A Survey on Deep Reinforcement Learning-based Recommender Systems

ABSTRACT

Artificial intelligence applications are expected both to be responsive to changes observed in the real-world and to learn from those changes. Reinforcement learning (RL) systems are suitable solutions to deal with these changes because they are capable of learning from interactions and adapting to new situations. In an RL system, an agent (or multitude of agents) observes and interacts with an environment. The agent receives rewards as the response to its actions and modifies its behaviour in such a way as to maximise the expected future rewards it can receive from its environment. Similarly, a Recommender System (RS) application utilises information obtained from its environment (e.g., user, item, context features) to execute actions (e.g., recommending products) in order to maximise the overall performance (e.g., click-through rate, product sales, page visits). The goal of this work is to give an overview of using RL algorithms to solve the RS problem, analysing the current state of the art literature. After introducing RL and RS systems, we detail the issues and challenges observed while applying RL methods to real-world applications, specifically in RS. We explain how RL algorithms are built, i.e., how samples are obtained, how agents are trained and evaluated, and how RL-based RS applications are designed, i.e., how the states, actions, and rewards are represented. Finally, we detail several current RL-based RS systems found in the literature, highlighting their design choices for their components, RL algorithms, and training and evaluation methods.

CCS CONCEPTS

• **Information systems → Recommender systems**; • **Theory of computation → Reinforcement learning**.

KEYWORDS

reinforcement learning, recommender system, neural networks

ACM Reference Format:

. 2018. A Survey on Deep Reinforcement Learning-based Recommender Systems. In *Woodstock '18: ACM Symposium on Neural Gaze Detection, June 03–05, 2018, Woodstock, NY*. ACM, New York, NY, USA, 26 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Learning from interaction has been the foundation of almost all learning and intelligence theories [215]. Artificial intelligence applications, e.g., robots, games, chat-bots, aim to be responsive to changes in the world and learn from their interactions with the world [10]. For example, researchers have developed a chat-bot software that uses natural language to interact with users [10]. The chat-bot receives a query from the user and tries to give a suitable response, by repeating clarifying questions until it finds an appropriate answer. The process of receiving queries or clarifications continues until the user is satisfied with the answer (or until the user gives up and leaves the system). In this example scenario, the environment of the chat-bot is updated by the queries or clarifications and the chat-bot adapts its answers according to these changes. Such AI applications that interact with the world over a sequence of interactions need to use techniques that can deal with the updates in the environments and optimize their behaviours according to the feedback they receive.

Recommender Systems (RSs) is one such domain of AI applications that learn using interactions with the environment. RSs estimate future preferences of users based on their previous interactions with an application [182]. Various real-world applications e.g., social networks, review websites, e-commerce websites, use recommender services to better serve their users. Recommender services are critical for industrial applications to improve the provided services and increase sales and revenue [263]. For example, 80% of movies watched on Netflix or 60% of video clicks on Youtube are attributed to recommendations [48, 76].

RS utilise information obtained from their environment (e.g., user, item, context features) to execute an action (e.g., recommending products) in order to maximise the overall performance (e.g., click-through rate, accuracy) [203]. This continuous interaction of users with the environment makes the RS problem suitable for being solved using Reinforcement Learning methods.

Reinforcement learning (RL) methods are suitable solutions to deal with the challenge of learning from interactions [10] and adapting to new and changing situations. In RL, the environment provides observations to the RL agent, the agent executes appropriate actions, receives rewards as the response to its actions and updates its behaviours according to the received rewards. The overall goal is to maximize an objective function.

There are two main categories of RL methods: The Multi-Armed Bandit (MAB) and the Markov Decision Process (MDP) methods. MAB-based RL agents aim to learn the probability distributions of possible actions, i.e., arms, by interacting with the arms and receiving feedback, i.e., rewards [23]. MDP-based RL methods aim to increase the overall reward by considering both current and future rewards. Many recent works in MDP-based RL literature utilise deep learning (DL) and represent the agent by a deep neural network (DNN) [105, 129]. In this paper, we focus on MDP-based

Woodstock '18, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-1-4503-9999-9/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

Deep RL (DRL) models and their application in RS. Throughout the paper, unless otherwise stated, we use the term RL to refer to MDP-based DRL models.

RL methods have been proved effective in many domains, but especially in games [132, 156, 207, 209], robotics [103, 107], process systems [120, 179] and bio-chemical systems [174, 197]. However, application and deployment of RL methods on other real-world problems, such as recommender systems, remain limited [59, 101, 105, 129].

An efficient implementation of an RL-based RS application has to consider various issues and challenges emerging from the characteristics of RL and RS. There are various RL-based RS applications in the literature, such as [12, 32–34, 36, 39, 57, 65, 72, 77, 78, 102, 103, 122, 131, 133, 134, 136, 142, 144, 145, 188, 199, 200, 206, 210, 219–221, 228, 241, 252, 255, 261, 262, 266–268, 270–273, 278–280], which aim to solve these issues and challenges from their own perspective, e.g. utilising different state representation, agent architectures, objective functions, etc.

The paper is organized as follows: section 2 gives general information on RL and RS systems. section 3 details the issues and challenges observed while applying RL methods in real-world applications, specifically in RS. section 4 explains how the RL algorithms are built, specifically how samples are obtained, how the agents are trained and how they are evaluated. section 5 explains how the RL-based RS applications are designed in terms of main components, i.e., how states, actions, rewards are represented. section 6 details the RL-based RS algorithms in the literature and highlight their choices on the design of the components, the RL algorithms and the training and evaluation settings. section 7 gives insights about the shortcomings and future directions.

2 OVERVIEW OF RECOMMENDER SYSTEMS AND REINFORCEMENT LEARNING

This section describes the basics of recommender systems (RS) and reinforcement learning (RL).

2.1 Recommender Systems

Recommender systems (RSs) estimate future preferences of users based on their previous interactions [182]. Traditionally, there are three basic approaches for Recommender Systems: content-based, collaborative filtering and hybrid approaches. **Content-based (CB)** approaches use the item features to make recommendations. Keeping track of the features of the previously interacted items is exploited to recommend new items which share similar features to the previously interacted ones. **Collaborative filtering (CF)** approaches use similarity among past preferences of users to decide which item to recommend. These methods depend on overlaps of the ratings across users and/or items [28]. There are two versions of CF algorithms: memory-based and model-based. Memory-based approaches calculate the similarity to decide which users/items are neighbours and then, the recommendations are made based on the previous interactions of the neighbours. Model-based CF algorithms usually utilise matrix factorization, which computes a low-rank approximation of input data [141]. In these methods, the items and the users are represented as vectors and the correlations between the vectors are used while making recommendations,

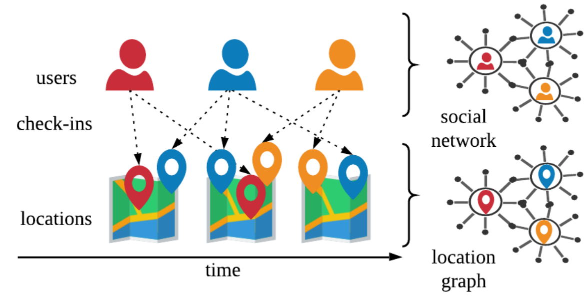

Figure 1: RS using contextual information

i.e., the high correlation between vectors leads to better recommendations [116]. **Hybrid methods** combine two or more recommendation techniques to make better recommendations. There are various combination approaches, such as weighted, switching, and feature augmentation. Details on hybrid recommender methods can be found in [28]. In addition to using only user-item interactions, the base recommendation approaches are extended to integrate **contextual information**, such as the mood of the user, the friendship relations among users, the time of the interaction or the geo-location of items or users (See Figure 1). For example, the geo-location of items can be used for recommending point-of-interests, such as cafes or cinemas that are closer to the target user. Further details on RS methods that focus on different dimensions than user-item interactions can be found in [205]. Examples of traditional recommender methods are given in [14, 18, 37, 47, 76, 82, 91, 98, 116, 127, 141, 157, 177, 190, 246, 254, 276].

Lately, Deep Learning (DL) and Reinforcement Learning (RL) methods have started to gain increasing attention from RS researchers [128, 263, 273]. **DL-based RS methods** are preferred (i) when there is an inherent structure that the model can exploit, e.g., sequential interactions in a session, (ii) when the complexity is huge, e.g., dealing with reviews on items, and (iii) when the number of training instances is large [263]. Various DL methods utilise embedding methods, such as Word2Vec or deep neural network algorithms such as Convolutional Neural Networks (CNN) or Recurrent Neural Networks (RNN) to make recommendations. Further details on deep learning-based RS can be found in [263]. Examples of DL-based recommender systems are given in [38, 75, 88, 95, 160, 167, 183, 189, 224, 237, 251, 265, 274].

RL-based RS methods are preferred (i) when an exact answer for a given situation is unknown, i.e., there are no labels, but instead, there is a natural reward (feedback) for that action, and (ii) when users' preferences dynamically change over a (long) time horizon. RL algorithms utilise the feedback received from the user to update their internal model, which is often a policy [32]. RS and RL systems have many common features [203]: Both utilise the information collected from an environment (e.g., user, item, context features) to decide which action(s) to take (e.g., recommending products) in order to maximise the overall performance (e.g., click-through rate, accuracy). There are various RL-based RS applications in the literature, such as [12, 32–34, 36, 39, 57, 65, 72, 77, 78, 102, 103, 122, 131, 133, 134, 136, 142, 144, 145, 188, 199, 200, 206, 210, 219–221, 228, 241, 252, 255, 261, 262, 266–268, 270–273, 278–280]. Throughout this paper, we'll give more details on the issues and challenges observed while applying RL methods in RS, how the RL algorithms are designed and built considering the specific requirements of RSs and present example research works of RL-based RS algorithms from the literature.

2.2 Reinforcement Learning

Reinforcement learning (RL) is an area of machine learning that deals with sequential decision-making [69], where an agent interacts with a dynamic environment and learns a policy which gives the best action to be performed for a given state. [111]. There are two families of RL methods: The Multi-Armed Bandit (MAB)-based and the Markov Decision Process (MDP)-based methods. MAB-based RL methods, such as [128, 169, 225, 226, 238, 239, 258], learn the probability distribution of possible actions (often called "arms") by having the RL agent continuously interacting with the arms and receive rewards [23]. Contextual MAB methods further incorporate additional information, such as the features of the arms (candidate items) or contextual features. In general, MAB methods only consider the reward of the current iteration, but not of future iterations.

MDP-based RL methods, such as [33, 34, 86, 112, 140, 162, 175, 185, 200, 219, 220, 270, 273], model the learning task as an MDP with finite set of states and actions and aim to increase the overall reward by considering both current and future rewards. Zhao et al. [267] state that conventional RL methods, such as [200, 219, 220], are infeasible for systems with huge numbers of items, such as recommender systems.

Recently, researchers focus more on Deep RL (DRL), which is a combination of deep learning and RL. In DRL, an agent is usually represented by a deep neural network (DNN) and the goal is to optimally learn the weights of the DNN [105, 129]. Further details on MAB- and MPD-based (deep) RL methods can be found in [23, 69, 111, 215].

In this paper, we focus on MDP-based DRL models and their application in RS. Throughout the paper, unless otherwise stated, we use the term RL to refer to MDP-based DRL models.

The classical formalisation of MDP-based RL systems contains an agent, an environment, a set of actions, the states/observations, the rewards and the transitions among states. Figure 2 shows the overview of the RL process: the agent executes an action; as a result of that action the environment returns the reward together with the updated state/observation and, according to the received reward, the agent updates its internal representation. This process runs in an infinite loop, i.e., a loop of actions, rewards, observations or until a terminal state is reached.

MDP-based RL formalization can be represented by a tuple (S, A, T, R, γ) as shown in Table 1. For an MDP, we define $\pi : S \longrightarrow A$ to be a policy which gives the action to be performed for each state. The value of a policy π is calculated by $V^{\pi}(s) = \mathbb{E}_{\pi}[R_t | s_t = s]$ and $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$, where $\mathbb{E}_{\pi}$ is the expected sum of discounted rewards under policy π, t is the current time point and r_{t+k} is the immediate reward at a future time step $t + k$. The objective of an agent is to find an optimal policy π^* with the highest value.

Two main categories of RL methods are Model-based and Model-free [49], according to the usage of a model of the environment or not. **Model-based RL methods** learn a model that estimates the environment dynamics and the reward function. The actions are chosen according to this model. These methods are unsuitable if the state space is too large or the environment is continuous [151]. **Model-free RL methods** assume no knowledge of the transition or the reward function. The agent interacts with the environment and selects the next action to be performed according to the trial-and-error experience [49]. The model-free methods are further divided into three categories [5]:

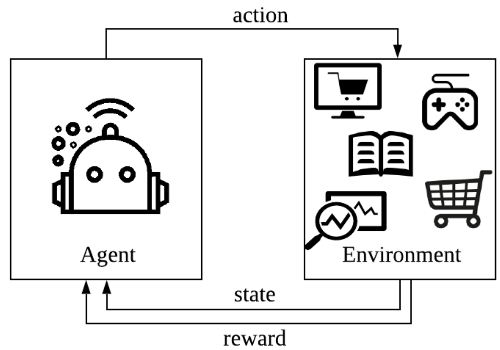

Figure 2: RL overview

Table 1: RL parameters

S	set of States
A	a finite set of actions
T	for each action $a \in A$, T_a is a set of state transition probabilities determining how the state updates upon action α
R	$R_a : S \times S \longrightarrow \mathbb{R}$ is the reward obtained when transitioning from state s to s' upon action a;
γ	$\gamma \in [0, 1)$ is a discount factor
π	$\pi : S \longrightarrow A$ is a policy which gives the action to be performed for each state

(i) **Value-based methods:** These methods (e.g., Q-learning [245] and Deep Q-learning[156]) learn an optimal policy π^* by maximising a value function, which is calculated from either a state $s \in S$ or a state-action pair (s, a) as shown by Equation 1 and 2, respectively. The value function can also be represented with a function approximator such as a neural network with parameters θ using either $V(s; \theta_v)$ or $Q(s, a; \theta)$.

$$V^*(s) = \max_{\pi} \mathbb{E}_{\pi} \Big\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k} | s_t = s \Big\}, \qquad (1)$$

$$Q^*(s, a) = \max_{\pi} \mathbb{E}_{\pi} \Big\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k} | s_t = s, a_t = a \Big\} \qquad (2)$$

(ii) **Policy-based methods:** Policy-based methods (e.g, Policy Gradients [216], and Proximal Policy Optimisation [196]) learn an optimal policy π^* by considering the probability of performing action a when the agent is in state s. They optimise the model parameters θ directly by gradient ascent on the performance of the objective function $J(\pi_\theta)$, without using a value function. This is useful when the action space is continuous or stochastic.

(iii) **Hybrid (Actor-Critic) methods:** Hybrid RL methods (e.g., A3C [155]) combine value-based and policy-based approaches. Usually, a neural network is used to maintain both (i) a policy $\pi(a_t|s_t;\theta)$ (i.e., Actor) that controls the action to choose, and (ii) an estimate of the value function $V(s_t,\theta_v)$ (i.e., Critic) that measures the quality of the current policy. Instead of using the total rewards of the episode to determine how good a policy is, an Actor-Critic model trains a critic that approximates the value function to better evaluate each stage in the episode. As a result, it reduces the variance in the training examples and makes the learning process more stable than pure policy-based methods. Since it uses an approximation of the expected return of a state, it introduces bias [10, 115, 195].

3 ISSUES AND CHALLENGES OF RL IN RS

There are various challenges that researchers encounter while developing and/or deploying RL methods in real-world systems. Application of RL in RS has also some distinct challenges, which are discussed in the following sections.

3.1 Issues and Challenges of RL in General

RL methods suffer from various issues and challenges, such as high dimensional states and action spaces, sample efficiency, generalization, reproducibility, explainability, interpretability, safety, scalability, accuracy, robustness [58, 59, 129]. In this section, we focus on (i) sample efficiency, (ii) unspecified and/or multi-objective reward functions, (iii) generalization issues, (iv) reproducibility issues and (v) evaluation.

Sample efficiency: RL systems normally require large amounts of data to efficiently learn their models, so they are not practical for problems where obtaining data is expensive [26]. Generally, there are three options for obtaining samples and training an agent: from a real system, from an environment simulator or from fixed logs [59] (Detailed in subsection 4.1). These methods suffer from the high cost of exploration, from poor actions or from the limited available number of samples. RL methods, especially model-free RL methods, are not sample efficient, such that they require millions of samples for training [26, 59, 105]. For example, Rainbow [92] reaches the human-level threshold on the Atari game at about 83 hours of play experience, which is far longer than the few minutes that humans require to reach the same level [105]. In order to solve the sample efficiency problem, various approaches have been proposed, such as meta-learning [66], expert demonstration [94, 234], sampling and ensembling [26, 40, 93, 165].

Unspecified or multi-objective reward functions: In RL systems, the reward function is used for optimizing the agents' behaviour and it must capture exactly what the agent should be doing [41, 59, 105]. However, in real-world applications, it is not always possible to define a suitable reward function for capturing the user intention. For example, an agent trained for the CoastRunners game learned to play the game achieving higher scores than human players, but in the wrong way [41]. The agent learned a false behaviour, gaining more points by only hitting other users instead of finishing first, which in the end resulted in higher scores.

It is common to have the agent trained for one metric and then to discover that another metric is also important [59]. For example in RS domain it is important to recommend items which will be clicked or purchased, but researchers or product owners may later find out that diversity is also important (e.g., if always similar items are recommended, users can get bored and leave the service earlier). For such cases, the design of a global reward which combines all the (known) sub-goals, e.g., accuracy and diversification, is necessary, but can be a more complex task. In order to address the problems related to reward function, various approaches are proposed, such as introducing new terms to the reward function and tuning coefficients [173] or recovering the underlying reward function from demonstrations and feedback [3, 80, 163, 186, 187].

Generalization Issues: RL models should be able to handle any kind of input, even in the unforeseen situations [64]. In RL generalization is considered in two ways: (i) in-distribution and (ii) out-of-distribution generalization [168]. The former is similar to supervised learning, i.e., the RL algorithms are expected to perform well when they are evaluated in similar environments, with similar data distribution [64]. The latter is more related to transferring the knowledge acquired by the trained model to different tasks with different data distributions [15, 16, 64]. RL agents tend to overfit [168], since most of them are trained and evaluated on a single environment [101]. As a result, they struggle to learn representations that generalize to the unseen instances [64, 168]. Some example approaches for analysing and solving the generalization problem are: Observing the effects of neural network architecture, training data size, reward functions and RL algorithms [42, 168, 231, 259], creating environments which have distinct training and test sets [42, 64, 67, 109, 110, 164, 250, 260], applying data augmentation techniques [180, 231] and applying regularization techniques, e.g., dropout, L2 regularization [42, 64, 123, 231].

Reproducibility Issues: True progress in RL, and other machine learning settings, can be measured by reproducing the existing works and judging the improvements of new methods compared to the baselines [46, 90]. However, reproducing results of RL methods is usually challenging, since they utilise various extrinsic and intrinsic settings [90]. Henderson et al. [90] inspect that hyper-parameter tuning, choice of network architecture, different implementations of the same algorithm, the scale of the reward, the random seeds used in training and the choice of the environment affect the performance of the RL model [105]. Engstrom et al [60] present an example comparison on different implementations of two algorithms and show that code-level optimizations are the source of the performance gain than the choice of the algorithm. In order to provide reproducible RL methods, all settings must be reported together with appropriate evaluation metrics and insights on how these settings affect the agent training [60, 90]. Besides these, using existing benchmarks is also useful for reproducibility. [54, 143, 247] provide benchmark implementations, evaluation metrics and environments.

