Recent Advances in Reinforcement Learning in Finance

Ben Hambly [*] Renyuan Xu [†] Huining Yang[‡*]

February 3, 2022

Abstract

The rapid changes in the finance industry due to the increasing amount of data have revolutionized the techniques on data processing and data analysis and brought new theoretical and computational challenges. In contrast to classical stochastic control theory and other analytical approaches for solving financial decision-making problems that heavily reply on model assumptions, new developments from reinforcement learning (RL) are able to make full use of the large amount of financial data with fewer model assumptions and to improve decisions in complex financial environments. This survey paper aims to review the recent developments and use of RL approaches in finance. We give an introduction to Markov decision processes, which is the setting for many of the commonly used RL approaches. Various algorithms are then introduced with a focus on value and policy based methods that do not require any model assumptions. Connections are made with neural networks to extend the framework to encompass deep RL algorithms. Our survey concludes by discussing the application of these RL algorithms in a variety of decision-making problems in finance, including optimal execution, portfolio optimization, option pricing and hedging, market making, smart order routing, and robo-advising.

Contents

[*]Mathematical Institute, University of Oxford. **Email:** {hambly, yang}@maths.ox.ac.uk

[†]Epstein Department of Industrial and Systems Engineering, University of Southern California. **Email:** renyuanx@usc.edu

[‡]Supported by the EPSRC Centre for Doctoral Training in Industrially Focused Mathematical Modelling (EP/L015803/1) in collaboration with BP plc.

1 Introduction

The mathematical approach to many financial decision making problems has traditionally been through modelling with stochastic processes and using techniques arising from stochastic control. The choice of models is often dictated by the need to balance tractability with applicability. Simple models lead to tractable and implementable strategies in closed-form or that can be found through traditional numerical methods. However, these models sometimes oversimplify the mechanisms and the behaviour of financial markets which may result in strategies that are sub-optimal in practice and that can potentially result in financial losses. On the other hand, models that try to capture realistic features of financial markets are much more complex and are often mathematically and computationally intractable using the classical tools of stochastic optimal control.

In recent years the availability of large amounts of financial data on transactions, quotes and order flows in electronic order-driven markets has revolutionized data processing and statistical modeling techniques in finance and brought new theoretical and computational challenges. In contrast to the classical stochastic control approach, new ideas coming from reinforcement learning (RL) are being developed to make use of all this information. Reinforcement learning describes methods by which agents acting within some system might learn to make optimal decisions through repeated experience gained by interacting with the system. In the finance industry there have been a number of recent successes in applying RL algorithms in areas such as order execution, market making and portfolio optimization that have attracted a lot of attention. This has led to rapid progress in adapting RL techniques to improve trading decisions in various financial markets when participants have limited information on the market and other competitors.

Although there are already a number of more specialized review papers concerning aspects of reinforcement learning in finance, we aim to review a broad spectrum of activity in this area. This survey is intended to provide a systematic introduction to the theory of RL and an introductory discussion of each of the following financial problems – optimal execution, portfolio optimization, option pricing and hedging, market making, smart order routing, and robo-advising. Moreover, we will also

discuss the advantages of RL methods over classical approaches such as stochastic control, especially for the problems that have been studied extensively in the mathematical finance literature. For other recent surveys with different emphases see [39], [44], [71], [121], [148], and [156]. For a discussion of RL methods with applications to financial problems, including option pricing and portfolio optimization, within the broader framework of machine learning, see [58, Chapter 10].

Our survey will begin by discussing Markov decision processes (MDP), the framework for many reinforcement learning ideas in finance. We will then consider different approaches to learning within this framework with the main focus being on value and policy based methods. In order to implement these approaches we will introduce deep reinforcement methods which incorporate deep learning ideas in this context. For our financial applications we will consider a range of topics and for each we will introduce the basic underlying models before considering the RL approach to tackling them. We will discuss a range of papers in each application area and give an indication of their contributions. We conclude with some thoughts about the direction of development of reinforcement learning in finance.

2 The Basics of Reinforcement Learning

Reinforcement learning is an approach to understanding and automating goal-directed learning and decision-making. It leads naturally to algorithms for such problems and is distinguished from other computational approaches by its emphasis on learning by the individual from direct interaction with their environment, without relying on exemplary supervision or complete models of the environment. RL uses a formal framework defining the interaction between a learning agent and their environment in terms of states, actions, and rewards. This framework is intended to be a simple way of representing essential features of the artificial intelligence problem. These features include a sense of cause and effect, a sense of uncertainty and non-determinism, and the existence of explicit goals.

The history of RL has two main threads that were pursued independently before intertwining in modern RL. One thread concerns learning by trial and error and started in the psychology of animal learning. The other thread concerns the problem of optimal control for an evolving system and its solution using value functions and dynamic programming. Although this thread did not involve learning directly, the Bellman equation developed from this line of literature is viewed as the foundation of many important modern RL algorithms such as Q-learning and Actor-Critic.

Over the last few years, RL, grounded on combining classical theoretical results with deep learning and the functional approximation paradigm, has proved to be a fruitful approach to many artificial intelligence tasks from diverse domains. Breakthrough achievements include reaching human-level performance in such complex games as Go [184], and multi-player StarCraft II [206]. The generality of the reinforcement learning framework allows its application in both discrete and continuous spaces to solve tasks in both real and simulated environments [138].

Classical RL research during the last third of the previous century developed an extensive theoretical core on which modern algorithms are based. Several algorithms, such as Temporal-Difference Learning and Q-learning, were developed and are able to solve small-scale problems when either the states of the environment can be enumerated (and stored in memory) or the optimal policy lies in the space of linear or quadratic functions of the state variable. Although these restrictions are extremely limiting, these foundations of classical RL theory underlie the modern approaches which benefit from increased computational power. Combining this framework with deep learning [83] was popularized by the Deep Q-learning algorithm, introduced in [153], which was able to play any of 57 Atari console games without tweaking the network architecture or algorithm hyperparameters. This novel approach was extensively researched and significantly improved over the following years [204, 88, 65].

Our aim in this section is to introduce the foundations of reinforcement learning. We start with the set-up for MDP in Section 2.1 with both an infinite time horizon and a finite time horizon, as

there are financial applications of both settings in the literature. In Section 2.2 we then introduce the learning procedure when the reward and transition dynamics of the MDPs are unknown to the decision-maker. In particular, we introduce various criteria to measure the performance of learning algorithms in different settings. Then we focus on two major classes of model-free reinforcement learning algorithms, value-based methods in Section 2.3 and policy-based methods in Section 2.4, with an infinite-time horizon. Finally, Section 2.5 contains a general discussion of (model-based and model-free) RL with a finite-time horizon, model-based RL with an infinite-time horizon, and regularization methods for RL.

2.1 Set-up: Markov Decision Processes

We first introduce MDPs with infinite time horizon. Many portfolio optimization problems, such as those arising from pension funds, investigate long-term investment strategies and hence it is natural to formulate them as a decision-making problem over an infinite time horizon. We then introduce MDPs with finite time horizon. Problems such as the optimal execution of the purchase or liquidation of a quantity of asset usually involve trading over a short time horizon and there may be a penalty at the terminal time if the targeted amount is not completely executed by then. Finally we discuss different policy classes that are commonly adopted by the decision-maker.

Infinite Time Horizon and Discounted Reward. Let us start with a discrete time MDP with an infinite time horizon and discounted reward. We have a Markov process taking values in a state space $\mathcal{S}$, where we can influence the evolution by taking an action from a set $\mathcal{A}$. The aim is to optimise the expected discounted return from the system by choosing a policy, that is a sequence of actions through time. We formulate this mathematically by defining the value function $V^\star$ for each $s \in \mathcal{S}$ to be

$$V^\star(s) = \sup_\Pi V^\Pi(s) := \sup_\Pi \mathbb{E}^\Pi\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \middle| s_0 = s\right], \tag{2.1}$$

subject to

$$s_{t+1} \sim P(s_t, a_t), \quad a_t \sim \pi_t(s_t). \tag{2.2}$$

Here and throughout the paper, $\mathbb{E}^\Pi$ denotes the expectation under the policy Π, where the probability measure $\mathbb{P}$ describes both dynamics and the rewards in the MDP. We will write $\mathcal{P}(\mathcal{X})$ for the probability measures over a space $\mathcal{X}$. The state space $(\mathcal{S}, d_\mathcal{S})$ and the action space $(\mathcal{A}, d_\mathcal{A})$ are both complete separable metric spaces, which includes the case of $\mathcal{S}$ and $\mathcal{A}$ being finite, as often seen in the RL literature; $\gamma \in (0,1)$ is a discount factor; $s_t \in \mathcal{S}$ and $a_t \in \mathcal{A}$ are the state and the action at time t; $P : \mathcal{S} \times \mathcal{A} \to \mathcal{P}(\mathcal{S})$ is the transition function of the underlying Markov process; the notation $s_{t+1} \sim P(s_t, a_t)$ denotes that s_{t+1} is sampled from the distribution $P(s_t, a_t) \in \mathcal{P}(\mathcal{S})$; the reward $r(s, a)$ is a random variable in $\mathbb{R}$ for each pair $(s, a) \in \mathcal{S} \times \mathcal{A}$; and the policy $\Pi = \{\pi_t\}_{t=0}^\infty$ is Markovian, in that it just depends on the current state, and can be either deterministic or randomized. For a deterministic policy, $\pi_t : \mathcal{S} \to \mathcal{A}$ maps the current state s_t to a deterministic action. On the other hand, a randomized policy $\pi_t : \mathcal{S} \to \mathcal{P}(\mathcal{A})$ maps the current state s_t to a distribution over the action space. For a randomized policy π_t, we will use $\pi_t(s) \in \mathcal{P}(\mathcal{A})$ and $\pi_t(s, a)$ to represent, respectively, the distribution over the action space given state s and the probability of taking action a at state s. In this case, with infinite-time horizon, we assume the reward r and the transition dynamics P are time homogeneous, which is a standard assumption in the MDP literature [171]. We also note that the set up is essentially the same if we consider the problem of minimizing costs rather than maximizing a reward.

The well-known Dynamic Programming Principle (DPP), that the optimal policy can be obtained by maximizing the reward from one step and then proceeding optimally from the new state, can be

used to derive the following Bellman equation for the value function (2.1);

$$V^\star(s) = \sup_{a\in\mathcal{A}} \Big\{ \mathbb{E}[r(s,a)] + \gamma\, \mathbb{E}_{s'\sim P(s,a)}[V^\star(s')] \Big\}. \tag{2.3}$$

We write the value function as

$$V^\star(s) = \sup_{a\in\mathcal{A}} Q^\star(s,a),$$

where the Q-function, one of the basic quantities used for RL, is defined to be

$$Q^\star(s,a) = \mathbb{E}[r(s,a)] + \gamma\, \mathbb{E}_{s'\sim P(s,a)}[V^\star(s')], \tag{2.4}$$

the expected reward from taking action a at state s and then following the optimal policy thereafter. There is also a Bellman equation for the Q-function given by

$$Q^\star(s,a) = \mathbb{E}[r(s,a)] + \gamma\, \mathbb{E}_{s'\sim P(s,a)} \sup_{a'\in\mathcal{A}} Q^\star(s',a'). \tag{2.5}$$

This allows us to retrieve the optimal (stationary) policy $\pi^*(s,a)$ (if it exists) from $Q(s,a)$, in that $\pi^*(s,a) \in \arg\max_{a\in\mathcal{A}} Q(s,a)$.

