepesvári. 2006. Bandit based Monte-Carlo planning. In *ECML'06*. 282–293.
[110] MovieLens. https://grouplens.org/datasets/movielens/. ([n. d.]).
[111] Thierry Bertin-Mahieux, Daniel P. W. Ellis, Brian Whitman, and Paul Lamere. 2011. The million song dataset. In *ISMIR'11*. 591–596.

[112] Peter Sunehag, Richard Evans, Gabriel Dulac-Arnold, Yori Zwols, Daniel Visentin, and Ben Coppin. 2015. Deep reinforcement learning with attention for slate Markov decision processes with high-dimensional states and actions. *arXiv preprint arXiv:1512.01124* (2015).
[113] Shamim Nemati, Mohammad M. Ghassemi, and Gari D. Clifford. 2016. Optimal medication dosing from suboptimal clinical examples: A deep reinforcement learning approach. In *EMBC'16*. 2978–2981.
[114] Claudio Greco, Alessandro Suglia, Pierpaolo Basile, and Giovanni Semeraro. 2017. Converse-Et-Impera: Exploiting deep learning and hierarchical reinforcement learning for conversational recommender systems. In *AIxIA'17*. 372–386.
[115] Aniruddh Raghu, Matthieu Komorowski, Imran Ahmed, Leo Celi, Peter Szolovits, and Marzyeh Ghassemi. 2017. Deep reinforcement learning for sepsis treatment. *arXiv preprint arXiv:1711.09602* (2017).
[116] Lu Wang, Wei Zhang, Xiaofeng He, and Hongyuan Zha. 2018. Supervised reinforcement learning with recurrent neural network for dynamic treatment recommendation. In *SIGKDD'18*. 2447–2456.
[117] Yueming Sun and Yi Zhang. 2018. Conversational recommender system. In *SIGIR'18*. 235–244.
[118] Chenfei Zhao and Lan Hu. 2019. CapDRL: A deep capsule reinforcement learning for movie recommendation. In *PRICAI'19*. Springer, 734–739.
[119] Lixin Zou, Long Xia, Zhuoye Ding, Jiaxing Song, Weidong Liu, and Dawei Yin. 2019. Reinforcement learning to optimize long-term user engagement in recommender systems. In *SIGKDD'19*. 2810–2818.
[120] Rong Gao, Haifeng Xia, Jing Li, Donghua Liu, Shuai Chen, and Gang Chun. 2019. DRCGR: Deep reinforcement learning framework incorporating CNN and GAN-based for interactive recommendation. In *ICDM'19*. 1048–1053.
[121] Tong Yu, Yilin Shen, Ruiyi Zhang, Xiangyu Zeng, and Hongxia Jin. 2019. Vision-language recommendation via attribute augmented multimodal reinforcement learning. In *MM'19*. 39–47.
[122] Daisuke Tsumita and Tomohiro Takagi. 2019. Dialogue based recommender system that flexibly mixes utterances and recommendations. In *WI'19*. 51–58.
[123] Jing Zhang, Bowen Hao, Bo Chen, Cuiping Li, Hong Chen, and Jimeng Sun. 2019. Hierarchical reinforcement learning for course recommendation in MOOCs. In *AAAI'19*. 435–442.
[124] Jason Zhang, Junming Yin, Dongwon Lee, and Linhong Zhu. 2019. Deep reinforcement learning for personalized search story recommendation. *Journal of Environmental Sciences* (2019).
[125] Dong Liu and Chenyang Yang. 2019. A deep reinforcement learning approach to proactive content pushing and recommendation for mobile users. *IEEE Access* 7 (2019), 83120–83136.
[126] Zhang Yuyan, Su Xiayao, and Liu Yong. 2019. A novel movie recommendation system based on deep reinforcement learning with prioritized experience replay. In *ICCT'19*. 1496–1500.
[127] Tao Gui, Peng Liu, Qi Zhang, Liang Zhu, Minlong Peng, Yunhua Zhou, and Xuanjing Huang. 2019. Mention recommendation in Twitter with cooperative multi-agent reinforcement learning. In *SIGIR'19*. 535–544.