Evaluation Careful evaluation is necessary for keeping the progress in RL and other machine learning research [13]. Evaluation can be explained in two folds: Having a reproducible evaluation environment and estimating the performance of the methods effectively. Simulation environments and/or RL platforms, such as

OpenAI Gym [25], are commonly used for training and evaluating RL methods [101]. Recently, Osband et al. [166] introduced Behaviour Suite for Reinforcement Learning (BSuite) which automates evaluation and analysis of any agent. BSuite collects the best available experiments to provide insight on the key aspects of agent behaviour. It aims to provide a simple and easily accessible library for researchers and developers. Also, Dulac et al. [58] present a framework, namely Real World Reinforcement Learning Suite (RealwWrlDRL-suite), to evaluate RL algorithms for their potential applicability to real-world systems. It aims to identify the challenging aspects of real-world tasks and analyse the RL agents in terms of those challenges.

Traditionally in RL, agents are trained and evaluated in exactly the same task (e.g., environment) [64], which leads to overfitted models and poor evaluation results. Recently, approaches which create distinct training and test environments are proposed, such as [42, 64, 67, 109, 110, 164, 171, 250, 260]. In order to estimate the performance of the methods, either the model is evaluated in a a real-world scenario, in a simulation environment or on logs collected from real-world. Since deployment to real-world is impractical and has a high cost, the performance of the learned policy is estimated without running it on the real system. In such cases, it is common to execute importance sampling [176], doubly-robust estimators [55, 63, 108] and combination of different estimators [229].

3.2 Issues and Challenges of RL in RS

RL-based RS suffer from the same general problems as other real-world RL applications. For example, in RS there are millions of items that can be recommended to the users, which can lead to high dimensional action spaces [44]. Moreover, in RS, each user has distinct (and independent) interactions with items, which are normally limited in number to fully train a model [101]. As a result, the available interactions, i.e., data samples, should be used efficiently and the recommender agent should be able to generalize across users [101].

There are other problems that are not specific to but frequently observed in RL-based RS applications. In this section, we'll discuss the issues and challenges related to (i) partial observability, (ii) high dimensional states and action spaces, (iii) long horizons and delayed rewards, (iv) slate recommendation and (v) stochasticity and non-stationarity.

Partial observability: Like most real-world systems, RS are only partially observable [59, 101]. For example, while making a recommendation, an RS utilise various features of the target user, such as her previous interactions, geo-location, and temporal preferences. However, the RS often does not know many other features, such as the mood or the personality of the user. In the literature, partial observability problems are formulated as a partially observable MDP (POMDP) [59, 101]. There are three common approaches to handle partial observability: (i) POMDP algorithms can maintain a belief on the current state based on the current observation, previous belief state and the action [10]. The example works maintaining belief states belong to [104, 150, 242] (ii) RNNs can be used to solve the POMDPs [10, 59, 242], by aggregating the previous observations to the model, track and recover hidden states. The example works utilising an RNN are [85, 277]. (iii) Incorporating history into the observation of the agent is also commonly used [10, 59, 242]. It stacks previous observations with the current observation and the result is then used as the input of the system. For example, the Atari agent [156] stacks four frames as the agents' observation.

High dimensional state and action spaces: Most real-world systems have very large state and action spaces [59]. For example, real-world RS systems choose a small set of items to recommend from hundreds of millions of items (i.e., actions) for each user at every second [44, 57]. As a result, it becomes intractable to evaluate and choose the right items (actions) for the target user [57, 59, 266, 269]. The relations among the actions is also an important factor in terms of the number of actions. In RL, the agent needs to learn from samples and having related actions may help to use each sample more efficiently [57]. Dulac et al. [59] state that learning with millions of related actions is much easier than with a couple hundred completely unrelated actions. In order to solve the high dimensional state and action space problem, various approaches are proposed, such as (i) factorization of the action space into binary sub-spaces [56, 170], (ii) learning in a continuous action space and then utilizing nearest neighbour approaches for discrete actions [57, 233], (iii) (separate) embeddings for the state and action spaces [87, 273], and (iv) eliminating irrelevant actions [257].

Long horizons and delayed rewards: The real-world RL systems may suffer from long horizons and delayed rewards. When the real-world application has a long horizon, the (latent) state of the environment can evolve over the horizon, e.g., in weeks, years [101, 154] and an RL agent should capture the evolution, even over the long horizon. For example, a user's movie genre preference can evolve over years from animation to science-fiction and the movie recommender agent should capture this. However, reasoning about MDPs over the extremely long horizon is challenging for many RL methods [101]. One solution to this problem is imitation learning [3, 96]. However, imitation learning may require a large amount of demonstration data [121]. An alternative solution is to exploit the hierarchical structure of the problem, as done in [117, 121, 217]. Another alternative approach called advantage amplification which introduces temporal abstraction across policy space is proposed recently [154]. Another issue raised from long horizon is delayed rewards. When the reward information arrives considerably later than the execution of the action, the RL agent may suffer from partial feedback and bias, weak monitoring and debugging and incorrect data collection [129]. In RS, it is common to have a delayed outcome, especially when the reward is based on the user's interaction with the recommended item [59]. For example, when the reward is set as finishing reading an e-book, the reward will be collected when the users finish the book, i.e., after a few days or weeks [148]. The example works focusing on delayed reward tasks are [7, 8, 100]. Recently, [148] proposed a factored learning approach which utilises intermediate reward signals for an RL-based RS application.

Slate recommendations: Recommending 'slates', e.g., a list or page of items, is named as 'slate recommendation' [101–103]. In the RL literature, the challenges on slate recommendations are usually associated with the combinatorial effect among multiple items [12, 36], such that the focus is on the optimization for recommending a whole set of items [77]. Additional to whole set of items, we consider top-k or list recommendation, where a list of items are recommended and each item is independently evaluated, as slate recommendation in this survey. In slate recommendation, the action space expands from a single action to a combination of actions. As a result, it introduces other challenges such as generalization, action optimization issues [103]. In the literature, there are mainly two groups of approaches related to slate recommenders. The first group focuses on the generic problem related to combinatorial actions [152, 213]. These approaches are not focusing on the recommendation problem and have a disadvantage of being unscalable for the large, real-world recommenders, which is attributed to their generality [102]. The second group focuses directly on the recommendation of slates [102, 103, 136, 218, 267], which have shown promising results on their applicability to real-world systems. Recently, Swaminathan et al. [218] introduced a new estimator for the evaluation of slate recommenders, which is useful for deployment of real-world slate recommenders.

Stochasticity and non-stationarity Real-world systems are often non-stationary, e.g., a pump's efficiency degrades, and stochastic, e.g., each robot being operated behaves differently [59]. RS systems are also non-stationary and stochastic. Firstly, the action space is non-stationary, i.e., the set of recommendable items is not fixed [101], and stochastic, i.e., the change in the set of items are unpredictable. For example, in a news recommendation system, as events happen, new articles on those events emerge and the out-dated ones are (sometimes) deleted. Secondly, the user space is also stochastic, i.e., even though users may have the same representation (because of the partial observability), each has his own (independent) preference. As a result, the same recommendation for those users may/not return the same reward, i.e., like/click the recommended item. The original MDP formulation cannot incorporate stochastic action sets efficiently [31]. One naive approach to solve the problem is to embed a subset of available actions into the state representation [24]. Because states are populated with subsets of all the possible actions, this approach results in huge state space and it is not tractable [24]. One solution for this problem can be using state-specific (ordered) decision lists over the action set [24]. Another solution is to use stochastic action sets, but it increases the uncertainty[31]. To reduce uncertainty variance reduction techniques gain more importance [31]. Additionally, for dealing with uncertainty the Robust MDP formalization [106] can be used. Example applications using Robust MDP are [50, 147, 201, 222].

There are other challenges that are frequently observed in RS applications, such as cold start users/items, long-tail users, fairness, explainability, diversity, privacy and security problems [22, 101, 182]. Further details on these challenges can be found in [22, 182]. While developing RL-based RS applications these kinds of RS-specific challenges should also be taken into account.

4 HOW TO BUILD RL APPLICATIONS?

Building an RL application requires multiple components that can integrate and communicate well with each other. The application needs to obtain the samples from some environment, implement the RL agent, train a model utilising the RL agent and evaluate the model. Utilising existing frameworks in each step helps developers/researchers to implement the RL applications easier [181] and allows them to reproduce existing works.

4.1 How to obtain samples?

There are three options for obtaining samples and training an agent, in general [59]:

- In **Learning from a real system**, all training data comes from the real-world in sequence, i.e., online mode. As a result, it is more likely to learn real interactions better. However, in these systems, exploratory and/or poor actions have a higher cost, such that in real-word/production systems cost of a performance drop could be very high [59]. For example, in a news article RS application, if the recommender agent starts to recommend outdated or out-of-interest articles, the user may end the session, i.e., leave the application.
- In **Learning from a simulator**, the simulator produces data in sequence, i.e. online mode. The simulation model can be trained from logs from real-world, benchmark datasets or synthetic data produced according to required statistical features, e.g., distribution of ratings. In this kind of learning, the number of samples is unlimited, there is no or low cost for poor actions (e.g., the cost related to training time, energy spent remains) and agent can interact with the simulator as long as it requires. However, for many real-world applications there is no existing simulator and developing one is difficult or impractical [59].
- In **Learning from fixed logs**, i.e., off-line learning, the logs can be the records from the same policy, i.e., on-policy, or they can be the records from another, unrelated policy, i.e., off-policy. Also, the logs may belong to a commercial application or they can be benchmark datasets. For example, Zhao et al [267] utilizes logs from a real e-commerce company for the evaluation of their RL-based RS system. In this kind of learning, there is no or low cost for poor actions, but the number of samples is limited. Since the benchmark datasets from the RS literature, e.g., MovieLens dataset [83], are usually composed of logs of user behaviours, we consider the usage of benchmarks as a way of learning from logs.

Researchers, especially the ones based in an academic institution, do not always access to a real system to train their RL agents. Moreover, it is better to use simulation environments and/or publicly accessible fixed logs in order not to introduce further reproducibility issues. In the literature there are various simulation frameworks (gyms) which provide benchmark environments [25, 202], such as OpenAI Gym [25], ViZDoom [113], Deepmind Lab [17], ELF [230], GymExtensions [89], CARLA [53], RecoGym [184], PyrecGym [203], Virtual Taobao [204], Google RecSim [101] and Park [149]. In this section, we'll give more details on the gyms which can be used for training RL-based RS agents.

OpenAI Gym [25]: OpenAI Gym is a toolkit for reinforcement learning, which provides common interfaces for benchmark tasks. It has publicly available, open-sourced implementation[1] and it is written by OpenAI. The OpenAI Gym separates the environment from agent implementation and only provides implementations (and interfaces) for environments, such that it makes no assumptions about the agent [25]. It provides common interfaces for a few functions, namely (i) 'make' for creating an environment, (ii) 'init' for the initialization of the environment, (iii) 'step' for acting in the environment (by one time-step), (iv) 'reset' for resetting the

[1]OpenAI Gym: https://github.com/openai/gym

	OpenAI Gym [25]	**RecoGym [184]**	**PyrecGym [203]**	**Virtual Taobao [204]**	**RecSim [101]**
Affiliation	OpenAI	Criteo	UCD	Alibaba Group	Google
Open Sourced?	Yes	Yes	No	Yes	Yes
Reward Function	Configurable	Static (i.e. click)	Configurable	Static	Configurable
Data	Simulated	Simulated	Logs	Simulated (i.e., trained from logs)	Logs (i.e., doc. database)
Data Preprocessing?	No	No	Yes	No	No
Customized for RS?	No	Yes	Yes	Yes	Yes
Allow New Env.? (I.e., Provide Interfaces?)	Yes	Yes	Yes	No	Yes

Table 2: Toolkits for environment implementations (Gyms)

environment's state. There are various tasks (called environments) implemented in this gym, such as Atari, CartPole. These environments are versioned to ensure the results remain meaningful even when the implementation is updated. Additionally, it provides a website where people can share and compare the performance results of their algorithms. As a result, this gym provides reproducible and comparable environments. OpenAI Gym is not specifically developed for RS applications, however, it is possible to use its function interfaces to create a custom RL-based RS application.

RecoGym [184]: RecoGym is an RL gym for recommendation, which is specifically defined for recommender systems on advertisement in e-commerce domain. It has publicly available, open-sourced implementation[2] and it is copyrighted by Criteo. RecoGym defines two types of user behaviours, namely organic and bandit sessions, which happens on e-commerce and publisher websites, respectively. An organic session is defined as the session where users view the recommended items, whereas the bandit session is defined as the session where the agent recommends items to users. The reward is set as the click. Additionally, RecoGym provides inverse propensity scoring (IPS)-based and other classical offline evaluation approaches to provide a benchmark, reproducible environment. Even though RecoGym is an extension on OpenAI Gym, it requires additional functions while training an agent. The user behaviours in these sessions are created by simulation and it does not support usage of existing benchmark datasets, e.g., logs. Since the data is simulated, it does not have support for data preprocessing.

PyRecGym [203]: PyRecGym is an RL gym which is specifically designed for recommender systems. It is developed at the Insight Centre for Data Analytics, University College Dublin (UCD) [203] on a grant given by Samsung Electronics. It is an extension over OpenAI Gym and capable of working together with any agent which is compatible with OpenAI gym. It does not provide any simulated data but works with standard RS datasets, such as MovieLens, Yelp datasets. Also, it supports common input types encountered in RS datasets, such as textual, numerical data. It lets the users define their states, actions and rewards according to the needs of the algorithm, which makes it more modular compared to some of its counterparts. One main drawback of this gym is that its code is not released for public usage, yet.

Virtual Taobao [204]: Virtual Taobao is rooted from the researches made on the search engine of an online shopping website named Taobao. It is an extension over OpenAI Gym and has open-sourced implementation[3]. It is implemented by researchers (i.e., full-time or intern) at Alibaba Group. Virtual Taobao uses historical customer behaviours data to simulate user interactions. Additional to the customers' historical data, they generate virtual customers by their GAN-for-Simulating-Distribution (GAN-SD) approach and interactions by their Multi-agent Adversarial Imitation Learning (MAIL) approach. In this gym, the customers are generated once at a time, then this virtual customer starts interacting with the system, i.e., the customer starts a query, the recommender agent makes the recommendations and the customer clicks/not the recommended items. Currently, there is a single environment which is trained from a middle-scaled Taobao dataset. It is unclear if it is possible/not to integrate any other benchmark dataset.

RecSim [101]: RecSim is a configurable gym specifically designed for RS. It has publicly available, open-sourced implementation[4]. Even though the code is written by Google employees and it is hosted on a page managed by Google, it is not an official Google product. RecSim is composed of a document model, a user model and a recommender model. While document model and user model compose the environment, the recommender model is the representation of the agent. The document model consists of the set of sample documents, document database, i.e. set of candidate document in each step, and document observable features. The user model consists set of sample users, users' observable and hidden features, users' choice model, users' transition model, users' response information and users' next hidden state features. Similar to most of the other gym implementations, RecSim is also an extension over OpenAI Gym. It allows creating custom environments, with its data (i.e., kept in document database) and with custom reward. However, it is not clear how to use other datasets and prepare the document database. Also, it is unclear how to preprocess and normalize custom or benchmark datasets.

4.2 How to train RL agents?

RL frameworks provide implementations of existing algorithms and high-level abstraction for the core RL components. As a result, they make it easier to train and evaluate the agents and to develop new algorithms. Even though researchers/developers aim

[2] RecoGym: https://github.com/criteo-research/reco-gym

[3] Virtual Taobao: https://github.com/eyounx/VirtualTaobao

[4] RecSim: https://github.com/google-research/recsim

	Coach [29]	**ReAgent [74]**	**RLLib [130]**	**ChainerRL [71]**	**RLGraph [192]**
Affiliation	Intel	Facebook	U. of California	U. of Tokyo, Preferred Networks	U. of Cambridge, rlcore, Helmut Schmidt U.
Aim	Research	Production	Research	Research, Development	Research, Practice
Open Sourced?	Yes	Yes	Yes	Yes	Yes
Multi-thread training?	Yes	Yes	Yes	Yes	Yes
Multi-node/ distributed training?	Yes	Yes	Yes	No	Yes
DL Framework(s)	TensorFlow, MXNet	PyTorch	TensorFlow, PyTorch	Chainer	TensorFlow, PyTorch
Algorithms	Value opt., Policy opt., Imitation Lrn., Multi-agent Lrn.	Value opt., Policy opt.	Value opt., Policy opt., Imitation Lrn., Multi-agent Lrn.	Value opt., Policy opt.	Value opt., Policy opt., Imitation Lrn.

Table 3: Reinforcement Learning frameworks

to implement modular frameworks which are easy to extend, combine and use, it is not easy to provide complete and simple RL implementations [130, 202]. Current RL frameworks have a tendency to implement RL agents as single strongly connected processes [130, 202]. In the literature there are many RL frameworks, such as DeeR [68], KerasRL [172], Decision Service [6], OpenAI Baselines [51], Coach [29], TensorForce [191], MAgent [275], TF-Agents [198], Dopamine [30], RLLib [130], ReAgent [74], Surreal [62], rlpyt [212], SimpleRL [4], ChainerRL [71], Catalyst [114], RLGraph [192], SLM Lab [138], Garage [73] and RLax [27]. Analysis of various RL frameworks can be found in [74, 181, 192, 202]. In this section, we give details on a subset of RL frameworks which can be used for training and evaluating RL-based RS agents.

Coach [29]: Coach, also known as 'Intel Coach', is a framework which contains the implementation of may state-of-the-art RL algorithms. It has publicly available, open-sourced implementation[5]. Even though its source code is hosted on a page managed by Intel, it is not an official Intel product. Coach provides basic RL components for modelling an agent, such as algorithms, neural network architectures and exploration policies. These components are decoupled, as a result, they can be combined in various ways while developing a new agent. Coach contains implementations for many of the state-of-the-art value optimization, policy optimization, imitation learning and hierarchical RL based algorithms, such as DQN [156], ACER [243], PPO [196], DFP [52], TD3 [70], Rainbow [92], CIL [43]. As the main back-end framework, it uses TensorFlow, but it also supports MXNet. It supports single-thread, multi-thread and multi-node (distributed) training. It has mechanisms for evaluating the agents and visualising the statistics collected during training and evaluation. Moreover, it provides interfaces for defining custom environments. It also supports a set of existing environments, such as DeepMind Control Suite [227], OpenAI Gym [25], GymExtensions [89], CARLA [53]. However, it is not clear if it is simple to integrate other environments, such as the environments created specifically for RS, to the system.

ReAgent [74]: ReAgent, formerly known as Horizon, is an end-to-end RL framework for applied RL. It has publicly available, open-sourced implementation[6] and it is developed by Facebook. ReAgent aims to solve industry-level RL problems where datasets are large, the reward is delayed and poor actions have higher cost (because experiments do not run in a simulator). It provides implementation for a set of algorithms, namely DQN [156], Double DQN [84], Dueling DQN [244], Dueling DDQN [92], C51 [19], QR-DQN [45], TD3 [70] and SAC [79]. It supports CPU, GPU, multi-GPU, and multi-node training. It uses PyTorch for modelling and training and TorchScript for model serving. Additionally, it contains mechanisms for data pre-processing, feature transformation, distributed training, counterfactual policy evaluation, and optimized serving. For visualisation, it uses TensorBoard using TensorboardX [99] plugin, which converts the PyTorch/Numpy tensors to the TensorBoard format. For the evaluations, it includes custom environments (i.e. Gridworld) and standard OpenAI Gym [25] environments. In general, it decouples gym from agent implementation and reads gym-related configurations from a JSON file [181].