An alternative setting over an infinite time horizon is the case of average reward, which is also referred to as an ergodic reward. This ergodic setting is not as relevant to financial applications and hence will not be covered in this review paper. We refer the reader to [1, 216] for a more detailed discussion of RL with infinite-time horizon and average reward.

Finite Time Horizon. When there is a finite time horizon $T < \infty$ we no longer discount future values and have a terminal reward. The MDP problem with finite time horizon can be expressed as

$$V_t^\star(s) = \sup_{\Pi} V_t^{\Pi}(s) := \sup_{\Pi} \mathbb{E}^{\Pi}\left[\sum_{u=t}^{T-1} r_u(s_u,a_u) + r_T(s_T) \middle| s_t = s \right], \ \ \forall s\in\mathcal{S}, \tag{2.6}$$

subject to

$$s_{u+1} \sim P_u(s_u,a_u), \quad a_u \sim \pi_u(s_u), \ \ t \le u \le T-1. \tag{2.7}$$

As in the infinite horizon case, we denote by $s_u \in \mathcal{S}$ and $a_u \in \mathcal{A}$ the state and the action of the agent at time u. However, where this differs from the infinite horizon case, is that we allow time-dependent transition and reward functions. Now $P_u : \mathcal{S}\times\mathcal{A} \to \mathcal{P}(\mathcal{S})$ denotes the transition function and $r_u(s,a)$ a real-valued random variable for each pair $(s,a) \in \mathcal{S}\times\mathcal{A}$ for $t \le u \le T-1$. Similarly $r_T(s)$, the terminal reward, is a real-valued random variable for all $s \in \mathcal{S}$. Finally, the Markovian policy $\Pi = \{\pi_u\}_{t=0}^T$ can be either deterministic or randomized.

The corresponding Bellman equation for the value function (2.6) is defined as

$$V_t^\star(s) = \sup_{a\in\mathcal{A}} \Big\{ \mathbb{E}[r_t(s,a)] + \mathbb{E}_{s'\sim P_t(s,a)}[V_{t+1}^\star(s')] \Big\}, \tag{2.8}$$

with the terminal condition $V_T^\star(s) = \mathbb{E}[r_T(s)]$. We write the value function as

$$V_t^\star(s) = \sup_{a\in\mathcal{A}} Q_t^\star(s,a),$$

where the $Q_t^\star$ function is defined to be

$$Q_t^\star(s,a) = \mathbb{E}[r_t(s,a)] + \mathbb{E}_{s'\sim P_t(s,a)}[V_t^\star(s')]. \tag{2.9}$$

There is also a Bellman equation for the Q-function given by

$$Q_t^\star(s,a) = \mathbb{E}[r_t(s,a)] + \mathbb{E}_{s'\sim P_t(s,a)}\left[\sup_{a'\in\mathcal{A}} Q_{t+1}^\star(s',a')\right], \tag{2.10}$$

with the terminal condition $Q_T^\star(s,a) = \mathbb{E}[r_T(s)]$ for all $a \in \mathcal{A}$.

For an infinite horizon MDP with finite state and action space, and finite reward r, a further useful observation is that such an MDP always has a stationary optimal policy, whenever an optimal policy exists.

Theorem 2.1 (Theorem 6.2.7 in [171]). *Assume $|\mathcal{A}| < \infty$, $|\mathcal{S}| < \infty$ and $|r| < \infty$ with probability one. For any infinite horizon discounted MDP, there always exists a deterministic stationary policy that is optimal.*

Theorem 2.1 implies that there always exists a fixed policy so that taking actions specified by that policy at each time step maximizes the discounted reward. The agent does not need to change policies with time. There is a similar result for the average reward case, see Theorem 8.1.2 in [171]. This insight reduces the question of finding the best sequential decision making strategy to the question of finding the *best stationary policy*. To this end, we write $\pi : \mathcal{S} \to \mathcal{P}(\mathcal{A})$ (without a time index) for a stationary policy throughout the paper.

Linear MDPs and Linear Functional Approximation. Linear MDPs have been extensively studied in the recent literature on theoretical RL in order to establish tractable and efficient algorithms. In a linear MDP, both the transition kernels and the reward function are assumed to be linear with respect to some feature mappings [28, 147].

In the infinite horizon setting, an MDP is said to be linear with a feature map $\phi : \mathcal{S}\times\mathcal{A} \to \mathbb{R}^d$, if there exist d unknown (signed) measures $\mu = (\mu^{(1)}, \cdots, \mu^{(d)})$ over $\mathcal{S}$ and an unknown vector $\theta \in \mathbb{R}^d$ such that for any $(s,a) \in \mathcal{S}\times\mathcal{A}$, we have

$$P(\cdot|s,a) = \langle\, \phi(s,a), \mu(\cdot)\,\rangle, \quad r(s,a) = \langle\, \phi(s,a), \theta\,\rangle. \tag{2.11}$$

Similarly for the finite horizon setting, an MDP is said to be linear with a feature map $\phi : \mathcal{S}\times\mathcal{A} \to \mathbb{R}^d$, if for any $0 \le t \le T$, there exist d unknown (signed) measures $\mu_t = (\mu_t^{(1)}, \cdots, \mu_t^{(d)})$ over $\mathcal{S}$ and an unknown vector $\theta_t \in \mathbb{R}^d$ such that for any $(s,a) \in \mathcal{S}\times\mathcal{A}$, we have

$$P_t(\cdot|s,a) = \langle\, \phi(s,a), \mu_t(\cdot)\,\rangle, \quad r_t(s,a) = \langle\, \phi(s,a), \theta_t\,\rangle, \tag{2.12}$$

Typically features are assumed to be known to the agent and bounded, namely, $\|\phi(s,a)\| \le 1$ for all $(s,a) \in \mathcal{S}\times\mathcal{A}$.

The linear MDP framework is closely related to the literature on RL with *linear functional approximation*, where the value function is assumed to be of the form

$$Q(s,a) = \langle\, \psi(s,a), \omega\,\rangle, \quad V(s) = \langle\, \xi(s), \eta\,\rangle \tag{2.13}$$

for the infinite horizon case, and

$$Q_t(s,a) = \langle\, \psi(s,a), \omega_t\,\rangle, \quad V_t(s) = \langle\, \xi(s), \eta_t\,\rangle, \forall\, 0 \le t \le T \tag{2.14}$$

for the finite horizon case. Here $\psi : \mathcal{S}\times\mathcal{A} \to \mathbb{R}^d$ and $\xi : \mathcal{S} \to \mathbb{R}^d$ are known feature mappings and ω, ω_t η, and η_t are unknown vectors. It has been shown that the linear MDP (i.e., (2.11) or (2.12)) and the linear functional approximation formulation (i.e., (2.13) or (2.14)) are equivalent under mild conditions [112, 232]. In addition to linear functional approximations, we refer the reader to [52] for a general discussion of RL with nonlinear functional approximations and to Section 3 for neural network approximation, one of the most popular nonlinear functional approximations used in practice.

2.2 From MDP to Learning

When the transition dynamics P and the reward function r for the infinite horizon MDP problem are unknown, this MDP becomes a reinforcement learning problem, which is to find an optimal stationary policy π (if it exists) while simultaneously learning the unknown P and r. The learning of P and r can be either explicit or implicit, which leads to model-based and model-free RL, respectively. The analogous ideas hold for the finite horizon case. We introduce some standard RL terminology.

Agent-environment Interface. In the RL setting, the learner or the decision-maker is called the agent. The physical world in which the agent operates and interacts with, comprising everything outside the agent, is called the environment. The agent and environment interact at each of a sequence of discrete time steps, $t = 0, 1, 2, 3, \cdots$, in the following way. At the beginning of each time step t, the agent receives some representation of the environment's state, $s_t \in \mathcal{S}$ and selects an action $a_t \in \mathcal{A}$. At the end of this time step, in part as a consequence of its action, the agent receives a numerical reward r_t (possibly stochastic) and a new state s_{t+1} from the environment. The tuple (s_t, a_t, r_t, s_{t+1}) is called a sample at time t and $h_t := \{(s_u, a_u, r_u, s_{u+1})\}_{u=0}^{t}$ is referred to as the history or experience up to time t. An RL algorithm is a finite sequence of well-defined policies for the agent to interact with the environment.

Exploration vs Exploitation. In order to maximize the accumulated reward over time, the agent learns to select her actions based on her past experiences (exploitation) and/or by trying new choices (exploration). Exploration provides opportunities to improve performance from the current sub-optimal solution to the ultimate globally optimal one, yet it is time consuming and computationally expensive as over-exploration may impair the convergence to the optimal solution. Meanwhile, pure exploitation, i.e., myopically picking the current solution based solely on past experience, though easy to implement, tends to yield sub-optimal global solutions. Therefore, an appropriate trade-off between exploration and exploitation is crucial in the design of RL algorithms in order to improve the learning and the optimization performance.

Simulator. In the procedure described above on how the agent interacts with the environment, the agent does so in an online fashion. Namely, the initial state at time step $t + 1$, s_{t+1}, is the state that the agent moves to after taking action a_t from state s_t. This is a challenging setting since an efficient exploration scheme is needed. For instance, Q-learning with the ε-greedy policy (to be introduced in Section 2.3.3) may take exponentially many episodes to learn the optimal policy [116].

Some results assume access to a simulator [119] (a.k.a., a generative model [12]), which allows the algorithm to query *arbitrary* state-action pairs and return the reward and the next state. This implies the agent can "restart" the system at any time. That is, the initial state at time step $t+1$ does not need to be the state that agent moves to after taking action a_t from state s_t. The "simulator" significantly alleviates the difficulty of exploration, since a naive exploration strategy which queries all state-action pairs uniformly at random already leads to the most efficient algorithm for finding an optimal policy [12].

Randomized Policy vs Deterministic Policy. A randomized policy $\pi_t : \mathcal{S} \to \mathcal{P}(\mathcal{A})$ is also known in the control literature as a relaxed control and in game theory as a mixed strategy. Despite the existence of a *deterministic* optimal policy as stated in Theorem 2.1 for the infinite time horizon case, most of the RL algorithms adopt randomized policies to encourage exploration when RL agents are not certain about the environment.