[128] Ruiyi Zhang, Tong Yu, Yilin Shen, Hongxia Jin, and Changyou Chen. 2019. Text-based interactive recommendation via constraint-augmented reinforcement learning. In *NIPS'19*.
[129] Yu Lei and Wenjie Li. 2019. Interactive recommendation with user-specific deep reinforcement learning. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 13, 6 (2019), 1–15.
[130] Xueying Bai, Jian Guan, and Hongning Wang. 2019. Model-based reinforcement learning with adversarial training for online recommendation. *arXiv preprint arXiv:1911.03845* (2019).
[131] Floris Den Hengst, Mark Hoogendoorn, Frank Van Harmelen, and Joost Bosman. 2019. Reinforcement learning for personalized dialogue management. In *WI'19*. 59–67.
[132] Yikun Xian, Zuohui Fu, S. Muthukrishnan, Gerard De Melo, and Yongfeng Zhang. 2019. Reinforcement knowledge graph reasoning for explainable recommendation. In *SIGIR'19*. 285–294.
[133] Haokun Chen, Xinyi Dai, Han Cai, Weinan Zhang, Xuejian Wang, Ruiming Tang, Yuzhou Zhang, and Yong Yu. 2019. Large-scale interactive recommendation with tree-structured policy gradient. In *AAAI'19*, Vol. 33. 3312–3320.
[134] Lixin Zou, Long Xia, Zhuoye Ding, Dawei Yin, Jiaxing Song, and Weidong Liu. 2019. Reinforcement learning to diversify top-n recommendation. In *DASFAA'19*. 104–120.
[135] Xinshi Chen, Shuang Li, Hui Li, Shaohua Jiang, Yuan Qi, and Le Song. 2019. Generative adversarial user model for reinforcement learning based recommendation system. In *ICML'19*. 1052–1061.
[136] Weiping Song, Zhijian Duan, Ziqing Yang, Hao Zhu, Ming Zhang, and Jian Tang. 2019. Explainable knowledge graph-based recommendation via deep reinforcement learning. *arXiv preprint arXiv:1906.09506* (2019).
[137] Lixin Zou, Long Xia, Pan Du, Zhuo Zhang, Ting Bai, Weidong Liu, Jian-Yun Nie, and Dawei Yin. 2020. Pseudo Dyna-Q: A reinforcement learning framework for interactive recommendation. In *WSDM'20*. 816–824.
[138] Yu Lei, Zhitao Wang, Wenjie Li, Hongbin Pei, and Quanyu Dai. 2020. Social attentive deep Q-networks for recommender systems. *IEEE Transactions on Knowledge and Data Engineering* (2020).
[139] Xiangyu Zhao, Xudong Zheng, Xiwang Yang, Xiaobing Liu, and Jiliang Tang. 2020. Jointly learning to recommend and advertise. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 3319–3327.

[140] Shenggong Ji, Zhaoyuan Wang, Tianrui Li, and Yu Zheng. 2020. Spatio-temporal feature fusion for dynamic taxi route recommendation via deep reinforcement learning. *Knowledge-Based Systems* 205 (2020), 106302.
[141] Peter Wei, Stephen Xia, Runfeng Chen, Jingyi Qian, Chong Li, and Xiaofan Jiang. 2020. A deep reinforcement learning based recommender system for occupant-driven energy optimization in commercial buildings. *IEEE Internet of Things Journal* (2020).
[142] Yu Lei, Hongbin Pei, Hanqi Yan, and Wenjie Li. 2020. Reinforcement learning based recommendation with graph convolutional Q-network. In *SIGIR'20*. 1757–1760.
[143] Dongyang Zhao, Liang Zhang, Bo Zhang, Lizhou Zheng, Yongjun Bao, and Weipeng Yan. 2020. MaHRL: Multi-goals abstraction based deep hierarchical reinforcement learning for recommendations. In *SIGIR'20*. 871–880.