RLLib [130]: RLlib is a library for RL which offers high scalability and modularity. It is based on Ray project [158], which is a framework for building and running distributed applications. It has publicly available, open-sourced implementation[7] and it is developed by researchers at University of California. RLLib provides various modules, such as the environment, neural network model, action distribution, and policy definitions. The implemented algorithms are grouped into three: (i) High-throughput architectures: APE-X [97], IMPALA [61], APPO [211], DD-PPO [248], (ii) Gradient-based algorithms: A2C [155], A3C [155], DDPG [132], TD3 [70], DQN [156], Parametric DQN [253], Rainbow [92], Policy Gradient [216], PPO [196], SAC [79] and (iii) Derivative-free algorithms: ARS [146], QMIX [178], VDN [214], IQN [223], MADDPG [139], MARWIL [240], AlphaZero [208]. Most of its internals are independent of any framework, but it supports TensorFlow and PyTorch. It supports training with offline data, simulation data and their combination. It provides three types of environments: gym environment (based on OpenAI Gym [25]) for single-agents, vector environment for multiple agents with a single policy and multi-agent environment for multiple agents with multiple policies. Custom environments can be integrated into the framework by implementing them according to the provided interfaces and registering the new environment. Even though the framework aims to be modular and flexible, its optimizer mixes Python, Ray and Tensorflow calls which makes it less flexible [192] and complex [181].

[5]Coach: https://github.com/NervanaSystems/coach
[6]ReAgent: https://github.com/facebookresearch/ReAgent
[7]RLLib: https://github.com/ray-project/ray/tree/master/rllib

ChainerRL [71]: ChainerRL is an RL framework that implements various state-of-the-art RL algorithms. It is based on Chainer [232], which is a flexible deep learning framework. It has publicly available, open-sourced implementation[8] and it is developed by Preferred Networks and researchers from the University of Tokyo. ChainerRL replicates the original papers' experimental settings and reproduces the published benchmark results. It contains implementations for the state-of-the-art RL algorithms, namely DQN [156], Categorical DQN [19], PAL [20], DPP [11], DDPG[132], Rainbow [92], IQN [223], ACER [243], A2C [155], A3C [155], Asynchronous N-step Q-learning [155], PCL [161], PPO[196], REINFORCE [249], SACSAC [79], TRPO [194], TD3 [70]. ChainerRL supports any gym that uses the OpenAI Gym [25]'s interface.It has accompanying visualization tools for understanding and debugging the agents. ChainerRL documentation does not explicitly mention support for multi-node training. However, Chainer framework, which ChainerRL is based on, has support for it. Unfortunately, it is not clear if this feature can be directly used by ChainerRL or not.

RLGraph [192]: RLgraph is a framework which aims to enable developers both in research and practice to quickly prototype, define and execute RL algorithms. It has publicly available, open-sourced implementation[9] and the related paper indicates that researchers at University of Cambridge, rlcore and Helmut Schmidt University equally contributed. RLGraph decouples the logical component composition from deep learning backend and distributed execution. As a result, it lets designing multiple distributed backends and device execution strategies without modifying the agent definitions. It implements DQN [156], Double DQN [84], Dueling DQN [244], Prioritized experience replay [193], Deep-Q learning from demonstration [94], APE-X [97], IMPALA[61], PPO[196], SAC [79], REINFORCE [249], A2C [155], A3C [155] algorithms. For distributed execution, it can use TensorFlow, Ray, and Horovod. For local backend operations, it can utilise TensorFlow, PyTorch or any other application library (e.g., numpy). It supports many existing environments, such as OpenAI Gym [25], Deepmind Lab [17], and provides a common interface for them.

4.3 How to evaluate RL applications?

There are a few different alternatives for the evaluation of RL-based RS applications. The RL agents can be evaluated (i) on real-systems, simulated data or logged data (ii) in an online or offline setting, (iii) while the parameters of the agent are kept fixed or updating.

The data for the evaluations can be obtained from real-world, simulations or logged data, as explained in subsection 4.1. The evaluation setting can also be defined by the kind of data used, such that if the environment changes along with time or not [34], i.e., online or offline:

- **Online mode**: In online evaluation setting, the data is expected to be obtained sequentially. Also, a reward is returned for each action, e.g., each recommended item, even if the user has never interacted with or rated the recommended item before [134]. This is the default mode for real-world applications, as the reward is obtained from users directly, e.g., click or not click. For the simulation environments, the simulator should have mechanisms to produce rewards and state transitions even if the related state-action pair is not observed in the source which the simulator is trained on, e.g., benchmark dataset. For example, Liu et al. [134] train a matrix factorization algorithm for the simulation environment to return feedback for the items that the user never rated before.
- **Offline mode**: In offline evaluation setting, the rewards for the unobserved interactions are expected not to be calculated but directly assigned to a predefined value, such as zero (or the minimum value according to the reward function). When log data is used for the evaluations, it is expected to use offline evaluation as the default setting, since there is no information on what the reward should be for an unobserved state-action pair. Some of the simulation environments can also be considered as using offline mode. For example, if the simulator provides a sequence of interactions by reading a benchmark dataset and use the labels provided by the data source as the reward, i.e., without computing rewards for unobserved actions, then that simulator can be considered as running in offline mode.

The evaluation settings where data from real-world, simulation or logs are used in online or offline mode may apply updating (dynamic) and fixed (static) modes while executing the evaluations:

- In the **updating (dynamic) evaluation** setting, the parameters of the RL agent, e.g., weights of the deep neural network, keeps updating [134].
- In the **fixed (static) evaluation**, the parameters of the RL agent remains the same, i.e., the policy is not updated. This is similar to the evaluation of supervised learning algorithms [134].

Depending on the updating/fixed setting, the agent updates its policy or not based on the reward feedback. For example, when real-world is used as the evaluation environment, it is possible to update the parameters and weights along the process or it is possible to keep the parameters and weights fixed. In such a fixed setting, according to the model's performance, the update can be later executed manually.

5 HOW TO DESIGN RL-BASED RS APPLICATIONS?

There are various RL-based RS methods, i.e., recommender agents, in the literature. Liu et al. [133] propose that these methods can be summarised in a unified architecture which is composed of three components, as shown in Figure 3:

- Embedding component maps the input sparse high-dimensional vector representations, e.g. items and a single user's demographic information in Figure 3, to low dimensional dense vector representations, i.e., embedding space.
- State representation component models the environment state. It can have various architectures, such as fully connected neural networks [273] or recurrent neural networks [270].
- Policy component outputs the Q-values or the policy for value-based or policy-based models, respectively. The action to take is decided based on the output. For example, in Figure 3 Q-values are used for deciding an action.

When the reward is received, all these components are updated, i.e. back-propagation is executed in the overall NN.

[8]ChainerRL: https://github.com/chainer/chainerrl
[9]RLGraph: https://github.com/rlgraph/rlgraph

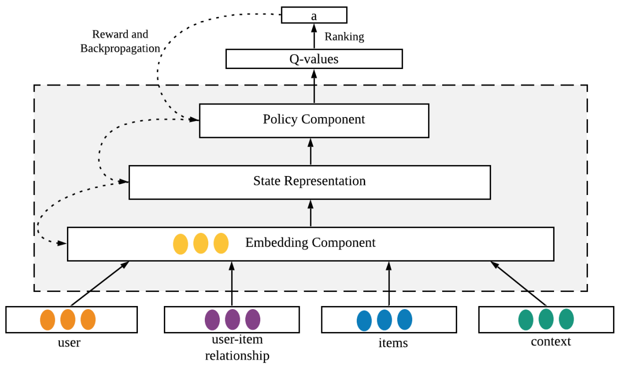

Figure 3: Unified RL-RS architecture described by Liu et al. [133]

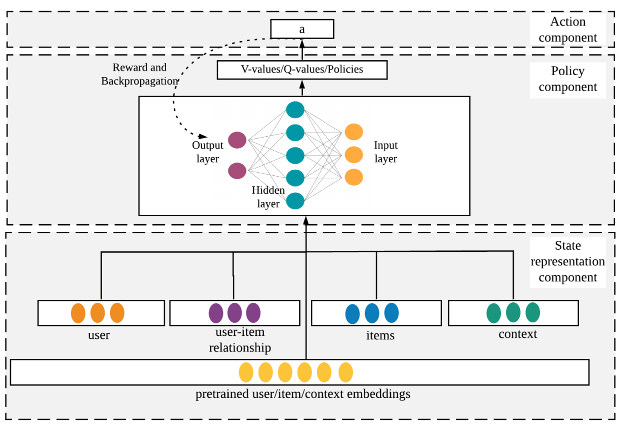

Figure 4: Unified RL-RS architecture

In this paper, we expand (and sometimes combine) the components proposed in Liu et al. [133]. We provide design alternatives for each MDP component, such that our proposed components are state representation component, policy component, action component and others (for rewards and transitions), as shown in Figure 4.

5.1 State representation component

We define the state representation component as the input layer of the RL agent. It can be considered as the equivalent of the input latent representations of Liu et al. [133]'s unified architecture, i.e., items and a single user's demographic information in Figure 3. Unlike Liu et al. [133], we do not consider only the sparse high-dimensional vector representations as the input. Our state representation component may contain any kind of input data, such that data may contain high-dimensional vector representations (e.g, a rating matrix, co-occurrence matrix) or dense embeddings. Since we accept embeddings as an input as well, one may also consider this component as an extension to embedding and state representation components of Liu et al. [133]'s unified architecture. The main difference of our representation is that our states are received from the environment and are not updated by the agent. In Liu et al. [133], their embedding component can be either fixed, i.e., not

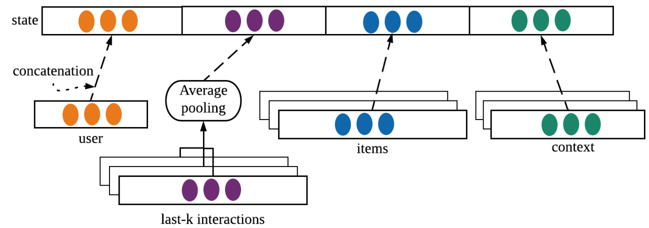

Figure 5: State representation and examples on how the features are combined

updated, or updated when the back-propagation is executed on the neural network of the agent.

In general, the input data contains a combination of one or more of the following representations: (i) item features, e.g., the genre of a movie, length of a playlist, (ii) user features, e.g., gender, occupation, (iii) context features, e.g., date, location of the interaction, (iv) user historical behaviours features, e.g., previous k-items purchased by the user (v) their combinations or inter-relations, e.g., similarity among items, or (vi) their embeddings (dense vector representations) which are created by an external software, e.g., Word2Vec [153]. Figure 5 presents an example state representation composed of the user, item, context features and the last-k interactions of the user.

The decision of which features to be used depends on the application and the research question. For example, if the researcher/ developer decide to use multiple (selected set of) items, she needs to decide those items to be used, e.g., in each iteration. Some naive approaches for item selection are using the popular items, the items the user previously interacted or the items that the user's friends previously interacted. In order to provide personalized recommendations, it is better to consider the user-item relations, such as utilising the last k-many interacted items for the target user, than popular items. The features which are decided to be used for the state representation can be combined in various ways. For example, the state representation can be the concatenation of multiple items' features, or the concatenation of a single user's features with the average of a selected set of items' features, i.e., the average of the previous k-many interacted items of the user (Figure 5).

5.2 Policy component

We define policy component as the layer that executes the RL algorithm. It is responsible for learning from input state representations and producing the output Q-values or policy. The outputs can be used directly as action or a post-processing step can be executed. Even though policy component does not depend on the algorithm type or its architecture, in the literature most of the real-world applications utilise deep neural networks.

We design the neural network architecture in three layers in general, similar to what's suggested by RLCoach framework [29, 118] and shown in Figure 6: (i) 'Input' layer converts the input observation into a feature vector representation. There can be more than one input embedder in a neural network, for example, to support multiple observations as input, (ii) 'Middle' layer processes the received feature vector representations. For example, it combines the

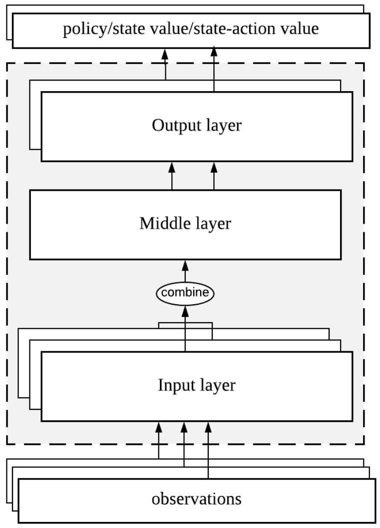

Figure 6: Network architecture, similar to RLCoach [118]

output of multiple input embedders into a single representation. (ii) 'Output' layer produces the output action-values, state-values or a policy. There can be more than one output layers in a neural network, for example in actor-critic algorithms there are layers for the policy (i.e., actor) and the state-value (i.e., critic). These layers are updated when the reward is received from the environment and backpropagation is executed. Example neural network architectures are presented in Figure 7. In Figure 7b, the architecture for DQN [156] algorithm is presented which contains a single input, middle and output layers. In Figure 7b an example actor-critic architecture is presented with its two output layers. In Figure 7c the architecture for DDPG [132] algorithm is presented. Its neural network architecture is more complex compared to the other examples, such that two networks are combined, where one of the network's output used as an input by the other.

The layers defined in policy component, which are equivalent to RLCoach framework [29, 118], are also similar to the overall unified architecture of Liu et al. [133]. The embedding component, state representation component and policy components in Liu et al. [133] correspond to input, middle and output layers, respectively. However, there is a slight difference in 'embedding' layer, such that in Liu et al. [133] the embedding layer can be fixed or updated, whereas in our design it is always updated (by back-propagation). We consider non-updated embeddings belong to the state representation component, rather than policy component.

5.3 Action component

We define action component as the actual action to be taken, e.g., which item is recommended.For RS, the action can be represented in various ways:

- **Discrete action:** The output is a single discrete value, which may indicate if a candidate item would be liked/clicked by the target user/not or the item id.

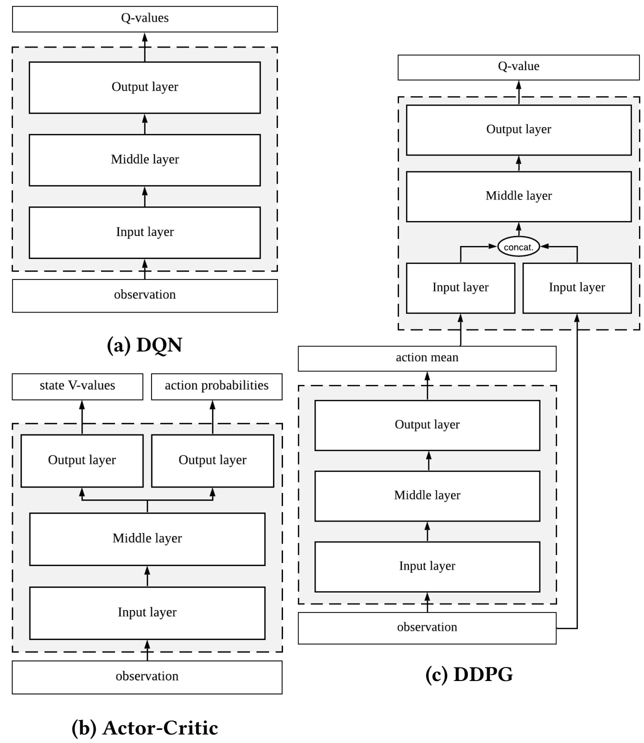

Figure 7: Example neural network architectures [118]

- **Continuous action:** The output is a single continuous value, indicating the relevance of a candidate item to the user or predicted ratings for the candidate item.
- **Slate recommendation:** In RL literature, slate recommendation focuses on recommending a whole set of items [77]. We additionally consider top-k or list recommendation, where each item is independently evaluated, as a slate recommendation. In slate recommendation, the output action is a list of discrete or continuous values. For example, the list can contain the recommended items' ids or relevance scores for the candidate items.

The action component may contain multiple post-processing step. For example, in Figure 3, the output Q-values are used for deciding on the output action, i.e., the item with the highest Q-value is selected as the output action.

5.4 Other components

RL applications also utilise the reward functions and the transitions from an old state/ observation to a new one.

Reward function: In RL-based RS applications, the reward can be defined as the traditional accuracy or ranking metrics, such as click count, precision, NDCG. For example, if the action is discrete and binary (i.e., indicating that the prediction is candidate item will be clicked/not by the target user), then the reward, e.g., click count, can be calculated by the environment by using the logs. Alternatively, a domain expert can decide the rewards heuristically, based on her experience. For example, the expert can decide to click the second item on the output list (slate) should obtain half the score of clicking the first item. Another approach is to learn/infer the reward function by observing the demonstrations, known as inverse reinforcement learning [3, 9]. It is also possible to define non-stationary rewards, where the reward may change by time [9]. For example, for an RS application users may have weights, e.g., loyal/frequent users, and their wights may change by time.

State transitions: In RL-based RS applications, state transitions actually occur in the environment (model-free RL) and/or their occurrence is modelled inside the algorithm (model-based RL). State transitions can be fixed or they can be updated in various ways depending on how states are represented. For example, in [134] the states are modelled by the user's positive interaction history and as a result, the state transition is fixed. However, let's say, the state representation utilises the similarity between the candidate item and the user history, then the update (transition) in state occurs when the user clicks on new items (update her user history) or when the features of the candidate item are updated, e.g., increase in price.

6 REINFORCEMENT LEARNING-BASED RECOMMENDERS

Significant effort has been directed to the application of RL techniques in RS to make better recommendations. In this section, we analyse the publications on RL-based RS methods. We collected the publications using "reinforcement learning" and "recommender systems" as the keywords on a scholarly literature search engine and expanded the search results using the citations. Even though our focus is on RL-based RS applications, we also included publications on advertisement whenever they are highly related to RS. As a result, we collected 56 publications (until 5 June 2020). In Table 4, the publications are presented together with the venue, the publication year and the affiliations of the authors.

The Table 4 reveals that even though research in RL-based RS started at mid-2000s, most of the research is published in the last few years, i.e., in and after 2018. The publication dates of these research work and the number of publications in those years can be listed as follow: prior-than-2018 (11), 2018 (10), 2019 (25) and 2020 (10), i.e., up to 5 June 2020. The 16/56 publications are published in arXiv, and 2 of them are published as a journal paper. The rest of them are published in conferences/workshops related to artificial intelligence, machine learning, data mining and information retrieval: KDD (5), WWW (4), WSDM (3), SIGIR (3), AAAI (2), ICDM (2), ICML (2), IJCAI (2), NeurIPS (2), RecSys (2), PAKDD (2), AAMAS (1), AIAM (1), DASFAA (1), HT (1), ICCT (1), ICEC (1), ICOIACT (1), ISMIR (1), SAC (1).