An Example: Multi-armed bandits. We illustrate these points by discussing multi-armed bandit problems, a special case of RL problems. The multi-armed bandit is a model for a set of slot machines. A simple version is that there are a number of arms, each with a stochastic reward coming from a fixed probability distribution which is initially unknown. We try these arms in some order, which may depend on the sequence of rewards that have been observed so far and attempt to maximize our return. A common objective in this context is to find a policy for choosing the next arm to be tried, under which the sum of the expected rewards comes as close as possible to the ideal reward, i.e., the expected reward that would be obtained if we were to try the "best" arm at all times. Thus the agent interacts with the environment by selecting an action - pulling an arm - and receiving a reward. The policy of the order of pulling on the arms has to balance exploitation, the continued use of an arm that is returning a decent reward, with exploration, sampling arms about which there is less information to find one with a possibly better return.

In this setting for a multi-armed bandit problem we have not involved state dynamics. In this case, an admissible policy is simply a distribution over the action space for a randomized policy and a single arm for a deterministic policy. A simple example of a bandit with a state space is the stochastic contextual bandit which is used extensively in the personalized advertisement recommendation and clinical treatment recommendation literature. In this case the state dynamics (also referred to as the context) are sampled i.i.d. from an unknown distribution. For the personalized advertisement recommendation example, the state at time t can be interpreted as the personal information and purchasing behavior of a targeted client.

The problem was first considered in the seminal work of Robbins [174], which derives policies that asymptotically attain an average reward that converges in the limit to the reward of the best arm. The multi-armed bandit problem was later studied in discounted, Bayesian, Markovian, expected reward, and adversarial setups. See Berry and Fristedt [22] for a review of the classical results on the multi-armed bandit problem.

2.2.1 Performance Evaluation

It is not possible to solve MDPs analytically in general and therefore we will discuss numerical algorithms to determine the optimal policies. In order to assess different algorithms we will need a measure of their performance. Here we will consider several types of performance measure: sample complexity, rate of convergence, regret analysis and asymptotic convergence.

For RL with a finite time horizon, one episode contains a sequence of states, actions and rewards, which starts at time 0 and ends at the terminal time T, and the performance is evaluated in terms of the total number of samples in all episodes.[1] Sometimes we also refer to this setting as episodic RL. For RL with infinite time horizon, the performance is evaluated in terms of the number of steps. In this setting one step contains one (state, action, reward, next state) tuple. It is worth noting that the episodic criterion can also be used to analyze infinite horizon problems. One popular technique is to truncate the trajectory (with infinite steps) at a large time and hence translate the problem into an approximating finite time horizon problem. Examples include REINFORCE [239] and the policy gradient method [142] (see Section 2.4).

Throughout the analysis, we use the notation $\widetilde{O}(\cdot)$ to hide logarithmic factors when describing the order of the scaling in ε, δ, $|\mathcal{A}|$ and $|\mathcal{S}|$ (when these are finite), and other possible model parameters such as the discount rate γ in the infinite horizon case or the dimension d of the features when functional approximation is used. We write $\widetilde{\mathcal{O}}'$ rather than $\widetilde{\mathcal{O}}$ to indicate that, although there may be other parameters, we only include the dependence on ε. We denote by poly$(\cdot)$ a polynomial function of its arguments.

[1] Sample complexity with respect to the number of episodes has also been adopted in the literature [54, 55]. The translation between these two criteria is straightforward.

Sample Complexity. In RL, a sample (s, a, r, s') is defined as a tuple consisting of a state s, an action a, an instantaneous reward received by taking action a at state s, and the next state s' observed afterwards. Sample complexity is defined as the total number of samples required to find an approximately optimal policy. This evaluation criterion can be used for any kind of RL problem.

For episodic RL, an algorithm returns, after the end of the $(k-1)$-th episode with M_{k-1} samples used in this episode, a policy π_k to be applied in the k-th episode. The sample complexity of this algorithm is the minimum number $\sum_{k=1}^{K} M_{k-1}(\varepsilon)$ of samples such that for all $k \geq K$, π_k is ε-optimal with probability at least $1-\delta$, i.e., for $k \geq K$,

$$\mathbb{P}\left(V_0^{\pi_k} \geq V_0^{\star} - \varepsilon\right) \geq 1 - \delta. \tag{2.15}$$

For discounted RL, an algorithm returns, after the end of the $(n-1)$-th step with M_{n-1} samples used in this iteration, a policy π_n to be applied in the n-th step. There are several notions of sample complexity in this setting. The most commonly used ones are defined through the value function and the Q-function with all input variables (i.e., $s \in \mathcal{S}$ for V and $(s,a) \in \mathcal{S} \times \mathcal{A}$ for Q) and they are referred to as the V-sample complexity and Q-sample complexity in the literature. The Q-sample complexity of a given algorithm is defined as the minimum number $\sum_{n=1}^{N} M_{n-1}(\varepsilon)$ of samples such that for all $n \geq N$, $Q^{(n)}$ is ε-close to Q^*, that is

$$\|Q^{(n)} - Q^*\|_\infty := \sup_{s\in\mathcal{S}, a\in\mathcal{A}} (Q^{(n)}(s,a) - Q^*(s,a)) \leq \varepsilon, \tag{2.16}$$

holds either with high probability (that is $\mathbb{P}\left(\|Q^{(n)} - Q^*\|_\infty < \varepsilon\right) \geq 1-\delta$) or in the expectation sense $\mathbb{E}[\|Q^{(n)} - Q^*\|_\infty] < \varepsilon$. Similarly, the V-sample complexity is defined as the minimum number $\sum_{n=1}^{N} M_{n-1}(\varepsilon)$ of samples such that for all $n \geq N$, $V^{(n)}$ is ε-close to V^*, that is

$$\|V^{(n)} - V^*\|_\infty := \sup_{s\in\mathcal{S}} (V^{(n)}(s) - V^*(s)) \leq \varepsilon, \tag{2.17}$$

holds either with high probability (in that $\mathbb{P}\left(\|V^{(n)} - V^*\|_\infty < \varepsilon\right) \geq 1-\delta$) or in the expectation sense $\mathbb{E}[\|V^{(n)} - V^*\|_\infty] < \varepsilon$. Note that (2.16) implies (2.17) since $V^{(n)}(s) = \sup_{a\in\mathcal{A}} Q^{(n)}(s,a)$ and $V^*(s) = \sup_{a\in\mathcal{A}} Q^*(s,a)$. In our analysis we do not distinguish between whether the sample complexity bounds for (2.16) and (2.17) are in the high probability sense or in the expectation sense.

The second type of sample complexity, the *sample complexity of exploration*, is defined as the number of samples such that the non-stationary policy π_n at time n is not ε-optimal for the current state s_n. This measure counts the number of mistakes along the whole trajectory. Mathematically speaking, the sample complexity of exploration for a given algorithm is the minimum number $\sum_{n=1}^{N} M_{n-1}(\varepsilon)$ of samples such that for all $n \geq N$, π_n is ε-optimal when starting from the current state s_n with probability at least $1-\delta$, i.e., for $n \geq N$,

$$\mathbb{P}\left(V^{\pi_n}(s_n) \geq V^{\star}(s_n) - \varepsilon\right) \geq 1 - \delta. \tag{2.18}$$

Note that the condition $\mathbb{P}\left(\|V^{(n)} - V^*\|_\infty < \varepsilon\right) \geq 1-\delta$ associated with (2.17) is stronger and implies the condition in (2.18).

Another weaker criterion, the *mean-square sample complexity*, measures the sample complexity in the average sense with respect to the steady state distribution or the initial distribution μ. In this case, the *mean-square sample complexity* is defined as the minimum number $\sum_{n=1}^{N} M_{n-1}(\varepsilon)$ of samples such that for all $n \geq N$,

$$\|V^{\pi_n} - V^*\|_\mu := \sqrt{\int_{\mathcal{S}} (V^{\pi_n}(s) - V^*(s))^2 \mu(ds)} \leq \varepsilon. \tag{2.19}$$

Since (2.19) is an aggregated guarantee via the V function over a state distribution μ, it is implied by (2.17).

Rate of Convergence. To emphasize the convergence with respect to the number of iterations required, we introduce the notion of *rate of convergence*, which uses the relationship between the number of iterations/steps N and the error term ε, to quantify how fast the learned policy converges to the optimal solution. This is also called the iteration complexity in the RL and optimization literature. Compared to the notion of sample complexity introduced above, the rate of convergence calculates the number of iterations needed to reach a given accuracy while ignoring the number of samples used within each iteration. When only one sample is used in each iteration, the rate of convergence coincides with the sample complexity. In addition when a constant number of samples (i.e., independent from ε) are used within each iteration, the rate of convergence and the sample complexity are of the same order with respect to ε.

In particular, an algorithm is said to converge at a *linear* rate if $N \sim \widetilde{\mathcal{O}}'(\log(1/\varepsilon))$. Similarly, an algorithm is said to converge at a *sublinear* rate (slower than linear) if $N \sim \widetilde{\mathcal{O}}'(1/\varepsilon^p)$ with $p \geq 1$, and is said to converge at a *superlinear* (faster than linear) if $N \sim \widetilde{\mathcal{O}}'(\log(\log(1/\varepsilon))$.

Regret Analysis. The *regret* of a given policy π is defined as the difference between the cumulative reward of the optimal policy and that gathered by π. It quantifies the exploration-exploitation trade-off.

For episodic RL with horizon T in formulation (2.6), the regret of an algorithm after K episodes with $M = T\,K$ steps is

$$R(M) = K \times V_0^* - \sum_{k=1}^{K} \mathbb{E}[V_0^{\pi_k}]. \tag{2.20}$$

There is currently no regret analysis for RL problems with infinite time horizon and discounted reward. The natural definition of regret as given above is less meaningful in this case as the cumulative discounted reward is bounded and hence does not scale with T.

Asymptotic Convergence. In addition to sample complexity and regret analysis discussed above, *asymptotic* convergence is another weaker convergence guarantee, which requires the error term to decay to zero as N goes to infinity (without specifying the order of N). This is often a first step in the theoretical analysis of RL algorithms.

2.2.2 Classification of RL Algorithms

An RL algorithm includes one or more of the following components:

- a representation of a value function that provides a prediction of how good each state or each state/action pair is,
- a direct representation of the policy $\pi(s)$ or $\pi(s, a)$,
- a model of the environment (the estimated transition function and the estimated reward function) in conjunction with a planning algorithm (any computational process that uses a model to create or improve a policy).

The first two components are related to what is called model-free RL. When the latter component is used, the algorithm is referred to as model-based RL.