[144] Feng Liu, Ruiming Tang, Xutao Li, Weinan Zhang, Yunming Ye, Haokun Chen, Huifeng Guo, and Yuzhou Zhang. 2018. Deep reinforcement learning based recommendation with explicit user-item interactions modeling. *arXiv preprint arXiv:1810.12027* (2018).
[145] Feng Liu, Ruiming Tang, Xutao Li, Weinan Zhang, Yunming Ye, Haokun Chen, Huifeng Guo, Yuzhou Zhang, and Xiuqiang He. 2020. State representation modeling for deep reinforcement learning based recommendation. *Knowledge-Based Systems* 205 (2020), 106170.
[146] Weiwen Liu, Feng Liu, Ruiming Tang, Ben Liao, Guangyong Chen, and Pheng Ann Heng. 2020. Balancing between accuracy and fairness for interactive recommendation with reinforcement learning. *Advances in Knowledge Discovery and Data Mining* 12084 (2020), 155.
[147] 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.
[148] Xu He, Bo An, Yanghua Li, Haikai Chen, Rundong Wang, Xinrun Wang, Runsheng Yu, Xin Li, and Zhirong Wang. 2020. Learning to collaborate in multi-module recommendation via multi-agent reinforcement learning without communication. In *RecSys'20*. 210–219.
[149] Xiangyu Zhao, Long Xia, Lixin Zou, Hui Liu, Dawei Yin, and Jiliang Tang. 2020. Whole-chain recommendations. In *CIKM'20*. 1883–1891.
[150] Xiang Wang, Yaokun Xu, Xiangnan He, Yixin Cao, Meng Wang, and Tat-Seng Chua. 2020. Reinforced negative sampling over knowledge graph for recommendation. In *WWW'20*. 99–109.
[151] Xiaocong Chen, Chaoran Huang, Lina Yao, Xianzhi Wang, Wenjie Zhang, et al. 2020. Knowledge-guided deep reinforcement learning for interactive recommendation. In *IJCNN'20*. 1–8.
[152] Xuhui Ren, Hongzhi Yin, Tong Chen, Hao Wang, Nguyen Quoc Viet Hung, Zi Huang, and Xiangliang Zhang. 2020. CRSAL: Conversational recommender systems with adversarial learning. *ACM Transactions on Information Systems (TOIS)* 38, 4 (2020), 1–40.
[153] Wenqiang Lei, Gangyi Zhang, Xiangnan He, Yisong Miao, Xiang Wang, Liang Chen, and Tat-Seng Chua. 2020. Interactive path reasoning on graph for conversational recommendation. In *SIGKDD'20*. 2073–2083.
[154] Xin Xin, Alexandros Karatzoglou, Ioannis Arapakis, and Joemon M. Jose. 2020. Self-supervised reinforcement learning for recommender systems. In *SIGIR'20*. 931–940.
[155] Kangzhi Zhao, Xiting Wang, Yuren Zhang, Li Zhao, Zheng Liu, Chunxiao Xing, and Xing Xie. 2020. Leveraging demonstrations for reinforcement recommendation reasoning over knowledge graphs. In *SIGIR'20*. 239–248.
[156] Pengfei Wang, Yu Fan, Long Xia, Wayne Xin Zhao, ShaoZhang Niu, and Jimmy Huang. 2020. KERL: A knowledge-guided reinforcement learning model for sequential recommendation. In *SIGIR'20*. 209–218.
[157] Huizhi Liang. 2020. DRprofiling: Deep reinforcement user profiling for recommendations in heterogenous information networks. *IEEE Transactions on Knowledge and Data Engineering* (2020).
[158] Sijin Zhou, Xinyi Dai, Haokun Chen, Weinan Zhang, Kan Ren, Ruiming Tang, Xiuqiang He, and Yong Yu. 2020. Interactive recommender system via knowledge graph-enhanced reinforcement learning. In *SIGIR'20*. 179–188.