The affiliation of researchers reveals that most of the research is conducted in collaboration of industry and academia. Some of the collaborations are through an internship of researchers at the companies and some others are through funding provided by companies to the academic institutes. Among the companies, JD (.com|Data Science Lab) is the company with the highest number of publications with 8 publications. It is followed by Google (Inc.|DeepMind| Research|AI) with 5 publications and Alibaba Group with 5 publications. Among the academic institutes, most of the publications are made by the researchers at Tsinghua University and Michigan State University with 8 and 5 publications respectively. Besides the industry-academy collaborations, there are 18 research works published by researchers associated only with an academic institute.

The initial research work on RL-based RS uses more conventional methods. The RS methods using RL evolve with the advancements in artificial intelligence, machine learning, specifically in RL domain [101] and start using deep neural networks, i.e., Deep RL (DRL). While most of the DRL-based RS applications utilise a single agent, multi-task, hierarchical or multi-agent RL systems are also used. Table 5 presents the publications which use conventional (non-deep) RL methods or multi-task, hierarchical or multi-agent RL systems. The multi-task, hierarchical or multi-agent RL systems utilise more than one agent or layer to full-fill the recommendation tasks, where each agent or layer have different purposes, states or actions. For example, Feng et al [65] utilise multi-agent RL to jointly optimize ranking strategies in multiple scenarios for an e-commerce platform. Each agent (actor) has its own local observation, learns a different ranking strategy for a different scenario in the system and decides its own (private) actions. Liu et al. [136] learn multiple tasks simultaneously by training multiple RL models for each task. Zhao et al. [272] jointly learn recommendation and advertising models in two-levels. In the first level, their recommender agent generates a list of recommendations and in the second level, their advertising agent decides the location to insert an advertisement.

Each of the RL-based RS methods in the literature has different characteristics. In the upcoming sections, they are analysed in terms of (i) what they are aiming to solve, i.e, what their research questions are, (ii) how they designed their components, (iii) how they built their RL application, i.e. which frameworks they used, (iv) how they trained and evaluated their RL-based RS application.

6.1 Research questions and algorithms

The research questions of RL-based RS applications have, in general, two components which are related to (i) how input states are represented and (ii) how the output, i.e., action, is represented. The input can be represented by (a combination of) features of the user, the context, a single item or a selected set of items. The output can be item-id, rating prediction, or relevance score to the user (i.e., probability of an item being liked or clicked by the target user). The output may contain a single action, e.g., recommending a single item, or multiple actions, e.g., recommending a list of items. Some example research questions are as follow:

RQ – Given a set of items $I_t = \{i_1, i_2, ..., i_k\}$, where I_t can be all or subset of the available items, where subset may contain a single item $I_t = \{i_t\}$ (optionally, given the user u_t and/or the context c_t as the input);

- a – Which item(s) from I_t should be recommended, such that the action space is composed of item-ids?
- b – Should the item(s) from I_t be recommended, such that action space is boolean (e.g., Yes, recommend the item i_t or No, do not recommend)?
- c – What would be the rating, e.g., stars, of each item in the output item-set, which are selected from the input I_t?
- d – How much relevance has each selected item i in the output item-set, which are selected from the input I_t? Or what is the probability of being liked or clicked of each selected item i in the output item-set, which are selected from the input I_t?

Method	Publication Venue	Publ. Year	Affiliations
Bai et al. [12]	NeurIPS	2019	Stony Brook Uni., Tsinghua Uni., Uni. of Virginia
Chen et al. [34]	KDD	2018	Nanjing Uni., Alibaba Group
Chen et al. [32]	AAAI	2019	Shanghai Jiao Tong Uni., Huawei Noah's Ark Lab
Chen et al. [33]	WSDM	2019	Google, Inc.
Chen et al. [36]	ICML	2019	Georgia Tech, Ant Financial
Chen et al. [35]	arXiv	2020	Uni. of New South Wales, Uni. of Technology Sydney
Choi et al. [39]	arXiv	2018	Seoul National Uni., NAVER Clova AI Research
Dulac et al. [57]	arXiv	2015	Google DeepMind
Feng et al [65]	WWW	2018	Tsinghua National Lab for Information Science and Technology, Tsinghua Uni., Alibaba Group
Gao et al. [72]	ICDM	2019	Hubei Uni. of Technology, Wuhan Uni., Xiaomi Inc.
Gong et al. [77]	KDD	2019	Alibaba Group, Zhejiang Cainiao Supply Chain Mng., Xidian Uni., Shanghai Jiao Tong Uni.
Gui et al. [78]	SIGIR	2019	Fudan Uni.
Han et al. [81]	AIAM	2019	Shanghai Jiao Tong Uni., Harbin Institute of Technology, Huawei Noah's Ark Lab
Ie et al. [102]	IJCAI	2019	Google Research, Uni. of Texas at Austin
Ie et al. [103]	arXiv	2019	Google Research, Uni. of Texas at Austin , YouTube
Lee et al. [122]	Jrnl. of Intelligent Automation and Soft Computing	2017	Daum Kakao Corp, Sogang Uni.
Lei et al. [126]	SIGIR	2019	The Hong Kong Polytechnic Uni., Jilin Uni.
Lei et al. [125]	arXiv	2019	The Hong Kong Polytechnic Uni.
Liebman et al. [131]	AAMAS	2015	The Uni. of Texas at Austin
Liu et al. [134]	arXiv	2018	Harbin Institute of Technology, Huawei Noah's Ark Lab, Shanghai Jiao Tong Uni.
Liu et al. [137]	arXiv	2019	Nanyang Technological Uni., Shandong Uni., Alibaba Group
Liu et al. [133]	WSDM	2020	Harbin Institute of Technology, Huawei Noah's Ark Lab
Liu et al. [136]	arXiv	2020	Texas A&M Uni., Samsung Research America, National Chiao Tung Uni., Uni. of Arizona
Liu et al. [135]	PAKDD	2020	The Chinese Uni. of Hong Kong, Harbin Institute of Technology, Chinese Academy of Sciences
Ma et al. [142]	WWW	2020	Uni. of Michigan, Google AI, Simon Fraser Uni.
Mahmood et al. [144]	ICEC	2007	Uni. of Trento, Free Uni. of Bozen-Bolzano
Mahmood et al. [145]	HT	2009	Uni. of Trento, Free Uni. of Bozen-Bolzano
Munemasa et al. [159]	ICOIACT	2018	Meiji Uni., DesignOne Japan
Saebi et al. [188]	arXiv	2020	Uni. of Notre Dame
Shang et al. [199]	KDD	2019	Nanjing Uni., Didi Chuxing
Shani et al. [200]	Jrnl. of Machine Learning Research	2005	Ben-Gurion Uni., Microsoft Research
Shih et al. [206]	ISMIR	2018	National Taiwan Uni., KKBOX
Silver et al. [210]	ICML	2013	UCL, Causata Ltd.
Taghipour et al. [220]	RecSys	2007	Amirkabir Uni. of Technology
Taghipour et al. [219]	SAC	2008	Amirkabir Uni. of Technology
Takanobu et al. [221]	WWW	2019	Tsinghua Uni., Alibaba Group, State Grid Zhejiang Electric Power
Theocharous et al. [228]	IJCAI	2015	Adobe Research, UMassAmherst, INRIA
Wang et al. [241]	ICDM	2018	Microsoft Research Asia, Peking Uni., Tsinghua Uni., Hefei Uni. of Technology, Uni. of Science and Technology of China
Wang et al. [236]	PAKDD	2020	Zhejiang Uni.
Wang et al. [235]	arXiv	2020	Huazhong Uni. of Science and Technology
Xian et al. [252]	SIGIR	2019	Rutgers Uni.
Yin et al. [255]	arXiv	2019	Pennsylvania State Uni., Uni. of Arizona, Facebook
Yuyan et al. [256]	ICCT	2019	Beijing Uni. of Posts and Telecommunications
Zhang et al. [262]	NeurIPS	2019	Duke Uni., Samsung Research America, Uni. at Buffalo
Zhang et al. [261]	AAAI	2019	Renmin Uni. of China, Georgia Institute of Technology
Zhao et al. [271]	arXiv	2017	Michigan State Uni., JD.com
Zhao et al. [267]	RecSys	2018	Michigan State Uni., JD.com
Zhao et al.[270]	KDD	2018	Michigan State Uni., JD.com
Zhao et al. [268]	arXiv	2019	Michigan State Uni., JD.com
Zhao et al. [266]	arXiv	2019	Michigan State Uni., Bytedance.com
Zhao et al. [272]	arXiv	2020	Michigan State Uni., Bytedance.com
Zhao et al. [264]	arXiv	2019	Peking Uni., JD.com
Zheng et al. [273]	WWW	2018	Pennsylvania State Uni., Microsoft Research Asia
Zou et al. [278]	KDD	2019	Tsinghua Uni., JD.com
Zou et al. [279]	DASFAA	2019	Tsinghua Uni., JD.com
Zou et al. [280]	WSDM	2020	Tsinghua Uni., York Uni., Uni. of Montreal, The Uni. of Melbourne, Beijing Uni. of Posts and Telecommunications, JD Data Science Lab

Table 4: Publications on RL-based RS methods (sorted by author)

Method	Publ. Year	Type
Shani et al. [200]	2005	Conventional (Non-Deep) RL
Taghipour et al. [220]	2007	Conventional (Non-Deep) RL
Mahmood et al. [144]	2007	Conventional (Non-Deep) RL
Taghipour et al. [219]	2008	Conventional (Non-Deep) RL
Mahmood et al. [145]	2009	Conventional (Non-Deep) RL
Silver et al. [210]	2013	Conventional (Non-Deep) RL
Theocharous et al. [228]	2015	Conventional (Non-Deep) RL
Feng et al [65]	2018	Multi-agent
Gui et al. [78]	2019	Multi-agent
Shang et al. [199]	2019	Multi-agent
Takanobu et al. [221]	2019	Hierarchical
Zhang et al. [261]	2019	Hierarchical
Zhao et al. [268]	2019	Multi-agent
Zou et al. [279]	2019	Hierarchical
Zhao et al. [264]	2019	Hierarchical
Liu et al. [136]	2020	Multi-task
Ma et al. [142]	2020	Hierarchical
Zhao et al. [272]	2020	Hierarchical, Multi-task

Table 5: Conventional (Non-Deep) RL and Multi-task, hierarchical or multi-agent DRL methods (sorted by year)

The input item-set I_t can be a part of the input state representation, or it can be given as an external input/knowledge. The output item-set may contain a single item $i \in I_t$, a subset from the I_t or all the elements of I_t.

The research questions listed above have different characteristics: The first two research questions, namely RQ-a and RQ-b, can be considered as a classificaton or ranking problem, where the action is expected to be discrete. On the other hand, the third and fourth questions, namely RQ-c, RQ-d, can be considered as a regression problem, since they are expected to return continuous actions, i.e. the actions are defined as the rating or the relevance/probability values. The ones that recommend multiple (i.e., a set of) items, i.e., when output set contains more than one item, aim to make slate recommendation. All the research questions can be extended by incorporating user features and/or contextual features, such as the time of the interaction, the location of the item. As the reward function, it is possible to use accuracy or ranking based metrics or domain expert knowledge. For example, for the research questions which are expected to return a discrete result for a single item, e.g., RQ-a, it is possible to use click count as the reward. On the other hand, for the research questions which are making predictions of a list of items, it is possible to use ranking metrics, such as NDCG.

The RL-based RS applications have research questions depending on what they aim to recommend. For example, Zhao et al. [270] recommend a product, Ie et al. [102] recommend a slate of documents, or Zhao et al. [266] recommend a location for a given ad on a given list of items, i.e., where to insert the ad. In Table 6, we present the single-agent RL-based RS publications in terms of their research questions and the underlining RL algorithms. The listed RL algorithms are either directly used or inspired to develop new algorithms. We excluded the conventional (non-deep), multi-task, hierarchical or multi-agent RL systems (See Table 5).

Table 6 shows that researchers frequently aim to recommend items directly, similar to the research question RQ-a. For example, Wang et al. [235] recommend a list of items and Lei et al. [126] recommend a single (next) item. The prediction of rating scores, e.g. similar to RQ-c, is studied in a few publications, namely Lee et al. [122], Yuyan et al. [256] and Wang et al. [241]. The computation of relevance score or probability of liking/clicking, e.g, RQ-d, is used in a few publications, namely in Liu et al [134], Liu et al. [137] and Han et al. [81]. In Liu et al [134], once the action, which is represented as a vector, is selected, the ranking score is calculated by the inner product of the action and the item embedding. Then top-ranked items are recommended to the user, i.e., their approach is a combination of RQ-d and RQ-a. Liu et al. [137] compute a relevance score for items and then execute a diveersification module to decide the items to recommend. Han et al. [81] use the relevance score (e.g., weights for items) to rank the items and decide the items to recommend. Even if the other publications compute a kind of scoring, such as Q-value, they usually execute a post-processing step to decide on the items to recommend. For example, Chen et al. [34] predict the Q-value for the given state-action pair (e.g., features of the customer and the tip), similar to RQ-d, and then decides on the item (i.e, tip) to present to the user. Similarly, answering a boolean question similar to RQ-b is often used as an internal step of the algorithms. For example, Zhao et al. [267] and Zhao et al.[271] use this kind of question in Critic section of their Actor-Critic models. They feed both user's current state and action and calculates the Q-value in order to judge if the action matches with the current state. Other questions related to the quality of recommendations, such as explainability, and diversity, are also studied in some of the publications.

The Table 6 reveals that the most popular base algorithms are DQN (9), DDPG (9), REINFORCE (7) and Q-learning (6). There are a few model-based approaches, i.e., 3 publications, and the rest of the publications use model-free approaches. There is no clear preference on the type of model-free algorithms, i.e., researchers utilise Value-based, Policy-based or Actor-Critic approaches. Recently, Imitation Learning is also used, namely by Yin et al. [255] and Gong et al. [77].

6.2 Designs of the components

In the RL-based RS literature, various state representations, neural network architectures, actions and rewards are used. Most commonly used components are explained in section 5. In this section, other state representations, actions or rewards used by the current RL-based RS approaches are explained.

State representation Additional to these basic state representations explained in section 5, namely concatenation and averaging, it is possible to introduce other mechanisms. For example, Liu et al. [134] present three versions of state representations (i) DRR-p concatenates a list of items and their element-wise productions, (ii) DRR-u concatenates the element-wise productions among user and items additional to element-wise products of items, (iii) DRR-ave concatenates a single user's features, the averaged features of multiple items and their element-wise productions. Figure 8 presents the DRR-ave as an example, which uses combination of concatenation, averaging and element-wise productions. Liu et al. [133] also utilise a similar representation to DRR-ave [134] and additionally utilises an attention network to generate user-dependent weights for items.

Method	PublicationYear	Research Question	Base-Algorithm (Algorithm Family)
Bai et al. [12]	2019	RQ-a	Model-based, REINFORCE (Policy opt.)
Chen et al. [34]	2018	RQ-a	Double DQN (Value Opt.)
Chen et al. [32]	2019	RQ-a	REINFORCE (Policy opt.)
Chen et al. [33]	2019	RQ-a	REINFORCE (Policy opt.)
Chen et al. [36]	2019	RQ-a	Model-based, DQN (Value opt.)
Chen et al. [35]	2020	RQ-a	DDPG (Actor-Critic)
Choi et al. [39]	2018	RQ-a, Explainability	Q-learning, SARSA (Value opt.)
Dulac et al. [57]	2015	RQ-a	DDPG (Actor-Critic)
Gao et al. [72]	2019	RQ-a	DQN (Value opt.)
Gong et al. [77]	2019	RQ-a	Imitation Learning, REINFORCE (Policy opt.)
Han et al. [81]	2019	RQ-d	DDPG (Actor-Critic)
Ie et al. [102]	2019	RQ-a	TD-learning, Q-learning (Value opt.)
Ie et al. [103]	2019	RQ-a	Q-learning, SARSA (Value opt.)
Lee et al. [122]	2017	RQ-c	Q-learning (Value opt.)
Lei et al. [126]	2019	RQ-a	DQN (Value opt.)
Lei et al. [125]	2019	RQ-a	DQN (Value opt.)
Liebman et al. [131]	2015	RQ-a	Model-based, MCTS (Policy-opt.)
Liu et al. [134]	2018	RQ-d	DDPG (Actor-Critic)
Liu et al. [137]	2019	RQ-d, Diversity	DDPG (Actor-Critic)
Liu et al. [133]	2020	RQ-a	DQN (Value opt.), DDPG (Actor-Critic)
Liu et al. [135]	2020	RQ-a, Fairness	Actor-Critic
Munemasa et al. [159]	2018	RQ-a	DDPG (Actor-Critic)
Saebi et al. [188]	2020	RQ-a, Explainability	REINFORCE (Policy opt.)
Shih et al. [206]	2018	RQ-a	Policy Gradient
Wang et al. [241]	2018	RQ-c, Explainability	REINFORCE (Policy opt.)
Wang et al. [236]	2020	RQ-a	Q-Learning (Value opt.)
Wang et al. [235]	2020	RQ-a	DDPG (Actor-Critic)
Xian et al. [252]	2019	RQ-a, Explainability	REINFORCE (Policy opt.)
Yin et al. [255]	2019	RQ-a	Imitation Learning, PPO (Policy opt.)
Yuyan et al. [256]	2019	RQ-c	DQN (Value opt.)
Zhang et al. [262]	2019	RQ-a	Constrained MDP (Policy opt.)
Zhao et al. [271]	2017	RQ-a	DDPG (Actor-Critic)
Zhao et al. [267]	2018	RQ-a	DDPG (Actor-Critic)
Zhao et al.[270]	2018	RQ-a	DQN (Value opt.)
Zhao et al. [266]	2019	RQ-a	DQN (Value opt.)
Zheng et al. [273]	2018	RQ-a	DQN (Value opt.)
Zou et al. [278]	2019	RQ-a, Diversity	Q-Learning (Value opt.)
Zou et al. [280]	2020	RQ-a	Dyna-Q (Model-based)

Table 6: Research questions and base-algorithms used by single-agent DRL-based RS methods

Liu et al. [135] includes fairness to the state representation. Their state is composed of user preference status and fairness status, which is used for promoting items from under-represented groups. Gap et al. [72] and Zhao et al.[270] represent states not only by previous N items that a user interacted, but also previous N items that the user skipped. Choi et al. [39] represent the environment as a grid-world by executing biclustering technique. They create n^2 distinct states each of which is composed of users and items (Figure 9). The user movement in the grid-world is used while making recommendations. Gao et al [72] uses convolution filters to capture and combine users' sequential interactions. Lei et al. [126] combines the embeddings of neighbors with the embedding of the target user for the state representation. The knowledge graph-based applications, such as Xian et al. [252] and Saebi et al. [188], use the edges and nodes while constructing the states. They use starting node, current node, k-step history of nodes and edges visited or queried relation, e.g., edge type, to represent the states. Wang et al. [235] combines textual data obtained from reviews and description of the items with the historical preferences of the users to represent the states.