In the MDP setting, model-based algorithms maintain an approximate MDP model by estimating the transition probabilities and the reward function, and derive a value function from the approximate MDP. The policy is then derived from the value function. Examples include [29, 116, 128] and [197]. Another line of model-based algorithms make structural assumptions on the model, using some prior

knowledge, and utilize the structural information for algorithm design. For example, see [67] for learning linear-quadratic regulators over an infinite time horizon and [100, 19] for the finite time horizon case.

Unlike the model-based method, model-free algorithms *directly* learn a value (or state-value) function or the optimal policy without inferring the model. Model-free algorithms can be further divided into two categories, value-based methods and policy-based methods. Policy-based methods explicitly build a representation of a policy and keep it in memory during learning. Examples include policy gradient methods and trust-region policy optimization methods. As an alternative, value-based methods store only a value function without an explicit policy during the learning process. In this case, the policy is implicit and can be derived directly from the value function (by picking the action with the best value).

In our discussion of methodology, we focus on model-free RL algorithms for MDP with infinite horizon and discounted reward. In particular, we introduce some classical value-based and policy-based methods in Sections 2.3 and 2.4 respectively. For the episodic setting and model-based algorithms, see the discussion in Section 2.5.

2.3 Value-based Methods

In this section, we introduce the methodologies of several classical value-based methods in the setting of an infinite time horizon with discounting, finite state and action spaces ($|\mathcal{A}| < \infty$ and $|\mathcal{S}| < \infty$), and stationary policies. The setting with finite state and action spaces is also referred to as the tabular case in the RL literature. For more general set-ups with an infinite number of actions and states, we refer the reader to Section 2.4.4 for a discussion of linear functional approximations and to Section 3.2 for neural network approximations.

Recall that for reinforcement learning, the distribution of the reward function r and the transition function P are unknown to the agent. Therefore the expectations in the Bellman equations (2.3) and (2.5) cannot be calculated directly. On the other hand, the agents can observe samples (s, a, r, s') by interacting with the system without a model of the system's dynamics P or any prior knowledge of r. The agents can then learn the value function or Q-function using the samples. Following this idea, we next introduce the classical temporal-difference learning algorithm, with Q-learning and State-Action-Reward-State-Action (SARSA) as two special cases.

2.3.1 Temporal-Difference Learning

Given some samples (s, a, r, s') obtained by following a policy π, the agent can update her estimate of the value function V^π (defined in (2.1)) at the $(n+1)$-th iteration by

$$V^{\pi,(n+1)}(s) \leftarrow (1-\beta_n(s,a)) \underbrace{V^{\pi,(n)}(s)}_{\text{current estimate}} + \beta_n(s,a) \underbrace{[r + \gamma V^{\pi,(n)}(s')]}_{\text{new estimate}}, \tag{2.21}$$

with some initialization of the value function $V^{\pi,(0)}$. Here $\beta_n(s,a)$ is the learning rate at iteration $n+1$ which balances the weighting between the current estimate and the new estimate. The quantity β_n can be a constant, can depend on the current state s, or even depend on the observed samples up to iteration $n+1$. Equation (2.21) is sometimes referred to as a TD(0) method. This is a special case of a TD(λ) method (for some $\lambda \in [0,1]$) which is a combination of TD learning (with weight λ) and a Monte Carlo method (with weight $1-\lambda$). Here a Monte Carlo method indicates a simulation-based method to calculate the value function with a longer period of data (in the episode-by-episode sense) rather than a single sample in each update. See Section 2.4.2 for a detailed discussion of the REINFORCE method which is an example of such a Monte Carlo method. Equation (2.21) is also referred to as a one-step TD method as it only uses information from one update step of the system. There are multi-step TD methods; see Chapter 7 and Chapter 12 in [195] for more details.

The TD update in (2.21) can also be written as

$$V^{\pi,(n+1)}(s) \leftarrow V^{\pi,(n)}(s) + \beta_n(s,a) \underbrace{\left[\overbrace{r + \gamma V^{\pi,(n)}(s')}^{\text{TD target}} - V^{\pi,(n)}(s) \right]}_{\text{TD error } \delta_n}. \tag{2.22}$$

The term δ_n, commonly called the TD error, measures the difference between the estimated value at s and the better estimate $r + \gamma V^{\pi,(n)}(s')$, which is often called the TD target in the literature. The TD error and TD target are two important components in analyzing the convergence of the approximate value function and arise in various forms in the reinforcement learning literature.

Algorithm 1 TD(0) Method for estimating V^π

1: **Input**: total number of iterations N; the policy π used to sample observations; rule to set learning rate $\beta_n \in (0,1]$ $(0 \le n \le N-1)$
2: Initialize $V^{\pi,(0)}(s)$ for all $s \in \mathcal{S}$
3: Initialize s
4: **for** $n = 0, \cdots, N-1$ **do**
5: Sample action a according to $\pi(s)$
6: Observe r and s' after taking action a
7: Update $V^{\pi,(n+1)}$ according to (2.21) with (s,a,r,s')
8: $s \leftarrow s'$
9: **end for**

2.3.2 Q-learning Algorithm

A version of TD learning is Q-learning. This is a stochastic approximation to the Bellman equation (2.5) for the Q-function with samples observed by the agent. At iteration n $(1 \le n \le N-1)$, the Q-function is updated using a sample (s,a,r,s') where $s' \sim P(s,a)$,

$$Q^{(n+1)}(s,a) \leftarrow (1-\beta_n(s,a)) \underbrace{Q^{(n)}(s,a)}_{\text{current estimate}} + \beta_n(s,a) \underbrace{\left[r(s,a) + \gamma \max_{a'} Q^{(n)}(s',a') \right]}_{\text{new estimate}}. \tag{2.23}$$

Here $\beta_n(s,a)$ is the learning rate which balances the weight between the current estimate $Q^{(n)}$ from iteration n and the new estimate $r(s,a) + \gamma \max_{a'} Q^{(n)}(s',a')$ calculated using the sample (s,a,r,s').

Algorithm 2 Q-learning with samples from a policy π

1: **Input**: total number of iterations N; the policy π to be evaluated; rule to set learning rate $\beta_n \in (0,1]$ $(0 \le n \le N-1)$
2: Initialize $Q^{(0)}(s,a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}$
3: Initialize s
4: **for** $n = 0, \cdots, N-1$ **do**
5: Sample action a according to $\pi(s)$
6: Observe r and s' after taking action a
7: Update $Q^{(n+1)}$ according to (2.23) with sample (s,a,r,s')
8: $s \leftarrow s'$
9: **end for**

The following theorem guarantees the asymptotic convergence of the Q-learning update (2.23) to the Q-function defined in (2.4).

Theorem 2.2 (Watkins and Dayan (1992) [215]). *Assume $|\mathcal{A}| < \infty$ and $|\mathcal{S}| < \infty$. Let $n^i(s,a)$ be the i-th time to visit (s,a). Let $R < \infty$ be a constant. Given bounded rewards $|r| \leq R$, learning rate $0 \leq \beta_n < 1$ and*

$$\sum_{i=1}^{\infty} \beta_{n^i(s,a)} = \infty, \ \sum_{i=1}^{\infty} (\beta_{n^i(s,a)})^2 < \infty, \ \forall s, a, \tag{2.24}$$

then $Q^{(n)}(s,a) \to Q^\star(s,a)$ as $n \to \infty$, $\forall s, a$ with probability 1.

2.3.3 SARSA

In contrast to the Q-learning algorithm, which takes samples from any policy π as the input where these samples could be collected in advance before performing the Q-learning algorithm, SARSA adopts a policy which is based on the agent's current estimate of the Q-function. The different source of samples is indeed the key difference between on-policy and off-policy learning, which will be discussed after the SARSA algorithm.

Algorithm 3 SARSA: On-policy TD Learning

1: **Input**: total number of iterations N, the learning rate $\beta \in (0,1)$[1], and small parameter $\varepsilon > 0$.
2: Initialize $Q^{(0)}(s,a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}$
3: Initialize $s \in \mathcal{S}$
4: Initialize a: Choose a when in s using a policy derived from $Q^{(0)}$ (e.g, ε-greedy)
5: **for** $n = 0, 1, \cdots, N-1$ **do**
6: Take action a, observe reward r and the next step s'
7: Choose a' when in s' using a policy derived from $Q^{(n)}$ (e.g., ε-greedy)

$$Q^{(n+1)}(s,a) \leftarrow (1-\beta) \underbrace{Q^{(n)}(s,a)}_{\text{current estimate}} + \beta \underbrace{(r + \gamma Q^{(n)}(s',a'))}_{\text{new estimate}} \tag{2.25}$$

8: $s \leftarrow s'$, $a \leftarrow a'$
9: **end for**

The policy derived from $Q^{(n)}$ using ε-greedy (line 4 and line 8 in Algorithm 3) suggests the following choice of action a when in state s:

$$\begin{cases} \text{select from } \arg\max_{a'} Q^{(n)}(s,a') & \text{with probability } 1-\varepsilon, \\ \text{uniformly sample from } \mathcal{A} & \text{with probability } \varepsilon. \end{cases} \tag{2.26}$$

In addition to the ε-greedy policy, other exploration methods such as the softmax operator (resulting in a Boltzmann policy) [10, 94, 210] can also be applied.

[1] The learning rate can depend on the number of iterations, the state-action pair and other parameters in the algorithm. For notational simplicity, we demonstrate the SARSA Algorithm (and the rest of the algorithms mentioned) with a constant learning rate β. In the convergence analysis, we use β_n to denote a learning rate which depends on the number of iterations.

On-policy Learning vs. Off-policy Learning. An off-policy agent learns the value of the optimal policy independently of the agent's actions. An on-policy agent learns the value of the policy being carried out by the agent including the exploration steps. Q-learning is an off-policy agent as the samples (s, a, r, s') used in updating (2.23) may be collected from any policy and may be independent of the agent's current estimate of the Q-function. The convergence of the Q-function relies on the exploration scheme of the policy π (see Theorem 2.2). In contrast to Q-learning, SARSA is an on-policy learning algorithm and uses only its own estimation of the Q-function to select an action and receive a real-time sample in each iteration. The difference is seen in how the new estimate in the update is calculated. In the update step of Q-learning (2.23), we have $\max_{a'} Q^{(n)}(s', a')$ which is the maximum value of the Q-function at a given state s'. On the other hand, the new estimate in the update of SARSA (2.25) takes $Q^{(n)}(s', a')$ where a' is selected based on a policy derived from $Q^{(n)}$ (via the ε-greedy policy (2.26)).

2.3.4 Discussion

In this section, we provide a brief overview of sample complexity results for model-free and value-based RL with infinite-time horizon and discounted reward. Sample complexity results for the model-based counterpart are deferred to Section 2.5.