[159] Ashudeep Singh, Yoni Halpern, Nithum Thain, Konstantina Christakopoulou, EH Chi, Jilin Chen, and Alex Beutel. 2020. Building healthy recommendation sequences for everyone: A safe reinforcement learning approach. In *FAccTRec Workshop*.
[160] Feng Liu, Huifeng Guo, Xutao Li, Ruiming Tang, Yunming Ye, and Xiuqiang He. 2020. End-to-end deep reinforcement learning based recommendation with supervised embedding. In *WSDM'20*. 384–392.
[161] Feng Liu, Ruiming Tang, Huifeng Guo, Xutao Li, Yunming Ye, and Xiuqiang He. 2020. Top-aware reinforcement learning based recommendation. *Neurocomputing* 417 (2020), 255–269.
[162] Vahid Baghi, Seyed Mohammad Seyed Motehayeri, Ali Moeini, and Rooholah Abedian. 2021. Improving ranking function and diversification in interactive recommendation systems based on deep reinforcement learning. In *CSICC'21*. 1–7.
[163] Zefang Liu, Shuran Wen, and Yinzhu Quan. 2021. Deep reinforcement learning based group recommender system. *arXiv preprint arXiv:2106.06900* (2021).

[164] Yingqiang Ge, Shuchang Liu, Ruoyuan Gao, Yikun Xian, Yunqi Li, Xiangyu Zhao, Changhua Pei, Fei Sun, Junfeng Ge, Wenwu Ou, et al. 2021. Towards long-term fairness in recommendation. In *WSDM'21*. 445–453.
[165] Mingsheng Fu, Anubha Agrawal, Athirai A. Irissappane, Jie Zhang, Liwei Huang, and Hong Qu. 2021. Deep reinforcement learning framework for category-based item recommendation. *IEEE Transactions on Cybernetics* (2021).
[166] Yuanguo Lin, Shibo Feng, Fan Lin, Wenhua Zeng, Yong Liu, and Pengcheng Wu. 2021. Adaptive course recommendation in MOOCs. *Knowledge-Based Systems* 224 (2021), 107085.
[167] Shaohua Tao, Runhe Qiu, Yuan Ping, and Hui Ma. 2021. Multi-modal knowledge-aware reinforcement learning network for explainable recommendation. *Knowledge-Based Systems* (2021), 107217.
[168] Weijia Zhang, Hao Liu, Fan Wang, Tong Xu, Haoran Xin, Dejing Dou, and Hui Xiong. 2021. Intelligent electric vehicle charging recommendation based on multi-agent reinforcement learning. In *WWW'21*. 1856–1867.
[169] Danyang Liu, Jianxun Lian, Zheng Liu, Xiting Wang, Guangzhong Sun, and Xing Xie. 2021. Reinforced anchor knowledge graph generation for news recommendation reasoning. In *SIGKDD'21*. 1055–1065.
[170] Yang Deng, Yaliang Li, Fei Sun, Bolin Ding, and Wai Lam. 2021. Unified conversational recommendation policy learning via graph-based reinforcement learning. *arXiv preprint arXiv:2105.09710* (2021).
[171] Ruobing Xie, Shaoliang Zhang, Rui Wang, Feng Xia, and Leyu Lin. 2021. Hierarchical reinforcement learning for integrated recommendation. In *AAAI'21*.
[172] Kai Wang, Zhene Zou, Qilin Deng, Runze Wu, Jianrong Tao, Changjie Fan, Liang Chen, and Peng Cui. 2021. Reinforcement learning with a disentangled universal value function for item recommendation. *arXiv preprint arXiv:2104.02981* (2021).
[173] Xiangyu Zhao, Changsheng Gu, Haoshenglun Zhang, Xiwang Yang, Xiaobing Liu, Hui Liu, and Jiliang Tang. 2021. DEAR: Deep reinforcement learning for online advertising impression in recommender systems. In *AAAI'21*. 750–758.