Many of the algorithms utilise RNN based techniques to update the state vector representations, such as features on browsing history. For example, Bai et al. [12], Gong et al. [77] and Shih et al. [206] utilise RNN, Yin et al. [255], Zhao et al. [267] and Zhao et al. [266] use gated recurrent unit (GRU), Zhang et al. [262] and Zou et al. [278] use LSTM, Chen et al. [32] uses simple recurrent unit (SRU) [124] and Chen et al. [33] uses Chaos Free RNN (CFN) [119] in order to integrate previous states with current

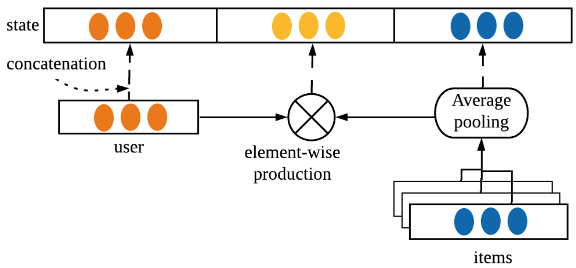

Figure 8: Liu et al. [134]'s DRR_ave state representation

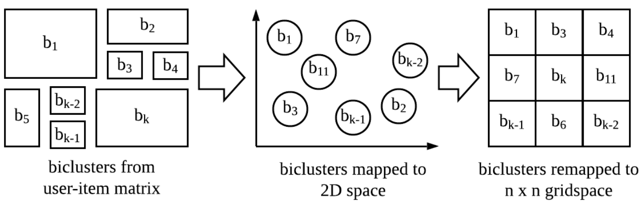

Figure 9: Choi et al. [39]'s state representation

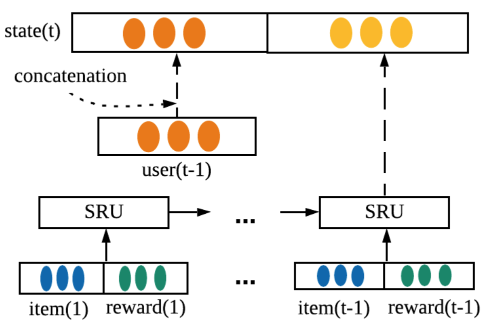

Figure 10: Chen et al. [32]'s state representation

states. An example for an RNN-based representation is presented in Figure 10. It shows the representation from Chen et al. [32] and concatenates the embedding of the historical interactions with the user status information. Each historical interaction is represented by the prior item-reward pairs and trained by a recurrent neural network. Zou et al. [280] observe that users' preferences are dependent on both recent and earlier items and RNN-based methods are not good at dealing with such long-term dependency. Instead, they propose another technique, namely State Tracker, which uses memory network and self-attention mechanisms to encode both the long-term and temporary preferences of the users.

Action The actions of RS applications may have various forms, as explained in section 5, predicting if an item will be clicked/not (a discrete action), predicting the rating the user would give to the target item (continuous action), predicting the probability of how much user will like an item (a continuous action) or presenting a list of recommended items (a slate recommendation).

In the literature, not all of the recommendations are directly made for the items to be clicked/liked, e.g., item-id. It is possible to use 'non-item' components, such as textual data, graph components (nodes and edges). For example, Chen et al. [34] recommend 'tips', i.e., keywords to refine the search queries, Yin et al. [255] recommend 'search stories'. Wang et al. [241] provide explanations for the recommended items by choosing a subset of interpretable components. Zhao et al. [266] decide the location for an advertisement given a recommendation list. The applications using knowledge graphs, namely Xian et al. [252] and Saebi et al. [188], define their actions as either staying at the current node or selecting an edge and moving to another node. Gong et al. [77] recommend K items (cards), by defining the input space as a graph representation and actions as finding cliques. Zhao et al. [267] recommend a page of items, such that they generate a list of complementary items and present them in a two-dimensional page. Choi et al. [39] represent the environment as a grid-world (See Figure 9) and defines actions as the movements around the grid-world, i.e., up, down, left, right.

RS applications usually have huge action space, e.g., in the order of millions [33]. The publications aim to solve this problem using various techniques. Zhao et al. [266] define a null action in addition to the existing actions, indicating that the item/advertisement will not be shown or it is not chosen. Both Ie et al. [102] and Ie et al. [103], which make slate recommendation, add an implicit null item to each and every slate, for the case of none of the items is chosen. Chen et al. [33] and Dulac et al. [57] use a nearest neighbour algorithm to retrieve candidate actions. Chen et al. [32] construct a balanced tree where leaf nodes represent the items and non-leaf nodes are used for making top-down decisions to decide which branch to use. Chen et al [36] design a cascading Q-networks. Liebman et al. [131], which make playlist recommendation, cluster songs by their type.

Reward The commonly used reward functions by RL-based RS applications are computed by (i) the accuracy or ranking metrics, such as click count, purchase count, precision, NDCG, (ii) the information on time, such as time spent on scrolling, dwell time or (iii) manually assigned rewards, by domain experts.

Some of the RL-based RS applications combine existing reward functions or define new reward functions in order to obtain better representation of the reward. For example, Chen et al. [34], Zhao et al. [266], Zheng et al. [273], Zou et al. [278] and Zou et al. [280] combine the outcome of the recommendation, such as click, purchase, income, with the user behaviours, such as page scrolling, activeness, leaving application. Chen et al. [32] combine the empirical reward, i.e., the rating, with the sequential reward, i.e., the number of consecutive positive and negative feedbacks. Liebman et al. [131], which make playlist recommendations, combine reward for each song, i.e., user's preference on each song in isolation, and reward for the sequence of songs, i.e., transition from the previously listened songs to the current song. Lee et al. [122] define their reward function as the difference of the predicted rating of the RL method from the the predicted rating of the base predictor. Shih et al. [206] combine rewards calculated according to the diversity, novelty, freshness and coherence of the recommendations. Liu et al. [135] uses fairness in addition to accuracy in their reward computation. Their reward function gives high positive rewards when the user interacts with the recommended item and the item belongs to an under-representing group. Wang et al. [241], which make explanations on recommended items, combine explainability of the item and presentation quality (i.e., readability and consistency) of the output explanations. Wang et al. [235] state that users usually do not give any explicit negative feedback, but just skip the unliked item. As a result, in their reward function, tin additionto explicit positive and negative feedbacks they include information of the other (non-explicit) negative interactions. Zhang et al. [262] which recommend visual items utilise visual similarity of the recommended and the expected items. The knowledge graph-based applications, namely Xian et al. [252] and Saebi et al. [188], utilise the features of the edges and nodes and/or their similarities.

Most of the current RL-based RS applications, e.g., [33, 72, 102, 103, 255, 267, 270, 271], utilise static rewards, whose computation does not change by time. However it is possible to have dynamic reward functions that change by time [9], for example as users interact with the system, it is possible to update their weight/attention. Choi et al. [39] use Jaccard distance of user vectors of two states s_t and s_{t+1} as the reward function. As the state representations are updated, the output of the reward function updates as well. Liebman et al. [131] calculate the reward per user, such that each user has a unique reward function. Also, the weights used by the components of the reward functions are updated as each user's state representation is updated. Zhao et al. [271] use the similarity between the current and the historical states and the similarity between the actions. As the states update, the output reward is updated.

6.3 Training and evaluation

Building RL-based application needs various modules and/or frameworks, as explained in section 4. They have different settings on how they obtain the samples, how they train and evaluate the models. The preferences of the current RL-based RS methods in the literature are presented in Table 8.

The Table 8 shows that only a few of the publications use an existing gym or environment implementations. Choi et al. [39] uses OpenAI Gym [25] with a public dataset, Ie et al. [102] and Ie et al. [103] use RecSim [101] with private datasets. Moreover, none of the applications, except Ie et al. [102] and Ie et al. [103], utilise existing RL frameworks for training. This can be related to the fact that most of the research works aim to propose their own algorithm. However, utilising an existing framework and implementing the new algorithms on those frameworks is useful, both for the researchers who aim to reproduce the algorithms and their results and for the researchers who are proposing and implementing new algorithms. The list of publications that use public datasets and the name of those datasets are presented in Table 7. There are few publications which utilise both public and private datasets. Many of the research work, 16 of them, is done only on private datasets.

We observe from Table 8 that researchers usually do not prefer to run the experiments in real-world. Even though evaluation on real-world systems makes evaluation easier by letting the users directly interact with the new RL model, it is too risky and impractical for a commercial application [134]. As a result, it is common to use simulation or logs of previous interactions (e.g., benchmark datasets). The ones that use simulators train their simulators using logs from real-world, benchmark datasets or synthetic data based on statistical features. In Table 8, we listed the evaluation settings as Sim. (simulation) when the simulation is responsive to the unseen actions, e.g., if the user never interacted with the recommended item, the reward is computed/modelled rather than directly using the logs. If the information from the log is directly used, we listed the evaluation setting as Logs. While 17 of the publications use offline evaluation setting only, the remaining publications use online setting using either a simulator or real-world as the evaluation environment. Among all the publications, 10 of them utilise both online and offline evaluation settings and 3 of them use online settings both with simulation and real-world. Most of the experiments are run in fixed evaluation fashion, such that the hyper-parameters of the algorithms are not updated during the test period, usually for the comparison with baseline supervised learning methods. There are a few publications that utilise both updating and fixed evaluation settings. For example, Chen at al. [34] executes both fixed and updating versions of the proposed algorithm, namely off-DL and on-DL respectively, to observe the efficiency of RL over offline or online supervised learning.

In terms of evaluation metrics, most of the applications use either reward or common information retrieval and machine learning metrics, such as click-through rate (CTR), precision, recall, F1-measure, hit-rate, NDCG, MAP, diversity, explainability, novelty or time-related information, such as time spent on training and execution.

Method	Public Datasets
Bai et al. [12]	CIKM Cup 2016
Chen et al. [32]	MovieLens-10M, Netflix
Chen et al. [36]	MovieLens, LastFm, Yelp, Taobao, RecSys15YooChoose, Ant Financial News
Chen et al. [35]	Book-Crossing, MovieLens-20M, Librarything, Amazon CDs and Vinyl, Netflix Prize, Goodreads
Choi et al. [39]	MovieLens-100K, MovieLens-1M
Gong et al. [77]	MovieLens-100K
Han et al. [81]	MovieLens-100K, MovieLens-1M
Lee et al. [122]	MovieLens-10M
Lei et al. [126]	LastFm, Ciao, Epinions
Lei et al. [125]	MovieLens-100K, MovieLens-1M, MovieLens-10M
Liebman et al. [131]	Yes.com, The Art of the Mix
Liu et al. [134]	MovieLens-100K, MovieLens-1M, Yahoo! Music (R3), Jester (2)
Liu et al. [137]	MovieLens-100K, MovieLens-1M
Liu et al. [133]	MovieLens-100K, MovieLens-1M, Jester (2)
Liu et al. [135]	MovieLens-100K
Saebi et al. [188]	NELL-995, Amazon Beauty, Amazon Cellphones
Wang et al. [241]	Amazon Toys and Games, Yelp 2018 LasVegas
Wang et al. [236]	MovieLens-1M, MovieLens-10M, Amazon Grocery and Gourmet Food
Wang et al. [235]	Amazon Music, Amazon Beauty, Amazon Clothing
Xian et al. [252]	Amazon CDs and Vinyl, Amazon Clothing, AmazonCell Phones, Amazon Beauty
Yin et al. [255]	JD.com
Yuyan et al. [256]	MovieLens-100K, MovieLens-1M
Zhang et al. [262]	UT-Zappos50K
Zou et al. [280]	Taobao, Retailrocket

Table 7: The public datasets which are used by single-agent DRL-based RS methods

Method	Gym/ Environment	Training Framework	Dataset Type	Evaluation Setting
Bai et al. [12]	Custom	Custom	Public, Private	Offline (Logs), Online(Sim.)
Chen et al. [34]	Custom	Custom	Private	Offline (Logs), Online (Real-world)
Chen et al. [32]	Custom	Custom	Public	Offline (Logs)
Chen et al. [33]	Custom	Custom	Private	Online (Sim.), Online (Real-world)
Chen et al. [36]	Custom	Custom	Public	Online (Sim.)
Chen et al. [35]	Custom	Custom	Public	Online (Sim.)
Choi et al. [39]	OpenAI Gym	Custom	Public	Offline (Logs)
Dulac et al. [57]	Custom	Custom	Private	Online (Sim.)
Gao et al. [72]	Custom	Custom	Private	Offline (Logs)
Gong et al. [77]	Custom	Custom	Public, Private	Offline (Logs)
Han et al. [81]	Custom	Custom	Public	Online (Sim.)
Ie et al. [102]	RecSim	Dopamine	Private	Online (Sim.), Online (Real-world)
Ie et al. [103]	RecSim	Dopamine	Private	Online (Sim.), Online (Real-world)
Lee et al. [122]	Custom	Custom	Public	Offline (Logs)
Lei et al. [126]	Custom	Custom	Public	Offline (Logs)
Lei et al. [125]	Custom	Custom	Public	Offline (Logs)
Liebman et al. [131]	Custom	Custom	Public	Offline (Logs) , Online (Real-world)
Liu et al. [134]	Custom	Custom	Public	Offline (Logs), Online (Sim.)
Liu et al. [137]	Custom	Custom	Public	Offline (Logs), Online (Sim.)
Liu et al. [133]	Custom	Custom	Public	Offline (Logs), Online (Sim.)
Liu et al. [135]	Custom	Custom	Public, Private	Offline (Logs)
Munemasa et al. [159]	Custom	Custom	Private	Offline (Logs)
Saebi et al. [188]	Custom	Custom	Public	Offline (Logs)
Shih et al. [206]	Custom	Custom	Private	Offline (Logs)
Wang et al. [241]	Custom	Custom	Public	Offline (Logs)
Wang et al. [236]	Custom	Custom	Public	Offline (Logs), Online (Sim.)
Wang et al. [235]	Custom	Custom	Public	Online (Sim.)
Xian et al. [252]	Custom	Custom	Public	Offline (Logs)
Yin et al. [255]	Custom	Custom	Public	Offline (Logs)
Yuyan et al. [256]	Custom	Custom	Public	Offline (Logs)
Zhang et al. [262]	Custom	Custom	Public	Online (Sim.)
Zhao et al. [271]	Custom	Custom	Private	Online (Sim.)
Zhao et al. [267]	Custom	Custom	Private	Offline (Logs), Online (Sim.)
Zhao et al.[270]	Custom	Custom	Private	Offline (Logs), Online (Sim.)
Zhao et al. [266]	Custom	Custom	Private	Offline (Logs)
Zheng et al. [273]	Custom	Custom	Private	Offline (Logs), Online (Real-world)
Zou et al. [278]	Custom	Custom	Private	Offline (Logs)
Zou et al. [280]	Custom	Custom	Public	Online (Sim.)

Table 8: Training and evaluation settings used by single-agent DRL-based RS methods

Besides these metrics, many other RL-based RS applications define new evaluation metrics. Chen et al. [34] define UV CTR which is the ratio of the number of customers who click on the item to the number of total customers who view it. Yin et al. [255] define conversion rate (CVR) which is the ratio of sessions with at least one product ordered to the total number of sessions. Liu et al. [135] measure CVR and Weighted Proportional Fairness (PropFair) metrics. They also introduce Unit Fairness Gain (UFG) metric which jointly considers accuracy and fairness. Zou et al. [278] measure the average cumulative number of clicks, the average browsing depth and the average revisiting days between consecutive visits of a user. Zou et al. [280] utilise the average clicks per trajectory, average diversity of recommendations and the average number of interactions as their evaluation metrics. Zhang et al. [262] evaluate their proposed algorithm on text-based interactive recommendation and text generation. For the recommendation task, they measure the task success rate (SR) after K interactions, the number of interactions (NI) before success and the number of violated attributes (NV). Ie et al. [102] and Ie et al. [103] make slate recommendation and they measure the average return per user session, which is the improvement in percentage relative to a random algorithm, the average quality of the output documents and page CTR, which is the ratio of the total clicks of the page to the number of impressions of the page. Wang et al. [241] measure the efficiency of explanations for the recommended items and the consistency of the explanations with the ratings given by the user.

7 INSIGHTS AND DISCUSSION

In this survey paper, we aimed to present various aspects of RL-based RS applications, from the issues and challenges to how to design and build RL-based RS applications. We present the highlights of the sections as follow:

- Both RS and RL systems utilise the information collected from an environment (e.g., user, item, context features) to decide which action(s) to take (e.g., recommending products) in order to maximise the overall performance (e.g., click-through rate, accuracy) [203].
- There are various challenges that researchers encounter while developing and deploying RL methods in real-world systems, such as high dimensional states and action spaces, reproducibility issues, partial observability [59, 129]. RS systems are not an exception to these issues and challenges. These systems also suffer from domain-specific challenges such as cold-start users/items, long-tail users, fairness, explainability, diversity, privacy and security [22, 101, 182]. While developing RL-based RS applications these kinds of RS-specific challenges should also be taken into account.
- Building an RL application requires multiple components for (i) obtaining the samples from some environment, (ii) implementing the RL agent and training the model (iii) evaluating the model. Utilising existing frameworks in each step helps developers/researchers to implement the RL applications easier [181] and allows them to reproduce existing works.
- The design of the RL-based RS applications can be summarised in a unified architecture. We proposed that the components of the unified architecture can naively follow the base idea of MDPs, such that the architecture can be represented by the state, action, policy components and complemented by reward and transition components. Each of these components can have various designs. For example, states can be represented simply by concatenation of features of users and items or they can be represented by more complex functions, such as RNNs or graph representations.
- The research work in RL-based RS domain gains attention not only from academia but also from industry. Even though research in RL-based RS started at the mid-2000s, most of the research is published in the last few years, i.e., in and after 2018.
- One can define various research questions while applying RL in RS domain, such as predicting ratings, selecting (a list of) items. However, we observed that the common focus for many of the research work is on finding the items (e.g., item ids) to recommend.
- Most of the research work in RL-based RS domain implement their own environments and RL algorithms, instead of using the existing frameworks. This can be related to the fact that most of the research works aim to propose their own algorithm. However, utilising an existing framework and implementing the new algorithms on those frameworks is useful, both for the researchers who propose and implement new algorithms and for researchers who reproduce the existing algorithms and their results.
- Most of the research work uses private datasets. Having private datasets in addition to custom environment and agent implementations, it is highly challenging to reproduce the existing RL-based RS applications and their results.
- For the experiments, the researchers prefer to use simulation or logs of previous interactions. Most of the experiments are run in fixed evaluation fashion, such that the hyper-parameters of the algorithms are not updated during the test period.

Each section revealed different characteristics that need attention while developing and deploying RL-based RS applications. Here we highlight four aspects that we think are important and needs further research and development.

- **Reproducibility**: Reproducing results of RL methods is usually challenging because they utilise various extrinsic and intrinsic settings, such as hyper-parameters, network architectures, codebases, random seeds, environment properties [90]. Unfortunately, the current research works in RL-based RS usually implement their custom environments and agents. Additionally, most of them use private datasets. The ones using public datasets do not necessarily explain the preprocessing steps of their experimental setting. For example, some of them split their datasets randomly or prune the datasets based on some heuristics, which are not detailed in the publications. All the above-mentioned problems make it nearly impossible to reproduce their implementations and results. Utilising an existing framework for the environments (such as OpenAI Gym [25], RecSim [101]) and the agents (such as Coach [29], RLLib [130]) is useful, both for the researchers who aim to reproduce existing algorithms (and their results) and for the researchers who are proposing and implementing new algorithms. Also, producing and sharing benchmark implementations together with the publicly accessible data (i.e., training, validation, testing sets), hyper-parameters and evaluation metrics is important for having reproducible results.
- **Modularisation**: It is not easy to provide complete and simple RL implementations [130, 202]. Having high-level abstractions both for environments and agent is important to be able to have tools which are easy to extend, combine and use. Current RL frameworks have a tendency to implement RL agents as single strongly connected processes [130, 202]. Modular tools (i.e., in terms of environment, agent and RL component implementations) are necessary both for reproducing the existing algorithms and their results and for proposing and implementing new algorithms easily. Each of these components should be implemented independently and should use software development paradigms, such as programming to interfaces, as much as possible. For example, a researcher should be able to use state representation from one tool and policy component (i.e., neural network architecture) from another tool with minimal effort and without any problem.
- **RS-specific problems**: In addition to the issues and challenges related to the application of RL, RS domain has its more specific problems, such as cold-start users/items, long-tail users, fairness, explainability, diversity, privacy and security problems [22, 101, 182]. In the RL-based RS literature, some of these problems are already gained attention from researchers. For example, Choi et al. [39], Saebi et al. [188], Wang et al. [241], Xian et al. [252] focus

on explainability in addition to accuracy of recommendations. However, the other aspects, such as fairness or the long tail users, did not get enough attention, yet. RS domain has already aimed to solve many of these problems for applications in various domains, such as POI, music track, product recommendation. The experience obtained from the previous RS publications should be integrated with RL-based RS applications.