There has been very little non-asymptotic analysis of TD(0) learning. Existing results have focused on linear function approximations. For example, [53] showed that the mean-squared error converges at a rate of $\widetilde{\mathcal{O}}'(\frac{1}{\varepsilon^{1/\sigma}})$ with decaying learning rate $\beta_n = \frac{1}{(1+n)^\sigma}$ for some $\sigma \in (0, 1)$ where i.i.d. samples are drawn from a stationary distribution; [127] provided an improved $\widetilde{\mathcal{O}}'(\frac{1}{\varepsilon})$ bound for iterate-averaged TD(0) with constant step-size; and [24] reached the same mean-square sample complexity $\widetilde{\mathcal{O}}'(\frac{1}{\varepsilon})$ with a simpler proof and extends the framework to the case where non-i.i.d. but Markov samples are drawn from the online trajectory. A similar analysis was applied by [246] to SARSA and Q-learning algorithms for a continuous state space using linear function approximation (see the end of Section 2.1).

The Q-sample complexity for the Q-learning algorithm of $\widetilde{\mathcal{O}}\left(\frac{|\mathcal{S}||\mathcal{A}|}{(1-\gamma)^5\varepsilon^{5/2}}\text{poly}(\log\delta^{-1})\right)$ was first established in [63] with decaying learning rate $\beta_n = \frac{1}{(1+n)^{4/5}}$ and access to a simulator. This was followed by [20] which showed that the same order of Q-sample complexity $\widetilde{\mathcal{O}}\left(\frac{|\mathcal{S}||\mathcal{A}|}{(1-\gamma)^5\varepsilon^{5/2}}\right)$ could be achieved with a constant learning rate $\beta_n \equiv \frac{(1-\gamma)^4\varepsilon^2}{|\mathcal{A}||\mathcal{S}|}$ $(n \leq N)$. Without access to a simulator, delayed Q-learning [191] was shown to achieve a sample complexity of exploration of order $\widetilde{\mathcal{O}}\left(\frac{|\mathcal{S}||\mathcal{A}|}{(1-\gamma)^8\varepsilon^4}\text{poly}(\log\delta^{-1})\right)$ assuming a known reward. An improvement to this result to $\widetilde{\mathcal{O}}\left(\frac{|\mathcal{S}||\mathcal{A}|}{(1-\gamma)^7\varepsilon^4}\text{poly}(\log\delta^{-1})\right)$ has recently been obtained using an Upper-Confidence-Bound (UCB) exploration policy [213] and a single trajectory. In addition, sample complexities of Q-learning variants with (linear) function approximations have been established in several recent papers. [232] assumed a linear MDP framework (see the end of Section 2.1) and the authors provided a V-sample complexity $\widetilde{\mathcal{O}}\left(\frac{d}{(1-\gamma)^3\varepsilon^2}\text{poly}(\log\delta^{-1})\right)$ with d denoting the dimension of the features. This result matches the information-theoretic lower bound up to $\log(\cdot)$ factors. Later work, [129] considered the setting where the transition kernel can be approximated by linear functions with small approximation errors to make the model more robust to model mis-specification. In this case, the authors provided a V-sample complexity of order $\widetilde{\mathcal{O}}\left(\frac{K}{(1-\gamma)^4\varepsilon^2}\text{poly}(\log\delta^{-1})\right)$. Although the dependence on γ is not as good as in [232], this framework can be applied to a more general class of models which are "near" linear. Although Q-learning is one of the most successful algorithms for finding the best action-value function Q, its implementation often suffers from large overestimation of the Q-function values incurred by random sampling. The double Q-learning algorithm proposed in [101] tackled this overestimation issue by randomly switching the update between two Q-estimators,

and has thus gained significant popularity in practice. The first sample complexity result for the double Q-learning algorithm was established in [223] with a Q-sample complexity on the order of $\widetilde{\mathcal{O}}\left(\left(\frac{1}{(1-\gamma)^6\varepsilon^2}\ln\frac{|\mathcal{A}||\mathcal{S}|}{(1-\gamma)^7\varepsilon^2\delta}\right)^{1/\omega}+\left(\frac{1}{1-\gamma}\ln\frac{1}{(1-\gamma)^2\varepsilon}\right)^{1/(1-\omega)}\right)$ and learning rate $\beta_n=\frac{1}{n^w}$ for some constant $\omega\in(0,1)$.

2.4 Policy-based Methods

So far we have focused on value-based methods where we learn the value function to generate the policy using, for instance, the ε-greedy approach. However, these methods may suffer from high computational complexity as the dimensionality of the state or action space increases (for example in continuous or unbounded spaces). In this case, it may be more effective to directly parametrize the policy rather than the value function.

In this section, we focus on model-free policy-based methods, where we learn a parametrized policy without inferring the value function in the setting of infinite time horizon with discounting, finite state and action spaces, and stationary policies. Examples of policy-based methods include REINFORCE [219], Actor-Critic methods [123], Trust Region Policy Optimization (TRPO) [177], Proximal Policy Optimization (PPO) [178] among others. We first parametrize the policy π by a vector θ, thus we define $\pi(s,\cdot;\theta)\in\mathcal{P}(\mathcal{A})$, the probability distribution parameterized by θ over the action space given the state s at time t. Here we will use $\pi(s,a;\theta)$ and π_θ interchangeably to represent the policy parameterized by θ. We write μ^{π_θ} for the stationary distribution of the state under policy $\pi(s,a;\theta)$. Then the policy objective function is defined to be

$$J(\theta):=\int_{\mathcal{S}}V^{\pi_\theta}(s)\mu^{\pi_\theta}(ds)$$

for RL with infinite time horizon, or

$$J(\theta):=\int_{\mathcal{S}}V_0^{\pi_\theta}(s)\mu^{\pi_\theta}(ds)$$

for RL with finite time horizon. In order to maximize this function, the policy parameter θ is updated using the gradient ascent rule:

$$\theta'=\theta+\beta\,\widehat{\nabla_\theta J(\theta)},\tag{2.27}$$

where β is the learning rate, and $\widehat{\nabla_\theta J(\theta)}$ is an estimate of the gradient of $J(\theta)$ with respect to θ.

2.4.1 Policy Gradient Theorem

Our task now is to estimate the gradient $\nabla_\theta J(\theta)$. The objective function $J(\theta)$ depends on both the policy and the corresponding stationary distribution μ^{π_θ}, and the parameter θ affects both of them. Calculating the gradient of the action with respect to θ is straightforward given the parametrization $\pi(s,a;\theta)$, whereas it might be challenging to analyze the effect of θ on the state when the system is unknown. Fortunately the well-known *policy gradient theorem* provides an expression for $\nabla_\theta J(\theta)$ which does not involve the derivatives of the state distribution with respect to θ. We write the Q-function under policy $\pi(s,a;\theta)$ as $Q^{\pi_\theta}(s,a)$, then we have the following theorem (for more details see [195]).

Theorem 2.3 (Policy Gradient Theorem [196])**.** *Assume that $\pi(s,a;\theta)$ is differentiable with respect to θ and that there exists μ^{π_θ}, the stationary distribution of the dynamics under policy π_θ, which is independent of the initial state s_0. Then the policy gradient is*

$$\nabla_\theta J(\theta)=\mathbb{E}_{s\sim\mu^{\pi_\theta},a\sim\pi_\theta}[\nabla_\theta\ln\pi(s,a;\theta)Q^{\pi_\theta}(s,a)].\tag{2.28}$$

For MDPs with finite state and action space, the stationary distribution exists under mild assumptions. For MDPs with infinite state and action space, more assumptions are needed, for example uniform geometric ergodicity [122].

2.4.2 REINFORCE: Monte Carlo Policy Gradient

Based on the policy gradient expression in (2.28), it is natural to estimate the expectation by averaging over samples of actual rewards, which is essentially the spirit of a standard *Monte Carlo* method. Now we introduce our first policy-based algorithm, called REINFORCE [219] or Monte Carlo policy gradient (see Algorithm 4). We use a simple empirical return function G_t to approximate the Q-function in (2.28). The return G_t is defined as the sum of the discounted rewards and given by $G_t := \sum_{s=t+1}^{T} \gamma^{s-t-1} r_s$. Note that REINFORCE is a Monte Carlo algorithm since it employs the complete sample trajectories, that is, when estimating the return from time t, it uses all future rewards up until the end of the trajectory (see line 4 in Algorithm 4).

As a standard Monte Carlo method, REINFORCE provides an unbiased estimate of the policy gradient, however it suffers from high variance which therefore may lead to slow convergence. A popular variant of REINFORCE is to subtract a *baseline*, which is a function of the state s, from the return G_t. This reduces the variance of the policy gradient estimate while keeping the mean of the estimate unchanged. A commonly used baseline is an estimate of the value function, $\widehat{V}(s)$, which can be learnt using one of the methods introduced in Section 2.3. Replacing the G_t in (2.29) by $G_t - \widehat{V}(s)$ gives a REINFORCE algorithm with baseline.

Algorithm 4 REINFORCE: Monte Carlo Policy Gradient

1: **Input**: total number of iterations N, learning rate β, a differentiable policy parametrization $\pi(s, a; \theta)$, finite length of the trajectory T.
2: Initialize policy parameter $\theta^{(0)}$.
3: **for** $n = 0, 1, \ldots, N-1$ **do**
4: Sample a trajectory $s_0, a_0, r_1, \cdots, s_{T-1}, a_{T-1}, r_T \sim \pi(s, a; \theta^{(n)})$
5: **for** $t = 0, \cdots, T-1$ **do**
6: Calculate the return: $G_t = \sum_{s=t+1}^{T} \gamma^{s-t-1} r_s$
7: Update the policy parameter

$$\theta^{(n+1)} \leftarrow \theta^{(n)} + \beta \gamma^t G_t \nabla_\theta \ln \pi(s_t, a_t; \theta^{(n)}) \tag{2.29}$$

8: **end for**
9: **end for**

2.4.3 Actor-Critic Methods

The fact that REINFORCE provides unbiased policy estimates but may suffer from high variance is an example of the bias-variance dilemma (see, e.g., [73] and [207]) which occurs in many machine learning problems. Also, as a Monte Carlo algorithm, REINFORCE requires complete trajectories so we need to wait until the end of each episode to collect the data. Actor-Critic methods can resolve the above issues by learning both the value function and the policy. The learned value function can improve the policy update through, for example, introducing bias into the estimation. Actor-Critic methods can also learn from incomplete experience in an online manner.

In Actor-Critic methods the value function is parametrized by w and the policy is parametrized by θ. These methods consist of two models: a *critic* which updates the value function or Q-function parameter w, and an *actor* which updates the policy parameter θ based on the information given by

the critic. A natural idea is to use policy-based methods for the actor and use one of the methods introduced in Section 2.3 for the critic; for example SARSA or Q-learning. We give the pseudocode for a simple Actor-Critic method in Algorithm 5. Here for the critic, we approximate the Q-function by a linear combination of features, that is, $Q(s,a;w) = \phi(s,a)^\top w$ for some known feature $\phi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^d$, and use TD(0) to update the parameter w to minimize the TD error with gradient taking the form $\delta\phi(s,a)$ (see lines 9-10 in Algorithm 5). For the actor, we use the (vanilla) policy-based method to update the policy parameter θ (see line 11 in Algorithm 5).