[174] Chengqian Gao, Ke Xu, and Peilin Zhao. 2021. Value penalized Q-learning for recommender systems. *arXiv preprint arXiv:2110.07923* (2021).
[175] Teng Xiao and Donglin Wang. 2021. A general offline reinforcement learning framework for interactive recommendation. In *AAAI'21*.
[176] Minmin Chen, Bo Chang, Can Xu, and Ed H. Chi. 2021. User response models to improve a REINFORCE recommender system. In *WSDM'21*. 121–129.
[177] Dankit K. Nassiuma. 2001. Survey sampling: Theory and methods. (2001).
[178] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. 2017. Hindsight experience replay. *arXiv preprint arXiv:1707.01495* (2017).
[179] Yisong Yue and Thorsten Joachims. 2009. Interactively optimizing information retrieval systems as a dueling bandits problem. In *ICML'09*. 1201–1208.
[180] Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. 2017. SeqGAN: Sequence generative adversarial nets with policy gradient. In *AAAI'17*.
[181] George E. Uhlenbeck and Leonard S. Ornstein. 1930. On the theory of the Brownian motion. *Physical Review* 36, 5 (1930), 823.
[182] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. 2018. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *ICML'18*. 1861–1870.
[183] Jakob N. Foerster, Yannis M. Assael, Nando De Freitas, and Shimon Whiteson. 2016. Learning to communicate with deep multi-agent reinforcement learning. *arXiv preprint arXiv:1605.06676* (2016).
[184] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. 2017. Multi-agent actor-critic for mixed cooperative-competitive environments. *arXiv preprint arXiv:1706.02275* (2017).
[185] Jonathan Ho and Stefano Ermon. 2016. Generative adversarial imitation learning. *NIPS'16* 29 (2016), 4565–4573.
[186] David G. Kleinbaum and Mitchel Klein. 2010. *Survival Analysis*. Springer.
[187] How Jing and Alexander J. Smola. 2017. Neural survival recommender. In *WSDM'17*. 515–524.
[188] Eugene Ie et al. 2019. RecSim: A configurable simulation platform for recommender systems. *arXiv preprint arXiv:1909.04847* (2019).
[189] Edward L. Ionides. 2008. Truncated importance sampling. *Journal of Computational and Graphical Statistics* 17, 2 (2008), 295–311.
[190] Yelp. https://www.yelp.com/academic_dataset. ([n. d.]).
[191] Lloyd S. Shapley. 1953. Stochastic games. *PNAS'53* 39, 10 (1953), 1095–1100.
[192] Kaiqing Zhang, Zhuoran Yang, and Tamer Başar. 2021. Multi-agent reinforcement learning: A selective overview of theories and algorithms. *Handbook of Reinforcement Learning and Control* (2021), 321–384.
[193] Robert J. Aumann. 1974. Subjectivity and correlation in randomized strategies. *Journal of Mathematical Economics* 1, 1 (1974), 67–96.

[194] Andrew G. Barto and Sridhar Mahadevan. 2003. Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems* 13, 1 (2003), 41–77.
[195] Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. 2017. Feudal networks for hierarchical reinforcement learning. In *ICML'17*. 3540–3549.
[196] Ofir Nachum, Shixiang Gu, Honglak Lee, and Sergey Levine. 2018. Data-efficient hierarchical reinforcement learning. *arXiv preprint arXiv:1805.08296* (2018).
[197] Tejas D. Kulkarni, Karthik Narasimhan, Ardavan Saeedi, and Josh Tenenbaum. 2016. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. *NIPS'16* 29 (2016), 3675–3683.
[198] Richard S. Sutton, Doina Precup, and Satinder Singh. 1999. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence* 112, 1–2 (1999), 181–211.
[199] Steffen Rendle. 2010. Factorization machines. In *ICDM'10*. 995–1000.
[200] Michael T. Rosenstein, Andrew G. Barto, Jennie Si, Andy Barto, Warren Powell, and Donald Wunsch. 2004. Supervised actor-critic reinforcement learning. *Learning and Approximate Dynamic Programming: Scaling Up to the Real World* (2004), 359–380.