- **Diversity of research questions**: The RS-specific problems are highly related to how the research questions are formed. In addition to these issues and challenges, presenting rating based predictions or the relevance score (i.e., probability of liking an item) didn't get much attention from the RL-based RS literature, yet. These kinds of scoring-based outputs can be integrated with approaches focusing on RS-specific problems and may produce better recommendations. For example, the calculated probability of liking an item can be integrated with explainability to produce both more accurate and more interpretable recommendations.

8 CONCLUSION

The reinforcement learning (RL) systems are capable of adapting to new situations and learning from interactions with the real-world [10]. RL methods are shown to be effective in various scenarios, such as in games, robotics, process systems and biochemical systems. However, application and deployment of RL methods on other real-world problems, such as recommender systems, remain limited [59, 101, 105, 129]. Recommender systems (RSs) estimate the future preferences of the users based on their previous interactions [182]. RL and RS applications are similar, such that both utilise information obtained from their environment (e.g., user, item, context features) to execute an action (e.g., recommending products) in order to maximise the overall performance (e.g., click-through rate, accuracy) [203]. While developing and deploying RL methods in real-world systems, specifically the RL-based RS applications, researchers need to deal with various issues and challenges emerging from the characteristics of RL and RS, such as high dimensional state and action spaces, partial observability, reproducability [59, 129]. Moreover, the researchers need to design the agent and the environment, decide on the data source which provides the samples and the reward function which is used for getting the feedback (or implement the related module to infer the reward function) and implement all these components either from scratch or using existing training and evaluation frameworks. In this survey paper, after introducing RL and RS systems, we gave details on the issues and challenges observed while applying RL methods in real-world applications, specifically in RS. We explained how the RL algorithms are built (i.e., how samples are obtained, how the agents are trained and how they are evaluated) and how RL-based RS applications are designed (i.e., how the states, actions, rewards are represented). We gave details on the current RL-based RS algorithms in the literature and highlighted their choices on design of the components, RL algorithms, training and evaluation settings. We finished the survey paper with a short discussion section which highlights the shortcomings of the current literature and gives insights for future work.

We highlighted four aspects which are important and needs further research and development, namely reproducibility, modularisation, RS-specific problems and diversity in research questions. It is important to re-use existing environment and agent implementations and to create benchmark implementations with public datasets, specifically for reproducibility. Having modular implementations of environments, algorithms and components of RL applications is also important for developing new algorithms as well as re-producing results of the state-of-the-art methods. RS domain has already aimed to solve various problems, such as cold start users/items problem, for different applications, such as POI, music track, product recommendation. The experience obtained from those applications should be integrated with RL-based RS applications. Also, instead of focusing only on recommending items, i.e., item ids, various research questions can be solved and combined with RS-specific questions to make better recommendations. For example, the calculated probability of liking an item can be integrated with explainability to produce both more accurate and more interpretable recommendations.

REFERENCES

[1] 2017. *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net. https://openreview.net/group?id=ICLR.cc/2017/conference

[2] 2020. *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net. https://openreview.net/group?id=ICLR.cc/2020/Conference

[3] Pieter Abbeel and Andrew Y Ng. 2004. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*. 1.

[4] David Abel. 2019. simple rl: Reproducible Reinforcement Learning in Python. In *ICLR Workshop on Reproducibility in Machine Learning*.

[5] Josh Achiam and Pieter Abbeel. 2018. Spinning Up in Deep RL-Introduction to RL. https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html.

[6] Alekh Agarwal, Sarah Bird, Markus Cozowicz, Luong Hoang, John Langford, Stephen Lee, Jiaji Li, Dan Melamed, Gal Oshri, Oswaldo Ribas, et al. 2016. Making contextual decisions with low technical debt. *arXiv preprint arXiv:1606.03966* (2016).

[7] Rishabh Agarwal, Chen Liang, Dale Schuurmans, and Mohammad Norouzi. 2019. Learning to Generalize from Sparse and Underspecified Rewards. In *International Conference on Machine Learning*. 130–140.

[8] Jose A Arjona-Medina, Michael Gillhofer, Michael Widrich, Thomas Unterthiner, Johannes Brandstetter, and Sepp Hochreiter. 2019. Rudder: Return decomposition for delayed rewards. In *Advances in Neural Information Processing Systems*. 13544–13555.

[9] Saurabh Arora and Prashant Doshi. 2018. A survey of inverse reinforcement learning: Challenges, methods and progress. *arXiv preprint arXiv:1806.06877* (2018).

[10] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. 2017. A brief survey of deep reinforcement learning. *arXiv preprint arXiv:1708.05866* (2017).

[11] Mohammad Gheshlaghi Azar, Vicenç Gómez, and Hilbert J Kappen. 2012. Dynamic policy programming. *Journal of Machine Learning Research* 13, Nov (2012), 3207–3245.

[12] Xueying Bai, Jian Guan, and Hongning Wang. 2019. A Model-Based Reinforcement Learning with Adversarial Training for Online Recommendation. In *Advances in Neural Information Processing Systems*. 10734–10745.

[13] David Balduzzi, Karl Tuyls, Julien Perolat, and Thore Graepel. 2018. Re-evaluating evaluation. In *Advances in Neural Information Processing Systems*. 3268–3279.

[14] Linas Baltrunas and Xavier Amatriain. 2009. Towards time-dependant recommendation based on implicit feedback. In *Workshop on context-aware recommender systems (CARS'09)*. Citeseer, 25–30.

[15] Andre Barreto, Diana Borsa, John Quan, Tom Schaul, David Silver, Matteo Hessel, Daniel Mankowitz, Augustin Zidek, and Remi Munos. 2018. Transfer in Deep Reinforcement Learning Using Successor Features and Generalised Policy Improvement. In *International Conference on Machine Learning*. 501–510.

[16] André Barreto, Will Dabney, Rémi Munos, Jonathan J Hunt, Tom Schaul, Hado P van Hasselt, and David Silver. 2017. Successor features for transfer in reinforcement learning. In *Advances in neural information processing systems*. 4055–4065.

[17] C Beattie, JZ Leibo, D Teplyashin, T Ward, M Wainwright, H Küttler, A Lefrancq, S Green, V Valdés, A Sadik, et al. [n.d.]. Deepmind lab. arXiv 2016. *arXiv preprint arXiv:1612.03801* ([n. d.]).

[18] Robert M. Bell and Yehuda Koren. 2007. Lessons From the Netflix Prize Challenge. *ACM SIGKDD Explorations Newsletter* 9, 2 (2007), 75–79.

[19] Marc G Bellemare, Will Dabney, and Rémi Munos. 2017. A distributional perspective on reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 449–458.

[20] Marc G Bellemare, Georg Ostrovski, Arthur Guez, Philip S Thomas, and Rémi Munos. 2016. Increasing the action gap: New operators for reinforcement learning. In *Thirtieth AAAI Conference on Artificial Intelligence*.

[21] Yoshua Bengio and Yann LeCun (Eds.). 2016. *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*. https://iclr.cc/archive/www/doku.php%3Fid=iclr2016:accepted-main.html

[22] Jesús Bobadilla, Fernando Ortega, Antonio Hernando, and Abraham Gutiérrez. 2013. Recommender systems survey. *Knowledge-based systems* 46 (2013), 109–132.

[23] Djallel Bouneffouf and Irina Rish. 2019. A survey on practical applications of multi-armed and contextual bandits. *arXiv preprint arXiv:1904.10040* (2019).

[24] Craig Boutilier, Alon Cohen, Avinatan Hassidim, Yishay Mansour, Ofer Meshi, Martin Mladenov, and Dale Schuurmans. 2018. Planning and learning with stochastic action sets. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*. 4674–4682.

[25] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. Openai gym. *arXiv preprint arXiv:1606.01540* (2016).

[26] Jacob Buckman, Danijar Hafner, George Tucker, Eugene Brevdo, and Honglak Lee. 2018. Sample-efficient reinforcement learning with stochastic ensemble value expansion. In *Advances in Neural Information Processing Systems*. 8224–8234.

[27] David Budden, Matteo Hessel, John Quan, and Steven Kapturowski. 2020. *RLax: Reinforcement Learning in JAX*. http://github.com/deepmind/rlax

[28] Robin Burke. 2002. Hybrid recommender systems: Survey and experiments. *User modeling and user-adapted interaction* 12, 4 (2002), 331–370.

[29] Itai Caspi, Gal Leibovich, Gal Novik, and Shadi Endrawis. 2017. Reinforcement Learning Coach. https://doi.org/10.5281/zenodo.1134899

[30] Pablo Samuel Castro, Subhodeep Moitra, Carles Gelada, Saurabh Kumar, and Marc G Bellemare. 2018. Dopamine: A research framework for deep reinforcement learning. *arXiv preprint arXiv:1812.06110* (2018).

[31] Yash Chandak, Georgios Theocharous, Blossom Metevier, and Philip S Thomas. 2019. Reinforcement learning when all actions are not always available. *arXiv preprint arXiv:1906.01772* (2019).

[32] 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 *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 3312–3320.

[33] 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 *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*. 456–464.

[34] 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 *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 1187–1196.

[35] Xiaocong Chen, Chaoran Huang, Lina Yao, Xianzhi Wang, Wei Liu, and Wenjie Zhang. 2020. Knowledge-guided Deep Reinforcement Learning for Interactive Recommendation. *arXiv preprint arXiv:2004.08068* (2020).

[36] 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 *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA (Proceedings of Machine Learning Research)*, Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.), Vol. 97. PMLR, 1052–1061. http://proceedings.mlr.press/v97/chen19f.html

[37] Chen Cheng, Haiqin Yang, Irwin King, and Michael R Lyu. 2012. Fused matrix factorization with geographical and social influence in location-based social networks. In *Twenty-sixth AAAI conference on artificial intelligence*.

[38] Zhiyong Cheng, Jialie Shen, Lei Zhu, Mohan S Kankanhalli, and Liqiang Nie. 2017. Exploiting Music Play Sequence for Music Recommendation.. In *IJCAI*, Vol. 17. 3654–3660.

[39] 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).

[40] Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. 2018. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Advances in Neural Information Processing Systems*. 4754–4765.

[41] Jack Clark and Dario Amodei. 2016. Faulty Reward Functions in the Wild. https://openai.com/blog/faulty-reward-functions/.

[42] Karl Cobbe, Oleg Klimov, Chris Hesse, Taehoon Kim, and John Schulman. 2019. Quantifying Generalization in Reinforcement Learning. In *International Conference on Machine Learning*. 1282–1289.

[43] Felipe Codevilla, Matthias Miiller, Antonio López, Vladlen Koltun, and Alexey Dosovitskiy. 2018. End-to-end driving via conditional imitation learning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 1–9.

[44] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM conference on recommender systems*. 191–198.

[45] Will Dabney, Mark Rowland, Marc G Bellemare, and Rémi Munos. 2018. Distributional reinforcement learning with quantile regression. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

[46] Maurizio Ferrari Dacrema, Paolo Cremonesi, and Dietmar Jannach. 2019. Are we really making much progress? A worrying analysis of recent neural recommendation approaches. In *Proceedings of the 13th ACM Conference on Recommender Systems*. 101–109.

[47] Abhinandan S Das, Mayur Datar, Ashutosh Garg, and Shyam Rajaram. 2007. Google news personalization: scalable online collaborative filtering. In *Proceedings of the 16th international conference on World Wide Web*. 271–280.

[48] James Davidson, Benjamin Liebald, Junning Liu, Palash Nandy, Taylor Van Vleet, Ullas Gargi, Sujoy Gupta, Yu He, Mike Lambert, Blake Livingston, et al. 2010. The YouTube video recommendation system. In *Proceedings of the fourth ACM conference on Recommender systems*. 293–296.

[49] Peter Dayan and Yael Niv. 2008. Reinforcement learning: the good, the bad and the ugly. *Current opinion in neurobiology* 18, 2 (2008), 185–196.

[50] Esther Derman, Daniel J. Mankowitz, Timothy A. Mann, and Shie Mannor. 2018. Soft-Robust Actor-Critic Policy-Gradient. In *Proceedings of the Thirty-Fourth Conference on Uncertainty in Artificial Intelligence, UAI 2018, Monterey, California, USA, August 6-10, 2018*, Amir Globerson and Ricardo Silva (Eds.). AUAI Press, 208–218. http://auai.org/uai2018/proceedings/papers/70.pdf

[51] Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, Yuhuai Wu, and Peter Zhokhov. 2017. OpenAI Baselines. https://github.com/openai/baselines.

[52] Alexey Dosovitskiy and Vladlen Koltun. 2017. Learning to Act by Predicting the Future, See [1]. https://openreview.net/forum?id=rJLS7qKel

[53] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. 2017. CARLA: An Open Urban Driving Simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*. 1–16.

[54] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. 2016. Benchmarking deep reinforcement learning for continuous control. In *International Conference on Machine Learning*. 1329–1338.

[55] Miroslav Dudík, John Langford, and Lihong Li. 2011. Doubly robust policy evaluation and learning. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*. 1097–1104.

[56] Gabriel Dulac-Arnold, Ludovic Denoyer, Philippe Preux, and Patrick Gallinari. 2012. Fast reinforcement learning with large action sets using error-correcting output codes for mdp factorization. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 180–194.

[57] 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).

[58] Gabriel Dulac-Arnold, Nir Levine, Daniel J Mankowitz, Jerry Li, Cosmin Paduraru, Sven Gowal, and Todd Hester. 2020. An empirical investigation of the challenges of real-world reinforcement learning. *arXiv preprint arXiv:2003.11881* (2020).

[59] Gabriel Dulac-Arnold, Daniel Mankowitz, and Todd Hester. 2019. Challenges of real-world reinforcement learning. *arXiv preprint arXiv:1904.12901* (2019).

[60] Logan Engstrom, Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Firdaus Janoos, Larry Rudolph, and Aleksander Madry. 2020. Implementation Matters in Deep RL: A Case Study on PPO and TRPO, See [2]. https://openreview.net/forum?id=r1etN1rtPB

[61] Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Vlad Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. 2018. IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures. In *International Conference on Machine Learning*. 1407–1416.

[62] Linxi Fan, Yuke Zhu, Jiren Zhu, Zihua Liu, Orien Zeng, Anchit Gupta, Joan Creus-Costa, Silvio Savarese, and Li Fei-Fei. 2018. SURREAL: Open-Source Reinforcement Learning Framework and Robot Manipulation Benchmark. In *Conference on Robot Learning*.

[63] Mehrdad Farajtabar, Yinlam Chow, and Mohammad Ghavamzadeh. 2018. More Robust Doubly Robust Off-policy Evaluation. In *International Conference on Machine Learning*. 1447–1456.

[64] Jesse Farebrother, Marlos C Machado, and Michael Bowling. 2018. Generalization and regularization in DQN. *arXiv preprint arXiv:1810.00123* (2018).

[65] Jun Feng, Heng Li, Minlie Huang, Shichen Liu, Wenwu Ou, Zhirong Wang, and Xiaoyan Zhu. 2018. Learning to collaborate: Multi-scenario ranking via multi-agent reinforcement learning. In *Proceedings of the 2018 World Wide Web Conference*. International World Wide Web Conferences Steering Committee, 1939–1948.

[66] Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 1126–1135.

[67] John Foley, Emma Tosch, Kaleigh Clary, and David Jensen. 2018. Toybox: Better Atari Environments for Testing Reinforcement Learning Agents. *arXiv preprint arXiv:1812.02850* (2018).

[68] Vincent François-Lavet et al. 2016. DeeR. https://deer.readthedocs.io/.

[69] Vincent François-Lavet, Peter Henderson, Riashat Islam, Marc G Bellemare, Joelle Pineau, et al. 2018. An introduction to deep reinforcement learning. *Foundations and Trends® in Machine Learning* 11, 3-4 (2018), 219–354.

[70] Scott Fujimoto, Herke Hoof, and David Meger. 2018. Addressing Function Approximation Error in Actor-Critic Methods. In *International Conference on Machine Learning*. 1587–1596.

[71] Yasuhiro Fujita, Toshiki Kataoka, Prabhat Nagarajan, and Takahiro Ishikawa. 2019. ChainerRL: A Deep Reinforcement Learning Library. *arXiv preprint arXiv:1912.03905* (2019).

[72] 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 *2019 IEEE International Conference on Data Mining (ICDM)*. IEEE, 1048–1053.

[73] The garage contributors. 2019. Garage: A toolkit for reproducible reinforcement learning research. https://github.com/rlworkgroup/garage.

[74] 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).

[75] Kostadin Georgiev and Preslav Nakov. 2013. A non-iid framework for collaborative filtering with restricted boltzmann machines. In *International conference on machine learning*. 1148–1156.

[76] Carlos A Gomez-Uribe and Neil Hunt. 2015. The netflix recommender system: Algorithms, business value, and innovation. *ACM Transactions on Management Information Systems (TMIS)* 6, 4 (2015), 1–19.

[77] Yu Gong, Yu Zhu, Lu Duan, Qingwen Liu, Ziyu Guan, Fei Sun, Wenwu Ou, and Kenny Q Zhu. 2019. Exact-k recommendation via maximal clique optimization. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 617–626.

[78] 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 *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 535–544.

[79] 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 *International Conference on Machine Learning*. 1856–1865.

[80] Dylan Hadfield-Menell, Smitha Milli, Pieter Abbeel, Stuart J Russell, and Anca Dragan. 2017. Inverse reward design. In *Advances in neural information processing systems*. 6765–6774.

[81] Jianhua Han, Yong Yu, Feng Liu, Ruiming Tang, and Yuzhou Zhang. 2019. Optimizing Ranking Algorithm in Recommender System via Deep Reinforcement Learning. In *2019 International Conference on Artificial Intelligence and Advanced Manufacturing (AIAM)*. IEEE, 22–26.

[82] Negar Hariri, Bamshad Mobasher, and Robin Burke. 2012. Context-aware music recommendation based on latenttopic sequential patterns. In *Proceedings of the sixth ACM conference on Recommender systems*. 131–138.

[83] F Maxwell Harper and Joseph A Konstan. 2015. The movielens datasets: History and context. *Acm transactions on interactive intelligent systems (tiis)* 5, 4 (2015), 1–19.

[84] Hado V Hasselt. 2010. Double Q-learning. In *Advances in neural information processing systems*. 2613–2621.

[85] Matthew Hausknecht and Peter Stone. 2015. Deep recurrent q-learning for partially observable mdps. In *2015 AAAI Fall Symposium Series*.

[86] Milos Hauskrecht. 1997. Incremental methods for computing bounds in partially observable Markov decision processes. In *AAAI/IAAI*. Citeseer, 734–739.

[87] Ji He, Jianshu Chen, Xiaodong He, Jianfeng Gao, Lihong Li, Li Deng, and Mari Ostendorf. 2016. Deep Reinforcement Learning with a Natural Language Action Space. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*. The Association for Computer Linguistics. https://doi.org/10.18653/v1/p16-1153

[88] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web*. 173–182.