There are three main ways to execute the algorithm. In the *nested-loop* setting (see, e.g. [125, 228]), the actor updates the policy in the outer loop after the critic's repeated updates in the inner loop. The second way is the *two time-scale* setting (see, e.g. [229]), where the actor and the critic update their parameters simultaneously with different learning rates. Note that the learning rate for the actor is typically much smaller than that of the critic in this setting. In the third way, the *single-scale* setting, the actor and the critic update their parameters simultaneously but with a much larger learning rate for the actor than for the critic (see, e.g. [74, 230]).

Algorithm 5 Actor-Critic Algorithm

1: **Input**: A differentiable policy parametrization $\pi(s,a;\theta)$, a differentiable Q-function parameterization $Q(s,a;w)$, learning rates $\beta^\theta > 0$ and $\beta^w > 0$, number of iterations N
2: Initialize policy parameter $\theta^{(0)}$ and Q-function parameter $w^{(0)}$
3: Initialize s
4: **for** $n = 0, 1, \ldots, N-1$ **do**
5: Sample $a \sim \pi(s,a;\theta^{(n)})$
6: Take action a, observe state s' and reward r
7: Sample action $a' \sim \pi(s',\cdot;\theta^{(n)})$
8: Compute the TD error: $\delta \leftarrow r + \gamma Q(s',a';w^{(n)}) - Q(s,a;w^{(n)})$
9: Update w: $w^{(n+1)} \leftarrow w^{(n)} + \beta^w \delta \phi(s,a)$
10: $\theta^{(n+1)} \leftarrow \theta^{(n)} + \beta^\theta Q(s,a;w^{(n)}) \nabla_\theta \ln \pi(s,a;\theta^{(n)})$
11: $s \leftarrow s'$, $a \leftarrow a'$
12: **end for**

2.4.4 Discussion

Variants of Policy-based Methods. There have been many variations of policy-based methods with improvements in various directions. For example, the *natural* policy-based algorithms [113, 168] modify the search direction of the vanilla version by involving a *Fisher information matrix* in the gradient ascent updating rule, which speeds the convergence significantly. To enhance the stability of learning, we may request the updated policy estimate to be not too far away from the previous estimate. Along this line, TRPO [177] uses the *Kullback-Leibler (KL)-divergence* to measure the change between the consecutive updates as a constraint, while PPO [178] incorporate an *adaptive KL-penalty* or a *clipped objective* in the objective function to eliminate the constraint. When using the KL-penalty, PPO can be viewed as a Lagrangian relaxation of the TRPO algorithm. To reduce the sample complexity, Deterministic Policy Gradient (DPG) [185] uses a deterministic function (rather than the stochastic policy π) to represent the policy to avoid sampling over actions (although enough exploration needs to be guaranteed in other ways); Actor-Critic with Experience Replay (ACER) [214] reuses some experience (tuples of data collected in previous iterations/episodes, see the discussion of Deep Q-Networks in Section 3.2) which improves the sample efficiency and decreases the data correlation. In addition we mention Asynchronous Advantage Actor-Critic (A3C) and Advantage Actor-Critic (A2C), two popular Actor-Critic methods with a special focus on parallel training [152]. The latter is a synchronous, deterministic version of the former.

Convergence Guarantees. Now we discuss the convergence guarantees of policy-based algorithms. [196] provided the first asymptotic convergence result for policy gradient methods with arbitrary differentiable function approximation for MDPs with bounded reward. They showed that such algorithms, including REINFORCE and Actor-Critic methods, converge to a locally stationary point. For more examples of asymptotic convergence, see for example [123] and [25].

Based on recent progress in non-convex optimization, non-asymptotic analysis of policy-based methods were first established for convergence to a stationary point. For example, [125] provided a convergence rate analysis for a nested-loop Actor-Critic algorithm to the stationary point through quantifying the smallest number of actor updates k required to attain $\inf_{0\leq m\leq k}\|\nabla J(\theta^{(k)})\|^2<\varepsilon$. We denote this smallest number as K. When the actor uses a policy gradient method, the critic achieves $K\leq\widetilde{\mathcal{O}}'(1/\varepsilon^4)$ by employing TD(0), $K\leq\widetilde{\mathcal{O}}'(1/\varepsilon^3)$ by employing the Gradient Temporal Difference, and $K\leq\widetilde{\mathcal{O}}'(1/\varepsilon^{5/2})$ by employing the Accelerated Gradient Temporal Difference, with continuous state and action spaces. [240] investigated a policy gradient method with Monte Carlo estimation of the policy gradient, called the RPG algorithm, where the rollout length of trajectories is drawn from a Geometric distribution. This generates unbiased estimates of the policy gradient with bounded variance, which was shown to converge to the first order stationary point with diminishing or constant learning rate at a sublinear convergence rate. They also proposed a periodically enlarged learning rate scheme and proved that the modified RPG algorithm with this scheme converges to the second order stationary point in a polynomial number of steps. For more examples of sample complexity analysis for convergence to a stationary point, see for example [162, 225, 226, 183, 222]. The global optimality of stationary points was studied in [23] where they identified certain situations under which the policy gradient objective function has no sub-optimal stationary points despite being non-convex.

Recently there have been results on the sample complexity analysis of the convergence to the global optimum. [41] proved that the entropy regularized natural policy gradient method achieves a Q-sample complexity (and ε-optimal policies) with linear convergence rate. [179] showed that with high probability at least $1-\delta$, the TRPO algorithm has sublinear rate $\widetilde{\mathcal{O}}'(1/\varepsilon^2)$ with mean-squared sample complexity $\widetilde{\mathcal{O}}\big(\frac{|\mathcal{A}|^2(|\mathcal{S}|+\log(1/\delta))}{(1-\gamma)^3\varepsilon^4}\big)$ in the unregularized (tabular) MDPs, and has an improved rate $\widetilde{\mathcal{O}}'(1/\varepsilon)$ with mean-squared sample complexity $\widetilde{\mathcal{O}}\big(\frac{|\mathcal{A}|^2(|\mathcal{S}|+\log(1/\delta))}{(1-\gamma)^4\varepsilon^3}\big)$ in MDPs with entropy regularization. [229] gave the first non-asymptotic convergence guarantee for the two time-scale natural Actor-Critic algorithms with mean-squared sample complexity of order $\widetilde{\mathcal{O}}(\frac{1}{(1-\gamma)^9\varepsilon^4})$. For single-scale Actor-Critic methods, the global convergence with sublinear convergence rate was established in both [74] and [230]. The non-asymptotic convergence of policy-based algorithms is shown in other settings, see [239] for the regret analysis of the REINFORCE algorithm for discounted MDPs and [7, 146] for policy gradient methods in the setting of known model parameters.

2.5 General Discussion

So far we have focused on model-free RL with discounting over an infinite time horizon. In this section, we discuss two other cases: RL over a finite time horizon with both model-based and model-free methods, and model-based RL with an infinite time horizon.

2.5.1 RL with Finite Time Horizon

As discussed earlier, episodic RL with finite time horizon has also been widely used in many financial applications. In this section, we discuss some episodic RL algorithms (with both model-based and model-free methods) and their performance through both sample complexity and regret analysis. [106] proposed a model-based algorithm, known as Posterior Sampling for Reinforcement Learning (PSRL), and establishes an $\widetilde{\mathcal{O}}(T|\mathcal{S}|\sqrt{|\mathcal{A}|M})$ expected regret bound. [54] proposed a so-called Upper-Confidence

Fixed-Horizon (UCFH) RL algorithm that is model-based and has V-sample complexity[2] of order $\widetilde{\mathcal{O}}(\frac{|\mathcal{S}|^2|\mathcal{A}|T^3}{\varepsilon^2})$ assuming known rewards. [14] considered two model-based algorithms in the setting of known reward, called the UCBVI-CH and UCBVI-BF algorithms, which were proved to achieve a regret bound of $\widetilde{\mathcal{O}}(T\sqrt{|\mathcal{S}|\,|\mathcal{A}|M})$ (when $M \geq T|\mathcal{S}|^3|\mathcal{A}|$ and $|\mathcal{S}||\mathcal{A}| \geq T$) and $\widetilde{\mathcal{O}}(\sqrt{T|\mathcal{S}|\,|\mathcal{A}|M})$ (when $M \geq T^3|\mathcal{S}|^3|\mathcal{A}|$ and $|\mathcal{S}||\mathcal{A}| \geq T$), respectively. [55] proposed an Upper Bounding the Expected Next State Value (UBEV) algorithm which achieves a sample complexity[2] of $\widetilde{\mathcal{O}}\big(\frac{|\mathcal{S}|^2|\mathcal{A}|T^5}{\varepsilon^2}\text{polylog}(\frac{1}{\delta})\big)$. They also proved that the algorithm has a regret bound of $\widetilde{\mathcal{O}}(T^2\sqrt{|\mathcal{S}|\,|\mathcal{A}|M})$ with $M \geq |\mathcal{S}|^5|\mathcal{A}|^3$. [111] proved that Q-learning with UCB exploration achieves regret of $\widetilde{\mathcal{O}}(\sqrt{T^3|\mathcal{S}|\,|\mathcal{A}|M})$ without requiring access to a simulator. The above regret bounds depend on the size of the state and action space and thus may suffer from the curse of dimensionality as $|\mathcal{S}|$ and $|\mathcal{A}|$ increase. To tackle this issue, [233] learned a low-dimensional representation of the probability transition model and proved that their MatrixRL algorithm achieves a regret bound of $\widetilde{\mathcal{O}}(T^2\,d\sqrt{M})$ which depends on the number of features d rather than $|\mathcal{S}|$ and $|\mathcal{A}|$. [112] provided a $\widetilde{\mathcal{O}}(\sqrt{d^3T^3M})$ regret bound with high probability for a modified version of the Least-Squares Value Iteration (LSVI) algorithm with UCB exploration.

2.5.2 Model-based RL with Infinite Time Horizon

To derive policies by utilizing estimated transition probabilities and reward functions under the infinite time horizon setting, [29] proposed an R-MAX algorithm which can also be applied in zero-sum stochastic games. The R-MAX algorithm was proved to achieve a sample complexity of exploration of order $\widetilde{\mathcal{O}}\big(\frac{|\mathcal{S}|^2|\mathcal{A}|}{(1-\lambda)^6\varepsilon^3}\big)$ by [135]. The Model-based Interval Estimation (MBIE) in [192] was proved to achieve a sample complexity of exploration of the same order as R-MAX. [197] further modified the R-MAX algorithm to an MoRmax algorithm with an improved sample complexity of exploration of order $\widetilde{\mathcal{O}}\big(\frac{|\mathcal{S}|\,|\mathcal{A}|}{(1-\lambda)^6\varepsilon^2}\big)$. [13] proved that the model-based Q-Value Iteration (QVI) achieves the Q-sample complexity of $\widetilde{\mathcal{O}}(\frac{|\mathcal{S}|\,|\mathcal{A}|}{(1-\gamma)^3\varepsilon^2})$ for some small ε with high probability at least $1-\delta$. [6] studied the approximate MDP model with any accurate black-box planning algorithm and showed that this has the same sample complexity as [13], but with a larger range of ε. See also [191, 116, 128] for model-based RL along this line.