[201] Javier Garcıa and Fernando Fernández. 2015. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research* 16, 1 (2015), 1437–1480.
[202] Ahmed Hussein, Mohamed Medhat Gaber, Eyad Elyan, and Chrisina Jayne. 2017. Imitation learning: A survey of learning methods. *ACM Computing Surveys (CSUR)* 50, 2 (2017), 1–35.
[203] M. Milani Fard and Joelle Pineau. 2011. Non-deterministic policies in Markovian decision processes. *Journal of Artificial Intelligence Research* 40 (2011), 1–24.
[204] Dan Cosley, Shyong K. Lam, Istvan Albert, Joseph A. Konstan, and John Riedl. 2003. Is seeing believing? How recommender system interfaces affect users' opinions. In *SIGCHI'03*. 585–592.
[205] Li Chen and Pearl Pu. 2005. Trust building in recommender agents. In *ICETE'05*. 135–145.
[206] Nava Tintarev and Judith Masthoff. 2007. Effective explanations of recommendations: User-centered design. In *RecSys'07*. 153–156.
[207] Zachary C. Lipton. 2018. The mythos of model interpretability. *Queue* 16, 3 (2018), 31–57.
[208] Xiting Wang, Yiru Chen, Jie Yang, Le Wu, Zhengtao Wu, and Xing Xie. 2018. A reinforcement learning framework for explainable recommendation. In *ICDM'18*. 587–596.
[209] Changhua Pei, Xinru Yang, Qing Cui, Xiao Lin, Fei Sun, Peng Jiang, Wenwu Ou, and Yongfeng Zhang. 2019. Value-aware recommendation based on reinforcement profit maximization. In *WWW'19*. 3123–3129.
[210] Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. 2017. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864* (2017).
[211] Mehdi Afsar, Trafford Crump, and Behrouz Far. 2021. A load balanced recommendation approach. *arXiv preprint arXiv:2105.09981* (2021).
[212] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. OpenAI Gym. *arXiv preprint arXiv:1606.01540* (2016).
[213] Jason Gauci, Edoardo Conti, Yitao Liang, Kittipat Virochsiri, Yuchen He, Zachary Kaden, Vivek Narayanan, Xiaohui Ye, Zhengxing Chen, and Scott Fujimoto. 2018. Horizon: Facebook's open source applied reinforcement learning platform. *arXiv preprint arXiv:1811.00260* (2018).
[214] David Rohde, Stephen Bonner, Travis Dunlop, Flavian Vasile, and Alexandros Karatzoglou. 2018. RecoGym: A reinforcement learning environment for the problem of product recommendation in online advertising. *arXiv preprint arXiv:1808.00720* (2018).
[215] Xiangyu Zhao, Long Xia, Zhuoye Ding, Dawei Yin, and Jiliang Tang. 2019. Toward simulating environments in reinforcement learning based recommendations. *arXiv preprint arXiv:1906.11462* (2019).
[216] Bichen Shi, Makbule Gulcin Ozsoy, Neil Hurley, Barry Smyth, Elias Z. Tragos, James Geraci, and Aonghus Lawlor. 2019. PyRecGym: A reinforcement learning gym for recommender systems. In *RecSys'19*. 491–495.
[217] Jin Huang, Harrie Oosterhuis, Maarten de Rijke, and Herke van Hoof. 2020. Keeping dataset biases out of the simulation: A debiased simulator for reinforcement learning based recommender systems. In *RecSys'20*. 190–199.
[218] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. 2017. Progressive growing of GANs for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196* (2017).
[219] Tero Karras, Samuli Laine, and Timo Aila. 2019. A style-based generator architecture for generative adversarial networks. In *CVPR'19*. 4401–4410.
[220] Animesh Karnewar and Oliver Wang. 2020. MSG-GAN: Multi-scale gradients for generative adversarial networks. In *CVPR'20*. 7799–7808.

Received 3 December 2020; revised 29 May 2022; accepted 3 June 2022