[89] P. Henderson, W.-D. Chang, F. Shkurti, J. Hansen, D. Meger, and G. Dudek. 2017. Benchmark Environments for Multitask Learning in Continuous Domains. *ICML Lifelong Learning: A Reinforcement Learning Approach Workshop* (2017).

[90] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. 2018. Deep reinforcement learning that matters. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

[91] Antonio Hernando, Jesús Bobadilla, and Fernando Ortega. 2016. A non negative matrix factorization for collaborative filtering recommender systems based on a Bayesian probabilistic model. *Knowledge-Based Systems* 97 (2016), 188–202.

[92] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. 2018. Rainbow: Combining improvements in deep reinforcement learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

[93] Todd Hester and Peter Stone. 2013. TEXPLORE: real-time sample-efficient reinforcement learning for robots. *Machine learning* 90, 3 (2013), 385–429.

[94] Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Dan Horgan, John Quan, Andrew Sendonaris, Ian Osband, et al. 2018. Deep q-learning from demonstrations. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

[95] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and D Tikk. 2016. Session-based recommendations with recurrent neural networks. In *4th International Conference on Learning Representations, ICLR 2016*.

[96] Jonathan Ho and Stefano Ermon. 2016. Generative adversarial imitation learning. In *Advances in neural information processing systems*. 4565–4573.

[97] Dan Horgan, John Quan, David Budden, Gabriel Barth-Maron, Matteo Hessel, Hado van Hasselt, and David Silver. 2018. Distributed Prioritized Experience Replay. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net. https://openreview.net/forum?id=H1Dy---0Z

[98] Yifan Hu, Yehuda Koren, and Chris Volinsky. 2008. Collaborative filtering for implicit feedback datasets. In *2008 Eighth IEEE International Conference on Data Mining*. Ieee, 263–272.

[99] Tzu-Wei Huang. 2018. tensorboardX. https://github.com/lanpa/tensorboardX/.

[100] Chia-Chun Hung, Timothy Lillicrap, Josh Abramson, Yan Wu, Mehdi Mirza, Federico Carnevale, Arun Ahuja, and Greg Wayne. 2019. Optimizing agent behavior over long time scales by transporting value. *Nature communications* 10, 1 (2019), 1–12.

[101] Eugene Ie, Chih-wei Hsu, Martin Mladenov, Vihan Jain, Sanmit Narvekar, Jing Wang, Rui Wu, and Craig Boutilier. 2019. RecSim: A Configurable Simulation Platform for Recommender Systems. *arXiv preprint arXiv:1909.04847* (2019).

[102] Eugene Ie, Vihan Jain, Jing Wang, Sanmit Narvekar, Ritesh Agarwal, Rui Wu, Heng-Tze Cheng, Tushar Chandra, and Craig Boutilier. 2019. SLATEQ: a tractable decomposition for reinforcement learning with recommendation sets. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*. AAAI Press, 2592–2599.

[103] Eugene Ie, Vihan Jain, Jing Wang, Sanmit Navrekar, Ritesh Agarwal, Rui Wu, Heng-Tze Cheng, Morgane Lustman, Vince Gatto, Paul Covington, et al. 2019. Reinforcement learning for slate-based recommender systems: A tractable decomposition and practical methodology. *arXiv preprint arXiv:1905.12767* (2019).

[104] Maximilian Igl, Luisa Zintgraf, Tuan Anh Le, Frank Wood, and Shimon Whiteson. 2018. Deep Variational Reinforcement Learning for POMDPs. In *International Conference on Machine Learning*. 2117–2126.

[105] Alex Irpan. 2018. Deep Reinforcement Learning Doesn't Work Yet. https://www.alexirpan.com/2018/02/14/rl-hard.html.

[106] Garud N Iyengar. 2005. Robust dynamic programming. *Mathematics of Operations Research* 30, 2 (2005), 257–280.

[107] Max Jaderberg, Wojciech M Czarnecki, Iain Dunning, Luke Marris, Guy Lever, Antonio Garcia Castaneda, Charles Beattie, Neil C Rabinowitz, Ari S Morcos, Avraham Ruderman, et al. 2019. Human-level performance in 3D multiplayer games with population-based reinforcement learning. *Science* 364, 6443 (2019), 859–865.

[108] Nan Jiang and Lihong Li. 2016. Doubly Robust Off-policy Value Evaluation for Reinforcement Learning. In *International Conference on Machine Learning*. 652–661.

[109] Arthur Juliani, Ahmed Khalifa, Vincent-Pierre Berges, Jonathan Harper, Ervin Teng, Hunter Henry, Adam Crespi, Julian Togelius, and Danny Lange. 2019. Obstacle tower: a generalization challenge in vision, control, and planning. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*. AAAI Press, 2684–2691.

[110] Niels Justesen, Ruben Rodriguez Torrado, Philip Bontrager, Ahmed Khalifa, Julian Togelius, and Sebastian Risi. 2018. Illuminating Generalization in Deep Reinforcement Learning through Procedural Level Generation. *NeurIPS 2018 Workshop on Deep Reinforcement Learning* (2018).

[111] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. 1996. Reinforcement learning: A survey. *Journal of artificial intelligence research* 4 (1996), 237–285.

[112] Michael Kearns, Yishay Mansour, and Andrew Y Ng. 2002. A sparse sampling algorithm for near-optimal planning in large Markov decision processes. *Machine*

learning 49, 2-3 (2002), 193–208.
[113] Michał Kempka, Marek Wydmuch, Grzegorz Runc, Jakub Toczek, and Wojciech Jaśkowski. 2016. ViZDoom: A Doom-based AI Research Platform for Visual Reinforcement Learning. In *IEEE Conference on Computational Intelligence and Games*. IEEE, Santorini, Greece, 341–348. http://arxiv.org/abs/1605.02097 The best paper award.
[114] Sergey Kolesnikov and Oleksii Hrinchuk. 2019. Catalyst. RL: A Distributed Framework for Reproducible RL Research. *arXiv preprint arXiv:1903.00027* (2019).
[115] Vijay R Konda and John N Tsitsiklis. 2003. On actor-critic algorithms. *SIAM journal on Control and Optimization* 42, 4 (2003), 1143–1166.
[116] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. *Computer* 42, 8 (2009), 30–37.
[117] Tejas D Kulkarni, Karthik Narasimhan, Ardavan Saeedi, and Josh Tenenbaum. 2016. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *Advances in neural information processing systems*. 3675–3683.
[118] Intel AI Lab. 2019. Reinforcement Learning Coach - Network Design. https://nervanasystems.github.io/coach/design/network.html.
[119] Thomas Laurent and James von Brecht. 2017. A recurrent neural network without chaos, See [1]. https://openreview.net/forum?id=S1dIzvclg
[120] Nevena Lazic, Craig Boutilier, Tyler Lu, Eehern Wong, Binz Roy, MK Ryu, and Greg Imwalle. 2018. Data center cooling using model-predictive control. In *Advances in Neural Information Processing Systems*. 3814–3823.
[121] Hoang Le, Nan Jiang, Alekh Agarwal, Miroslav Dudik, Yisong Yue, and Hal Daumé. 2018. Hierarchical Imitation and Reinforcement Learning. In *International Conference on Machine Learning*. 2917–2926.
[122] Jungkyu Lee, Byonghwa Oh, Jihoon Yang, and Unsang Park. 2017. RLCF: A collaborative filtering approach based on reinforcement learning with sequential ratings. *Intelligent Automation & Soft Computing* 23, 3 (2017), 439–444.
[123] Kimin Lee, Kibok Lee, Jinwoo Shin, and Honglak Lee. 2019. A Simple Randomization Technique for Generalization in Deep Reinforcement Learning. *arXiv preprint arXiv:1910.05396* (2019).
[124] Tao Lei, Yu Zhang, and Yoav Artzi. 2017. Training RNNs as Fast as CNNs. *CoRR* abs/1709.02755 (2017). arXiv:1709.02755 http://arxiv.org/abs/1709.02755
[125] Yu Lei and Wenjie Li. 2019. When Collaborative Filtering Meets Reinforcement Learning. *arXiv preprint arXiv:1902.00715* (2019).
[126] Yu Lei, Zhitao Wang, Wenjie Li, and Hongbin Pei. 2019. Social Attentive Deep Q-network for Recommendation. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 1189–1192.
[127] Justin J Levandoski, Mohamed Sarwat, Ahmed Eldawy, and Mohamed F Mokbel. 2012. Lars: A location-aware recommender system. In *2012 IEEE 28th international conference on data engineering*. IEEE, 450–461.
[128] Lihong Li, Wei Chu, John Langford, and Robert E Schapire. 2010. A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th international conference on World wide web*. ACM, 661–670.
[129] Yuxi Li. 2019. Reinforcement learning applications. *arXiv preprint arXiv:1908.06973* (2019).
[130] Eric Liang, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Ken Goldberg, Joseph Gonzalez, Michael Jordan, and Ion Stoica. 2018. RLlib: Abstractions for Distributed Reinforcement Learning. In *International Conference on Machine Learning*. 3053–3062.
[131] Elad Liebman, Maytal Saar-Tsechansky, and Peter Stone. 2015. DJ-MC: A Reinforcement-Learning Agent for Music Playlist Recommendation. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*. 591–599.
[132] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2016. Continuous control with deep reinforcement learning. See[21]. http://arxiv.org/abs/1509.02971
[133] 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 *Proceedings of the 13th International Conference on Web Search and Data Mining*. 384–392.
[134] 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).
[135] 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. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, 155–167.
[136] Xi Liu, Li Li, Ping-Chun Hsieh, Muhe Xie, Yong Ge, and Rui Chen. 2020. Developing Multi-Task Recommendations with Long-Term Rewards via Policy Distilled Reinforcement Learning. *arXiv preprint arXiv:2001.09595* (2020).
[137] Yong Liu, Yinan Zhang, Qiong Wu, Chunyan Miao, Lizhen Cui, Binqiang Zhao, Yin Zhao, and Lu Guan. 2019. Diversity-promoting deep reinforcement learning for interactive recommendation. *arXiv preprint arXiv:1903.07826* (2019).
[138] Keng Wah Loon, Laura Graesser, and Milan Cvitkovic. 2019. SLM Lab: A Comprehensive Benchmark and Modular Software Framework for Reproducible Deep Reinforcement Learning. *arXiv preprint arXiv:1912.12482* (2019).
[139] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. 2017. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in neural information processing systems*. 6379–6390.
[140] Zhongqi Lu and Qiang Yang. 2016. Partially observable Markov decision process for recommender systems. *arXiv preprint arXiv:1608.07793* (2016).
[141] Hao Ma, Dengyong Zhou, Chao Liu, Michael R Lyu, and Irwin King. 2011. Recommender systems with social regularization. In *Proceedings of the fourth ACM international conference on Web search and data mining*. 287–296.
[142] Jiaqi Ma, Zhe Zhao, Xinyang Yi, Ji Yang, Minmin Chen, Jiaxi Tang, Lichan Hong, and Ed H Chi. 2020. Off-policy Learning in Two-stage Recommender Systems. In *Proceedings of The Web Conference 2020*. 463–473.
[143] Marlos C Machado, Marc G Bellemare, Erik Talvitie, Joel Veness, Matthew Hausknecht, and Michael Bowling. 2018. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research* 61 (2018), 523–562.
[144] Tariq Mahmood and Francesco Ricci. 2007. Learning and adaptivity in interactive recommender systems. In *Proceedings of the ninth international conference on Electronic commerce*. ACM, 75–84.
[145] Tariq Mahmood and Francesco Ricci. 2009. Improving recommender systems with adaptive conversational strategies. In *Proceedings of the 20th ACM conference on Hypertext and hypermedia*. 73–82.
[146] Horia Mania, Aurelia Guy, and Benjamin Recht. 2018. Simple random search provides a competitive approach to reinforcement learning. *arXiv preprint arXiv:1803.07055* (2018).
[147] Daniel J Mankowitz, Timothy A Mann, Pierre-Luc Bacon, Doina Precup, and Shie Mannor. 2018. Learning robust options. In *Thirty-Second AAAI Conference on Artificial Intelligence*.
[148] Timothy Arthur Mann, Sven Gowal, Andras Gyorgy, Huiyi Hu, Ray Jiang, Balaji Lakshminarayanan, and Prav Srinivasan. 2019. Learning from Delayed Outcomes via Proxies with Applications to Recommender Systems. In *International Conference on Machine Learning*. 4324–4332.
[149] Hongzi Mao, Parimarjan Negi, Akshay Narayan, Hanrui Wang, Jiacheng Yang, Haonan Wang, Ryan Marcus, Mehrdad Khani Shirkoohi, Songtao He, Vikram Nathan, et al. 2019. Park: An Open Platform for Learning-Augmented Computer Systems. In *Advances in Neural Information Processing Systems*. 2490–2502.
[150] Rowan McAllister and Carl Edward Rasmussen. 2017. Data-efficient reinforcement learning in continuous state-action Gaussian-POMDPs. In *Advances in Neural Information Processing Systems*. 2040–2049.
[151] Alberto Maria Metelli, Amarildo Likmeta, and Marcello Restelli. 2019. Propagating Uncertainty in Reinforcement Learning via Wasserstein Barycenters. In *Advances in Neural Information Processing Systems*. 4335–4347.
[152] Luke Metz, Julian Ibarz, Navdeep Jaitly, and James Davidson. 2017. Discrete sequential prediction of continuous actions for deep rl. *arXiv preprint arXiv:1705.05035* (2017).
[153] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. In *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.). http://arxiv.org/abs/1301.3781
[154] Martin Mladenov, Ofer Meshi, Jayden Ooi, Dale Schuurmans, and Craig Boutilier. 2019. Advantage amplification in slowly evolving latent-state environments. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*. AAAI Press, 3165–3172.
[155] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*. 1928–1937.
[156] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529.
[157] Raymond J. Mooney and Loriene Roy. 2000. Content-based Book Recommending using Learning for Text Categorization. In *Proceedings of the 5th ACM Conference on Digital Libraries (DL '00)*. ACM, ACM Press, New York, USA, 195–204.
[158] Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I Jordan, et al. 2018. Ray: A distributed framework for emerging {AI} applications. In *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*. 561–577.
[159] Isshu Munemasa, Yuta Tomomatsu, Kunioki Hayashi, and Tomohiro Takagi. 2018. Deep reinforcement learning for recommender systems. In *2018 international conference on information and communications technology (icoiact)*. IEEE, 226–233.
[160] Cataldo Musto, Giovanni Semeraro, Marco De Gemmis, and Pasquale Lops. 2015. Word Embedding Techniques for Content-based Recommender Systems: An

Empirical Evaluation.. In *Recsys posters*.

[161] Ofir Nachum, Mohammad Norouzi, Kelvin Xu, and Dale Schuurmans. 2017. Bridging the gap between value and policy based reinforcement learning. In *Advances in Neural Information Processing Systems*. 2775–2785.

[162] Andrew Y Ng and Michael Jordan. 2000. PEGASUS: a policy search method for large MDPs and POMDPs. In *Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence*. 406–415.

[163] Andrew Y Ng, Stuart J Russell, et al. 2000. Algorithms for inverse reinforcement learning.. In *Icml*, Vol. 1. 663–670.

[164] Alex Nichol, Vicki Pfau, Christopher Hesse, Oleg Klimov, and John Schulman. 2018. Gotta learn fast: A new benchmark for generalization in rl. *arXiv preprint arXiv:1804.03720* (2018).

[165] Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. 2016. Deep exploration via bootstrapped DQN. In *Advances in neural information processing systems*. 4026–4034.

[166] Ian Osband, Yotam Doron, Matteo Hessel, John Aslanides, Eren Sezener, Andre Saraiva, Katrina McKinney, Tor Lattimore, Csaba Szepesvári, Satinder Singh, Benjamin Van Roy, Richard S. Sutton, David Silver, and Hado van Hasselt. 2020. Behaviour Suite for Reinforcement Learning, See [2]. https://openreview.net/forum?id=rygf-kSYwH

[167] Makbule Gulcin Ozsoy. 2016. From word embeddings to item recommendation. *arXiv preprint arXiv:1601.01356* (2016).

[168] Charles Packer, Katelyn Gao, Jernej Kos, Philipp Krähenbühl, Vladlen Koltun, and Dawn Song. 2018. Assessing generalization in deep reinforcement learning. *arXiv preprint arXiv:1810.12282* (2018).

[169] Feiyang Pan, Qingpeng Cai, Pingzhong Tang, Fuzhen Zhuang, and Qing He. 2019. Policy gradients for contextual recommendations. In *The World Wide Web Conference*. 1421–1431.

[170] Jason Pazis and Ron Parr. 2011. Generalized value functions for large action sets. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*. 1185–1192.

[171] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. 2018. Sim-to-real transfer of robotic control with dynamics randomization. In *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE, 1–8.

[172] Matthias Plappert. 2016. keras-rl. https://github.com/keras-rl/keras-rl.

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

[174] Mariya Popova, Olexandr Isayev, and Alexander Tropsha. 2018. Deep reinforcement learning for de novo drug design. *Science advances* 4, 7 (2018), eaap7885.

[175] Pascal Poupart and Craig Boutilier. 2005. VDCBPI: an approximate scalable algorithm for large POMDPs. In *Advances in Neural Information Processing Systems*. 1081–1088.

[176] Doina Precup, Richard S. Sutton, and Satinder P. Singh. 2000. Eligibility Traces for Off-Policy Policy Evaluation. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000), Stanford University, Stanford, CA, USA, June 29 - July 2, 2000*, Pat Langley (Ed.). Morgan Kaufmann, 759–766.

[177] Xueming Qian, He Feng, Guoshuai Zhao, and Tao Mei. 2013. Personalized recommendation combining user interest and social circle. *IEEE transactions on knowledge and data engineering* 26, 7 (2013), 1763–1777.

[178] Tabish Rashid, Mikayel Samvelyan, Christian Schroeder, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. 2018. QMIX: Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning. In *International Conference on Machine Learning*. 4295–4304.

[179] James B Rawlings and Christos T Maravelias. 2019. Bringing new technologies and approaches to the operation and control of chemical process systems. *AIChE Journal* 65, 6 (2019), e16615.

[180] Xinyi Ren, Jianlan Luo, Eugen Solowjow, Juan Aparicio Ojea, Abhishek Gupta, Aviv Tamar, and Pieter Abbeel. 2019. Domain randomization for active pose estimation. In *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 7228–7234.

[181] Winder Research. 2020. A Comparison of Reinforcement Learning Frameworks: Dopamine, RLLib, Keras-RL, Coach, TRFL, Tensorforce, Coach and more. https://winderresearch.com/a-comparison-of-reinforcement-learning-frameworks-dopamine-rllib-keras-rl-coach-trfl-tensorforce-coach-and-more/.

[182] Francesco Ricci, Lior Rokach, and Bracha Shapira. 2015. Recommender systems: introduction and challenges. In *Recommender systems handbook*. Springer, 1–34.

[183] Petar Ristoski, Jessica Rosati, Tommaso Di Noia, Renato De Leone, and Heiko Paulheim. 2019. RDF2Vec: RDF graph embeddings and their applications. *Semantic Web* 10, 4 (2019), 721–752.

[184] 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).

[185] Pornthep Rojanavasu, Phaitoon Srinil, and Ouen Pinngern. 2005. New recommendation system using reinforcement learning. *Special Issue of the Intl. J. Computer, the Internet and Management* 13, SP 3 (2005).