Linear-quadratic (LQ) problems are another type of model-based RL problem that assumes linear state dynamics and quadratic objective functions with continuous state and action space. These problems are one of the few optimal control problems for which there exists a closed-form analytical representation of the optimal feedback control. Thus LQ frameworks, including the (single-agent) *linear-quadratic regulator* (LQR) and (multi-agent) LQ games, can be used in a wide variety of applications, such as the optimal execution problems which we will discuss later. For a theoretical analysis of RL algorithms within the LQ framework, see for example [67] and [100] for the global linear convergence of policy gradient methods in LQR problems with infinite and finite horizon, respectively. See also [19] and [93] for the regret analysis of linear-quadratic reinforcement learning algorithms in continuous time.

2.5.3 Regularization in Reinforcement Learning

Many recent developments in RL benefit from regularization, which has been shown to improve exploration and robustness. Examples include both policy-based methods and value-based methods in various settings. Here we provide a general overview of different regularization techniques and their advantages along with several examples. For example, TRPO and PPO use the KL-divergence between two consecutive policies as a penalty in policy improvement [177, 178]. Soft-Q-learning uses *Shannon*

[2]In the original papers the sample complexity is defined in terms of the number of episodes. Here we multiply the original order in these papers by T to match the definition of sample complexity in this paper.

entropy as a penalty in value iteration [94]. The authors confirmed in simulated experiments that entropy regularization helps to improve exploration and allows transferring skills between tasks. [81] proposed a unified framework to analyze the above algorithms via a regularized Bellman operator. [41] showed that entropy regularization can also help to speed up the convergence rate from a theoretical perspective. It is worth noting that [210] explained the exploitation-exploration trade-off with entropy regularization from a continuous-time stochastic control perspective and provided theoretical support for Gaussian exploration from the special case of linear-quadratic regulator. Recently [77] and [199] extended this idea to a general control framework beyond the linear-quadratic case. [87] extended the idea of Shannon's entropy regularization to mutual-information regularization and showed its superior performance when actions have significantly different importance. When functional approximations are applied in RL training, the regularization technique is shown to be a powerful, flexible, and principled way to balance approximation and estimation errors [145, 66]. The idea of regularization is that one starts from a large function space and controls the solution's complexity by a regularization (or penalization) term. Examples of regularization include the Hilbert norm (for Q-functions) and the l_1 norm (for parameters).

3 Deep Reinforcement Learning

In the setting where the state and action spaces are very large, or high dimensional or they are continuous it is often challenging to apply classical value-based and policy-based methods. In this context, parametrized value functions ($Q(s,a;\theta)$ and $V(s,a;\theta)$) or policies ($\pi(s,a;\theta)$) are more practical. Here we focus on neural-network-based parametrizations and functional approximations, which have the following advantages:

- Neural networks are well suited for dealing with high-dimensional inputs such as time series data, and, in practice, they do not require an exponential increase in data when adding extra dimensions to the state or action space.
- In addition, they can be trained incrementally and make use of additional samples obtained as learning happens.

In this section, we will introduce deep RL methods in the setting of an infinite time horizon with discounting, finite state and action spaces ($|\mathcal{A}| < \infty$ and $|\mathcal{S}| < \infty$), and stationary policies.

3.1 Neural Networks

A typical neural network is a nonlinear function, represented by a collection of neurons. These are typically arranged as a number of layers connected by operators such as filters, poolings and gates, that map an input variable in $\mathbb{R}^n$ to an output variable in $\mathbb{R}^m$ for some $n, m \in \mathbb{Z}^+$.

In this subsection, we introduce three popular neural network architectures including fully-connected neural networks, convolutional neural networks [131] and recurrent neural networks [194].

Fully-connected Neural Networks. A fully-connected Neural Network (FNN) is the simplest neural network architecture where any given neuron is connected to all neurons in the previous layer. To describe the set up we fix the number of layers $L \in \mathbb{Z}^+$ in the neural network and the width of the l-th layer $n_l \in \mathbb{Z}^+$ for $l = 1, 2, \ldots, L$. Then for an input variable $\mathbf{z} \in \mathbb{R}^n$, the functional form of the FNN is

$$F(\mathbf{z}; \boldsymbol{W}, \boldsymbol{b}) = \mathbf{W}_L \cdot \sigma\left(\mathbf{W}_{L-1} \cdots \sigma(\mathbf{W}_1 \mathbf{z} + \mathbf{b}_1) \cdots + \mathbf{b}_{L-1}\right) + \mathbf{b}_L, \tag{3.1}$$

in which $(\mathbf{W}, \mathbf{b})$ represents all the parameters in the neural network, with $\mathbf{W} = (\mathbf{W}_1, \mathbf{W}_2, \ldots, \mathbf{W}_L)$ and $\mathbf{b} = (\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_L)$. Here $\mathbf{W}_l \in \mathbb{R}^{n_l \times n_{l-1}}$ and $\mathbf{b}_l \in \mathbb{R}^{n_l \times 1}$ for $l = 1, 2, \ldots, L$, where $n_0 = n$ is set as the dimension of the input variable. The operator $\sigma(\cdot)$ takes a vector of any dimension as input, and applies a function component-wise. Specifically, for any $q \in \mathbb{Z}^+$ and any vector $\mathbf{u} = (u_1, u_2, \cdots, u_q)^\top \in \mathbb{R}^q$, we have that

$$\sigma(\mathbf{u}) = (\sigma(u_1), \sigma(u_2), \cdots, \sigma(u_q))^\top.$$

In the neural networks literature, the $\mathbf{W}_l$'s are often called the *weight* matrices, the $\mathbf{b}_l$'s are called *bias* vectors, and $\sigma(\cdot)$ is referred to as the *activation function*. Several popular choices for the activation function include ReLU with $\sigma(u) = \max(u, 0)$, Leaky ReLU with $\sigma(u) = a_1 \max(u, 0) - a_2 \max(-u, 0)$ where $a_1, a_2 > 0$, and smooth functions such as $\sigma(\cdot) = \tanh(\cdot)$. In (3.1), information propagates from the input layer to the output layer in a feed-forward manner in the sense that connections between the nodes do not form a cycle. Hence (3.1) is also referred to as fully-connected feed-forward neural network in the deep learning literature.

Convolutional Neural Networks. In addition to the FNNs above, convolutional neural networks (CNNs) are another type of feed-foward neural network that are especially popular for image processing. In the finance setting CNNs have been successfully applied to price prediction based on inputs which are images containing visualizations of price dynamics and trading volumes [109]. The CNNs have two main building blocks – convolutional layers and pooling layers. Convolutional layers are used to capture local patterns in the images and pooling layers are used to reduce the dimension of the problem and improve the computational efficiency.

Each convolutional layer uses a number of trainable *filters* to extract features from the data. We start with the simple case of a single filter $\boldsymbol{H}$, which applies the following convolution operation $\boldsymbol{z} * \boldsymbol{H} : \mathbb{R}^{n_x \times n_y} \times \mathbb{R}^{k_x \times k_y} \to \mathbb{R}^{(n_x - k_x + 1) \times (n_y - k_y + 1)}$ to the input $\boldsymbol{z}$ through

$$[\boldsymbol{z} * \boldsymbol{H}]_{i,j} = \sum_{i'=1}^{k_x} \sum_{j'=1}^{k_y} [\boldsymbol{z}]_{i+i'-1, j+j'-1} [\boldsymbol{H}]_{i',j'}.$$

The output of the convolutional layer $\widehat{\boldsymbol{z}}$ is followed by the activation function $\sigma(\cdot)$, that is

$$[\widehat{\boldsymbol{z}}]_{i,j} = \sigma([\boldsymbol{z} * \boldsymbol{H}]_{i,j}).$$

The weights and bias introduced for FNNs can also be incorporated in this setting thus, in summary, the above simple CNN can be represented by

$$F(\boldsymbol{z}; \boldsymbol{H}, \boldsymbol{W}, \boldsymbol{b}) = \boldsymbol{W} \cdot \sigma(\boldsymbol{z} * \boldsymbol{H}) + \boldsymbol{b}.$$

This simple convolutional layer can be generalized to the case of multiple channels with multiple filters, where the input is a 3D tensor $\boldsymbol{z} \in \mathbb{R}^{n_x \times n_y \times n_z}$ where n_z denotes the number of channels. Each channel represents a feature of the input, for example, an image as an input typically has three channels, red, green, and blue. Then each filter is also represented by a 3D tensor $\boldsymbol{H}_k \in \mathbb{R}^{k_x \times k_y \times n_z}$ $(k = 1, \ldots, K)$ with the same third dimension n_z as the input and K is the total number of filters. In this case, the convolution operation becomes

$$[\boldsymbol{z} * \boldsymbol{H}_k]_{i,j} = \sum_{i'=1}^{k_x} \sum_{j'=1}^{k_y} \sum_{l=1}^{n_z} [\boldsymbol{z}]_{i+i'-1, j+j'-1, l} [\boldsymbol{H}]_{i',j',l},$$

and the output is given by

$$[\widehat{\boldsymbol{z}}]_{i,j,k} = \sigma([\boldsymbol{z} * \boldsymbol{H}_k]_{i,j}).$$

Pooling layers are used to aggregate the information and reduce the computational cost, typically after the convolutional layers. A commonly used pooling layer is the max pooling layer, which computes the maximum value of a small neighbourhood in the spatial coordinates. Note that in addition, fully-connected layers, as introduced above, are often used in the last few layers of a CNN.

Recurrent Neural Networks. Recurrent Neural Networks (RNNs) are a family of neural networks that are widely used in processing sequential data, including speech, text and financial time series data. Unlike feed-forward neural networks, RNNs are a class of artificial neural networks where connections between units form a *directed cycle*. RNNs can use their internal memory to process arbitrary sequences of inputs and hence are applicable to tasks such as sequential data processing.