[186] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. 2011. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. 627–635.

[187] Stuart Russell. 1998. Learning agents for uncertain environments. In *Proceedings of the eleventh annual conference on Computational learning theory*. 101–103.

[188] Mandana Saebi, Steven Krieg, Chuxu Zhang, Meng Jiang, and Nitesh Chawla. 2020. Heterogeneous Relational Reasoning in Knowledge Graphs with Reinforcement Learning. *arXiv preprint arXiv:2003.06050* (2020).

[189] Ruslan Salakhutdinov, Andriy Mnih, and Geoffrey Hinton. 2007. Restricted Boltzmann machines for collaborative filtering. In *Proceedings of the 24th international conference on Machine learning*. 791–798.

[190] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. 2001. Item-Based Collaborative Filtering Recommendation Algorithms. In *Proceedings of the 10th International Conference on World Wide Web (WWW '01)*. ACM, ACM Press, New York, USA, 285–295.

[191] Michael Schaarschmidt, Alexander Kuhnle, Ben Ellis, Kai Fricke, Felix Gessert, and Eiko Yoneki. 2018. LIFT: Reinforcement Learning in Computer Systems by Learning From Demonstrations. *CoRR* abs/1808.07903 (2018). arXiv:1808.07903 http://arxiv.org/abs/1808.07903

[192] Michael Schaarschmidt, Sven Mika, Kai Fricke, and Eiko Yoneki. 2019. RLgraph: Modular Computation Graphs for Deep Reinforcement Learning. In *Proceedings of the 2nd Conference on Systems and Machine Learning (SysML)*.

[193] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. 2016. Prioritized Experience Replay, See [21]. http://arxiv.org/abs/1511.05952

[194] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. 2015. Trust region policy optimization. In *International conference on machine learning*. 1889–1897.

[195] John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and Pieter Abbeel. 2016. High-Dimensional Continuous Control Using Generalized Advantage Estimation, See [21]. http://arxiv.org/abs/1506.02438

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

[197] Marwin HS Segler, Mike Preuss, and Mark P Waller. 2018. Planning chemical syntheses with deep neural networks and symbolic AI. *Nature* 555, 7698 (2018), 604–610.

[198] Sergio Guadarrama, Anoop Korattikara, Oscar Ramirez, Pablo Castro, Ethan Holly, Sam Fishman, Ke Wang, Ekaterina Gonina, Neal Wu, Efi Kokiopoulou, Luciano Sbaiz, Jamie Smith, Gábor Bartók, Jesse Berent, Chris Harris, Vincent Vanhoucke, Eugene Brevdo. 2018. TF-Agents: A library for Reinforcement Learning in TensorFlow. https://github.com/tensorflow/agents. https://github.com/tensorflow/agents [Online; accessed 24-Feb-2020].

[199] Wenjie Shang, Yang Yu, Qingyang Li, Zhiwei Qin, Yiping Meng, and Jieping Ye. 2019. Environment Reconstruction with Hidden Confounders for Reinforcement Learning based Recommendation. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 566–576.

[200] Guy Shani, David Heckerman, and Ronen I Brafman. 2005. An MDP-based recommender system. *Journal of Machine Learning Research* 6, Sep (2005), 1265–1295.

[201] Shirli Di-Castro Shashua and Shie Mannor. 2017. Deep robust kalman filter. *arXiv preprint arXiv:1703.02310* (2017).

[202] Pavlo Shevchenko. 2019. *A Comparative Evaluation of Deep Reinforcement Learning Frameworks*. Ph.D. Dissertation. Otto-von-Guericke-University Magdeburg.

[203] 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 *Proceedings of the 13th ACM Conference on Recommender Systems*. 491–495.

[204] Jing-Cheng Shi, Yang Yu, Qing Da, Shi-Yong Chen, and An-Xiang Zeng. 2019. Virtual-taobao: Virtualizing real-world online retail environment for reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 4902–4909.

[205] 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.

[206] Shun-Yao Shih and Heng-Yu Chi. 2018. Automatic, Personalized, and Flexible Playlist Generation using Reinforcement Learning. In *Proceedings of the 19th International Society for Music Information Retrieval Conference, ISMIR 2018, Paris, France, September 23-27, 2018*, Emilia Gómez, Xiao Hu, Eric Humphrey, and Emmanouil Benetos (Eds.). 168–174. http://ismir2018.ircam.fr/doc/pdfs/18_Paper.pdf

[207] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. Mastering the game of Go with deep neural networks and tree search. *nature* 529, 7587 (2016), 484.

[208] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. 2017. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815* (2017).
[209] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. 2018. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science* 362, 6419 (2018), 1140–1144.
[210] David Silver, Leonard Newnham, David Barker, Suzanne Weller, and Jason McFall. 2013. Concurrent reinforcement learning from customer interactions. In *International Conference on Machine Learning*. 924–932.
[211] Adam Stooke and Pieter Abbeel. 2018. Accelerated methods for deep reinforcement learning. *arXiv preprint arXiv:1803.02811* (2018).
[212] Adam Stooke and Pieter Abbeel. 2019. rlpyt: A research code base for deep reinforcement learning in pytorch. *arXiv preprint arXiv:1909.01500* (2019).
[213] 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).
[214] Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z Leibo, Karl Tuyls, et al. 2018. Value-decomposition networks for cooperative multi-agent learning based on team reward. In *Proceedings of the 17th international conference on autonomous agents and multiagent systems*. International Foundation for Autonomous Agents and Multiagent Systems, 2085–2087.
[215] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction.* MIT press.
[216] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. 2000. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*. 1057–1063.
[217] 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.
[218] Adith Swaminathan, Akshay Krishnamurthy, Alekh Agarwal, Miro Dudik, John Langford, Damien Jose, and Imed Zitouni. 2017. Off-policy evaluation for slate recommendation. In *Advances in Neural Information Processing Systems*. 3632–3642.
[219] Nima Taghipour and Ahmad Kardan. 2008. A hybrid web recommender system based on q-learning. In *Proceedings of the 2008 ACM symposium on Applied computing*. 1164–1168.
[220] Nima Taghipour, Ahmad Kardan, and Saeed Shiry Ghidary. 2007. Usage-based web recommendations: a reinforcement learning approach. In *Proceedings of the 2007 ACM conference on Recommender systems*. ACM, 113–120.
[221] Ryuichi Takanobu, Tao Zhuang, Minlie Huang, Jun Feng, Haihong Tang, and Bo Zheng. 2019. Aggregating e-commerce search results from heterogeneous sources via hierarchical reinforcement learning. In *The World Wide Web Conference*. 1771–1781.
[222] Aviv Tamar, Shie Mannor, and Huan Xu. 2014. Scaling up robust MDPs using function approximation. In *International Conference on Machine Learning*. 181–189.
[223] Ming Tan. 1993. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the tenth international conference on machine learning*. 330–337.
[224] Jiaxi Tang and Ke Wang. 2018. Personalized top-n sequential recommendation via convolutional sequence embedding. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*. 565–573.
[225] Liang Tang, Yexi Jiang, Lei Li, and Tao Li. 2014. Ensemble Contextual Bandits for Personalized Recommendation. In *Proceedings of the 8th ACM Conference on Recommender Systems (RecSys '14)*. ACM, New York, NY, USA, 73–80. https://doi.org/10.1145/2645710.2645732
[226] Liang Tang, Yexi Jiang, Lei Li, Chunqiu Zeng, and Tao Li. 2015. Personalized Recommendation via Parameter-Free Contextual Bandits. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '15)*. ACM, New York, NY, USA, 323–332. https://doi.org/10.1145/2766462.2767707
[227] Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, Timothy Lillicrap, and Martin Riedmiller. 2018. *DeepMind Control Suite*. Technical Report. DeepMind. https://arxiv.org/abs/1801.00690
[228] Georgios Theocharous, Philip S Thomas, and Mohammad Ghavamzadeh. 2015. Personalized ad recommendation systems for life-time value optimization with guarantees. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*.
[229] Philip Thomas and Emma Brunskill. 2016. Data-efficient off-policy policy evaluation for reinforcement learning. In *International Conference on Machine Learning*. 2139–2148.
[230] Yuandong Tian, Qucheng Gong, Wenling Shang, Yuxin Wu, and C Lawrence Zitnick. 2017. Elf: An extensive, lightweight and flexible research platform for real-time strategy games. In *Advances in Neural Information Processing Systems*. 2659–2669.
[231] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. 2017. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE, 23–30.
[232] Seiya Tokui, Kenta Oono, Shohei Hido, and Justin Clayton. 2015. Chainer: a next-generation open source framework for deep learning. In *Proceedings of workshop on machine learning systems (LearningSys) in the twenty-ninth annual conference on neural information processing systems (NIPS)*, Vol. 5. 1–6.
[233] Hado Van Hasselt and Marco A Wiering. 2009. Using continuous action spaces to solve discrete problems. In *2009 International Joint Conference on Neural Networks*. IEEE, 1149–1156.
[234] Mel Vecerik, Oleg Sushkov, David Barker, Thomas Rothörl, Todd Hester, and Jon Scholz. 2019. A practical approach to insertion with variable socket position using deep reinforcement learning. In *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 754–760.
[235] Chaoyang Wang, Zhiqiang Guo, Jianjun Li, Peng Pan, and Guohui Li. 2020. A Text-based Deep Reinforcement Learning Framework for Interactive Recommendation. *arXiv preprint arXiv:2004.06651* (2020).
[236] Chengwei Wang, Tengfei Zhou, Chen Chen, Tianlei Hu, and Gang Chen. 2020. Off-Policy Recommendation System Without Exploration. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, 16–27.
[237] Hao Wang, Naiyan Wang, and Dit-Yan Yeung. 2015. Collaborative deep learning for recommender systems. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*. 1235–1244.
[238] Huazheng Wang, Qingyun Wu, and Hongning Wang. 2016. Learning Hidden Features for Contextual Bandits. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management (CIKM '16)*. ACM, New York, NY, USA, 1633–1642. https://doi.org/10.1145/2983323.2983847
[239] Huazheng Wang, Qingyun Wu, and Hongning Wang. 2017. Factorization bandits for interactive recommendation. In *Thirty-First AAAI Conference on Artificial Intelligence*. 2695–2702.
[240] Qing Wang, Jiechao Xiong, Lei Han, Han Liu, Tong Zhang, et al. 2018. Exponentially weighted imitation learning for batched historical data. In *Advances in Neural Information Processing Systems*. 6288–6297.
[241] Xiting Wang, Yiru Chen, Jie Yang, Le Wu, Zhengtao Wu, and Xing Xie. 2018. A reinforcement learning framework for explainable recommendation. In *2018 IEEE International Conference on Data Mining (ICDM)*. IEEE, 587–596.
[242] Yuhui Wang, Hao He, and Xiaoyang Tan. 2019. Robust Reinforcement Learning in POMDPs with Incomplete and Noisy Observations. *arXiv preprint arXiv:1902.05795* (2019).
[243] Ziyu Wang, Victor Bapst, Nicolas Heess, Volodymyr Mnih, Rémi Munos, Koray Kavukcuoglu, and Nando de Freitas. 2017. Sample Efficient Actor-Critic with Experience Replay, See [1]. https://openreview.net/forum?id=HyM25Mqel
[244] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Van Hasselt, Marc Lanctot, and Nando De Freitas. 2016. Dueling network architectures for deep reinforcement learning. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning-Volume 48*. 1995–2003.
[245] Christopher John Cornish Hellaby Watkins. 1989. *Learning from delayed rewards*. Ph.D. Dissertation. King's College, Cambridge.
[246] Hao Wen, Liping Fang, and Ling Guan. 2012. A hybrid approach for personalized recommendation of news on the Web. *Expert Systems with Applications* 39, 5 (2012), 5806–5814.
[247] Shimon Whiteson, Brian Tanner, Matthew E Taylor, and Peter Stone. 2011. Protecting against evaluation overfitting in empirical reinforcement learning. In *2011 IEEE symposium on adaptive dynamic programming and reinforcement learning (ADPRL)*. IEEE, 120–127.
[248] Erik Wijmans, Abhishek Kadian, Ari Morcos, Stefan Lee, Irfan Essa, Devi Parikh, Manolis Savva, and Dhruv Batra. 2019. DD-PPO: Learning Near-Perfect Point-Goal Navigators from 2.5 Billion Frames. *arXiv* (2019), arXiv–1911.
[249] Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning* 8, 3-4 (1992), 229–256.
[250] Sam Witty, Jun Ki Lee, Emma Tosch, Akanksha Atrey, Michael Littman, and David Jensen. 2018. Measuring and characterizing generalization in deep reinforcement learning. *arXiv preprint arXiv:1812.02868* (2018).
[251] Chao-Yuan Wu, Amr Ahmed, Alex Beutel, Alexander J Smola, and How Jing. 2017. Recurrent recommender networks. In *Proceedings of the tenth ACM international conference on web search and data mining*. 495–503.
[252] Yikun Xian, Zuohui Fu, S Muthukrishnan, Gerard De Melo, and Yongfeng Zhang. 2019. Reinforcement knowledge graph reasoning for explainable recommendation. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 285–294.
[253] Jiechao Xiong, Qing Wang, Zhuoran Yang, Peng Sun, Lei Han, Yang Zheng, Haobo Fu, Tong Zhang, Ji Liu, and Han Liu. 2018. Parametrized deep q-networks learning: Reinforcement learning with discrete-continuous hybrid action space. *arXiv preprint arXiv:1810.06394* (2018).

[254] Mao Ye, Peifeng Yin, Wang-Chien Lee, and Dik-Lun Lee. 2011. Exploiting geographical influence for collaborative point-of-interest recommendation. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*. 325–334.

[255] Junming Yin, Dongwon Lee, Linhong Zhu, et al. 2019. Deep Reinforcement Learning for Personalized Search Story Recommendation. *arXiv preprint arXiv:1907.11754* (2019).

[256] Zhang Yuyan, Su Xiayao, and Liu Yong. 2019. A Novel Movie Recommendation System Based on Deep Reinforcement Learning with Prioritized Experience Replay. In *2019 IEEE 19th International Conference on Communication Technology (ICCT)*. IEEE, 1496–1500.

[257] Tom Zahavy, Matan Haroush, Nadav Merlis, Daniel J Mankowitz, and Shie Mannor. 2018. Learn what not to learn: Action elimination with deep reinforcement learning. In *Advances in Neural Information Processing Systems*. 3562–3573.

[258] Chunqiu Zeng, Qing Wang, Shekoofeh Mokhtari, and Tao Li. 2016. Online Context-Aware Recommendation with Time Varying Multi-Armed Bandit. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16)*. ACM, New York, NY, USA, 2025–2034. https://doi.org/10.1145/2939672.2939878

[259] Amy Zhang, Nicolas Ballas, and Joelle Pineau. 2018. A dissection of overfitting and generalization in continuous reinforcement learning. *arXiv preprint arXiv:1806.07937* (2018).

[260] Chiyuan Zhang, Oriol Vinyals, Remi Munos, and Samy Bengio. 2018. A study on overfitting in deep reinforcement learning. *arXiv preprint arXiv:1804.06893* (2018).

[261] Jing Zhang, Bowen Hao, Bo Chen, Cuiping Li, Hong Chen, and Jimeng Sun. 2019. Hierarchical Reinforcement Learning for Course Recommendation in MOOCs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 435–442.

[262] Ruiyi Zhang, Tong Yu, Yilin Shen, Hongxia Jin, and Changyou Chen. 2019. Text-Based Interactive Recommendation via Constraint-Augmented Reinforcement Learning. In *Advances in Neural Information Processing Systems*. 15188–15198.

[263] 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.

[264] Dongyang Zhao, Liang Zhang, Bo Zhang, Lizhou Zheng, Yongjun Bao, and Weipeng Yan. 2019. Deep Hierarchical Reinforcement Learning Based Recommendations via Multi-goals Abstraction. *CoRR* abs/1903.09374 (2019). arXiv:1903.09374 http://arxiv.org/abs/1903.09374

[265] Shenglin Zhao, Tong Zhao, Irwin King, and Michael R Lyu. 2017. Geo-teaser: Geo-temporal sequential embedding rank for point-of-interest recommendation. In *Proceedings of the 26th international conference on world wide web companion*. 153–162.

[266] Xiangyu Zhao, Changsheng Gu, Haoshenglun Zhang, Xiaobing Liu, Xiwang Yang, and Jiliang Tang. 2019. Deep Reinforcement Learning for Online Advertising in Recommender Systems. *arXiv preprint arXiv:1909.03602* (2019).

[267] Xiangyu Zhao, Long Xia, Liang Zhang, Zhuoye Ding, Dawei Yin, and Jiliang Tang. 2018. Deep Reinforcement Learning for Page-wise Recommendations. In *Proceedings of the 12th ACM Conference on Recommender Systems (RecSys '18)*. ACM, New York, NY, USA, 95–103. https://doi.org/10.1145/3240323.3240374

[268] Xiangyu Zhao, Long Xia, Yihong Zhao, Dawei Yin, and Jiliang Tang. 2019. Model-based(or Deep) reinforcement learning for whole-chain recommendations. *arXiv preprint arXiv:1902.03987* (2019).

[269] Xiangyu Zhao, Tong Xu, Qi Liu, and Hao Guo. 2016. Exploring the choice under conflict for social event participation. In *International conference on database systems for advanced applications*. Springer, 396–411.

[270] Xiangyu Zhao, Liang Zhang, Zhuoye Ding, Long Xia, Jiliang Tang, and Dawei Yin. 2018. Recommendations with negative feedback via pairwise deep reinforcement learning. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 1040–1048.

[271] 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).

[272] Xiangyu Zhao, Xudong Zheng, Xiwang Yang, Xiaobing Liu, and Jiliang Tang. 2020. Jointly Learning to Recommend and Advertise. *arXiv preprint arXiv:2003.00097* (2020).

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

[274] Lei Zheng, Vahid Noroozi, and Philip S Yu. 2017. Joint deep modeling of users and items using reviews for recommendation. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*. 425–434.

[275] Lianmin Zheng, Jiacheng Yang, Han Cai, Ming Zhou, Weinan Zhang, Jun Wang, and Yong Yu. 2018. MAgent: A many-agent reinforcement learning platform for artificial collective intelligence. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

[276] Nan Zheng and Qiudan Li. 2011. A recommender system based on tag and time information for social tagging systems. *Expert Systems with Applications* 38, 4 (2011), 4575–4587.

[277] Pengfei Zhu, Xin Li, Pascal Poupart, and Guanghui Miao. 2017. On improving deep reinforcement learning for pomdps. *arXiv preprint arXiv:1704.07978* (2017).

[278] Lixin Zou, Long Xia, Zhuoye Ding, Jiaxing Song, Weidong Liu, and Dawei Yin. 2019. Reinforcement Learning to Optimize Long-term User Engagement in Recommender Systems. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2810–2818.

[279] Lixin Zou, Long Xia, Zhuoye Ding, Dawei Yin, Jiaxing Song, and Weidong Liu. 2019. Reinforcement Learning to Diversify Top-N Recommendation. In *International Conference on Database Systems for Advanced Applications*. Springer, 104–120.

[280] Lixin Zou, Long Xia, Pan Du, Zhuo Zhang, Ting Bai, Weidong Liu, Jian-Yun Nie, and Dawei Yin. 2020. Pseudo Dyna-Q: A Reinforcement Learning Framework for Interactive Recommendation. In *Proceedings of the 13th International Conference on Web Search and Data Mining*. 816–824.