For RNNs, our input is a sequence of data $\boldsymbol{z}_1, \boldsymbol{z}_2, \ldots, \boldsymbol{z}_T$. An RNN models the *internal* state $\boldsymbol{h}_t$ by a recursive relation

$$\boldsymbol{h}_t = F(\boldsymbol{h}_{t-1}, \boldsymbol{z}_t; \theta),$$

where F is a neural network with parameter θ (for example $\theta = (\boldsymbol{W}, \boldsymbol{b})$). Then the output is given by

$$\widehat{\boldsymbol{z}}_t = G(\boldsymbol{h}_{t-1}, \boldsymbol{z}_t; \phi),$$

where G is another neural network with parameter ϕ (ϕ can be the same as θ). There are two important variants of the vanilla RNN introduced above – the long short-term memory (LSTM) and the gated recurrent units (GRUs). Compared to vanilla RNNs, LSTM and GRUs are shown to have better performance in handling sequential data with long-term dependence due to their flexibility in propagating information flows. Here we introduce the LSTM. Let $\odot$ denote element-wise multiplication. The LSTM network architecture is given by

$$\begin{aligned}
\boldsymbol{f}_t &= \sigma(\boldsymbol{U}^f \boldsymbol{z}_t + \boldsymbol{W}^f \boldsymbol{h}_{t-1} + \boldsymbol{b}^f) \\
\boldsymbol{g}_t &= \sigma(\boldsymbol{U}^g \boldsymbol{z}_t + \boldsymbol{W}^g \boldsymbol{h}_{t-1} + \boldsymbol{b}^g) \\
\boldsymbol{q}_t &= \sigma(\boldsymbol{U}^q \boldsymbol{z}_t + \boldsymbol{W}^q \boldsymbol{h}_{t-1} + \boldsymbol{b}^q) \\
\boldsymbol{c}_t &= \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{g}_t \odot \sigma(\boldsymbol{U}\boldsymbol{z}_t + \boldsymbol{W}\boldsymbol{h}_{t-1} + \boldsymbol{b}) \\
\boldsymbol{h}_t &= \tanh(\boldsymbol{c}_t) \odot \boldsymbol{q}_t
\end{aligned}$$

where $\boldsymbol{U}, \boldsymbol{U}^f, \boldsymbol{U}^g, \boldsymbol{U}^q, \boldsymbol{W}, \boldsymbol{W}^f, \boldsymbol{W}^g, \boldsymbol{W}^q$ are trainable weights matrices, and $\boldsymbol{b}, \boldsymbol{b}^f, \boldsymbol{b}^g, \boldsymbol{b}^q$ are trainable bias vectors. In addition to the internal state $\boldsymbol{h}_t$, the LSTM also uses the *gates* and a *cell* state $\boldsymbol{c}_t$. The forget gate $\boldsymbol{f}_t$ and the input gate $\boldsymbol{g}_t$ determine how much information can be transmitted from the previous cell state $\boldsymbol{c}_{t-1}$ and from the update $\sigma(\boldsymbol{U}\boldsymbol{z}_t + \boldsymbol{W}\boldsymbol{h}_{t-1} + \boldsymbol{b})$, to the current cell state $\boldsymbol{c}_t$. The output gate $\boldsymbol{q}_t$ controls how much $\boldsymbol{c}_t$ reveals to $\boldsymbol{h}_t$. For more details see [194] and [64].

Training of Neural Networks. Mini-batch stochastic gradient descent is a popular choice for training neural networks due to its sample and computational efficiency. In this approach the parameters $\theta = (\boldsymbol{W}, \boldsymbol{b})$ are updated in the descent direction of an objective function $\mathcal{L}(\theta)$ by selecting a mini-batch of samples at random to estimate the gradient of the objective function $\nabla_\theta \mathcal{L}(\theta)$ with respect to the parameter θ. For example, in supervised learning, we aim to learn the relationship between an input X and an output Y, and the objective function $\mathcal{L}(\theta)$ measures the distance between the model prediction (output) and the actual observation Y. Assume that the dataset contains M samples of (X, Y). Then the mini-batch stochastic gradient descent method is given as

$$\theta^{(n+1)} = \theta^{(n)} - \beta \widehat{\nabla_\theta \mathcal{L}(\theta^{(n)})}, \qquad n = 0, \ldots, N-1,$$

where β is a constant learning rate and $\widehat{\nabla_\theta \mathcal{L}(\theta^{(n)})}$ is an estimate of the true gradient $\nabla_\theta \mathcal{L}(\theta^{(n)})$, which is computed by averaging over m ($m \ll M$) samples of (X, Y). It is called (vanilla) stochastic

gradient descent when $m = 1$,and it is the (traditional) gradient descent when $m = M$. Compared with gradient descent, mini-batch stochastic gradient descent is noisy but computationally efficient since it only uses a subset of the data, which is advantageous when dealing with large datasets. In addition to the standard mini-batch stochastic gradient descent methods discussed above, there are many other variants such as RMSprop [201] and ADAM [117], both of which employ adaptive learning rates.

3.2 Deep Value-based RL Algorithms

In this section, we introduce several Q-learning algorithms with neural network approximations. We refer the reader to [72] for other deep value-based RL algorithms such as Neural TD learning and Dueling Q-Networks.

Neural Fitted Q-learning. Fitted Q-learning [86] is a generalization of the classical Q-learning algorithm with functional approximations and it is applied in an off-line setting with a pre-collected dataset in the form of tuples (s, a, r, s') with $s' \sim P(s, a)$ and $r = r(s, a)$ which is random. When the class of approximation functionals is constrained to neural networks, the algorithm is referred to as Neural Fitted Q-learning and the Q-function is parameterized by $Q(s, a; \theta) = F((s, a); \theta)$ with F a neural network function parameterized by θ [173]. For example, F can be set as (3.1) with $\theta = (\boldsymbol{W}, \boldsymbol{b})$.

In Neural Fitted Q-learning, the algorithm starts with some random initialization of the Q-values $Q(s, a; \theta^{(0)})$ where $\theta^{(0)}$ refers to the initial parameters. Then, an approximation of the Q-values at the k-th iteration $Q(s, a; \theta^{(k)})$ is updated towards the target value

$$Y_k^Q = r + \gamma \max_{a' \in \mathcal{A}} Q(s', a'; \theta^{(k)}), \tag{3.2}$$

where $\theta^{(k)}$ are the neural network parameters in the k-th iteration, updated by stochastic gradient descent (or a variant) by minimizing the square loss:

$$\mathcal{L}_{NFQ}(\theta^{(k)}) = \|Q(s, a; \theta^{(k)}) - Y_k^Q\|_2^2. \tag{3.3}$$

Thus, the Q-learning update amounts to updating the parameters:

$$\theta^{(k+1)} = \theta^{(k)} + \beta \left(Y_k^Q - Q(s, a; \theta^{(k)}) \right) \nabla_{\theta^{(k)}} Q(s, a; \theta^{(k)}) \tag{3.4}$$

where β is a learning rate. This update resembles stochastic gradient descent, updating the current value $Q(s, a; \theta^{(k)})$ towards the target value Y_k^Q.

When neural networks are applied to approximate the Q-function, it has been empirically observed that Neural Fitted Q-learning may suffer from slow convergence or even divergence [173]. In addition, the approximated Q-values tend to be overestimated due to the max operator [204].

Deep Q-Network (DQN). To overcome the instability issue and the risk of overestimation mentioned above, [154] proposed a Deep Q-Network (DQN) algorithm in an online setting with two novel ideas. One idea is the slow-update of the target network and the second is the use of 'experience replay'. Both these ideas dramatically improve the empirical performance of the algorithm and DQN has been shown to have a strong performance for a variety of ATARI games [21].

We first discuss the use of experience replay [139]. That is we introduce a replay memory $\mathcal{M}$. At each time t, the tuple (s_t, a_t, r_t, s_{t+1}) is stored in $\mathcal{M}$ and a mini-batch of independent samples is randomly selected from $\mathcal{M}$ to train the neural network via stochastic gradient descent. Since the trajectory of an MDP has strong temporal correlation, the goal of experience replay is to obtain uncorrelated samples, which can give more accurate gradient estimation for the stochastic optimization problem. For experience replay, the replay memory size is usually very large in practice. For example, the replay

memory size is 10^6 in [154]. Moreover, DQN uses the ε-greedy policy, which enables exploration over the state-action space $\mathcal{S} \times \mathcal{A}$. Thus, when the replay memory is large, experience replay is close to sampling independent transitions from an explorative policy. This reduces the variance of the gradient which is used to update θ. Thus, experience replay stabilizes the training of DQN, which benefits the algorithm in terms of computational efficiency.

We now discuss using a target network $Q(\cdot, \cdot; \theta^-)$ with parameter θ^- (the current estimate of the parameter). With independent samples $\{(s_k, a_k, r_k, s'_k)\}_{k=0}^n$ from the replay memory (we use s'_k instead of s_{k+1} for the state after (s_k, a_k) to avoid notational confusion with the next independent sample s_{k+1} in the state space), to update the parameter θ of the Q-network, we compute the target

$$\widetilde{Y}_k^Q = r_k + \gamma \max_{a' \in \mathcal{A}} Q(s'_k, a'_k; \theta^{(k)-}), \tag{3.5}$$

and update θ using the gradient of

$$\mathcal{L}_{DQN}(\theta^{(k)}, \theta^{(k)-}) = \|Q(s, a; \theta^{(k)}) - \widetilde{Y}_k^Q\|_2^2. \tag{3.6}$$

Whereas the parameter $\theta^{(k)-}$ is updated once every T_{target} steps by letting $\theta^{(k)-} = \theta^{(k)}$, if $k = mT_{\text{target}}$ for some $m \in \mathbb{Z}^+$, and $\theta^{(k)-} = \theta^{(k-1)-}$ otherwise. That is, the target network is held fixed for T_{target} steps and then updated usng the current weights of the Q-network. The introduction of a target network prevents the rapid propagation of instabilities and it reduces the risk of divergence as the target values $\widetilde{Y}_k^Q$ are kept fixed for T_{target} iterations. The idea of target networks can be seen as an instantiation of Fitted Q-learning, where each period between target network updates corresponds to a single Fitted Q-iteration.

Double Deep Q-network (DQN). The max operator in Neural Fitted Q-learning and DQN, in (3.2) and (3.5), uses the same values both to select and to evaluate an action. This makes it more likely to select overestimated values, resulting in over optimistic value estimates. To prevent this, Double Q-learning [101] decouples the selection from the evaluation and this is further extended to the neural network setting [204].

In double Q-learning [101] and double deep Q-network [204], two value functions are learned by assigning experiences randomly to update one of the two value functions, resulting in two sets of weights, θ and η. For each update, one set of weights is used to determine the greedy policy and the other to determine its value. For a clear comparison, we can untangle the selection and evaluation in Neural Fitted Q-learning and rewrite its target (3.2) as

$$Y_k^Q = r + \gamma Q(s', \arg\max_{a \in \mathcal{A}} Q(s', a; \theta^{(k)}); \theta^{(k)}). \tag{3.7}$$

The target of Double (deep) Q-learning can then be written as

$$\overline{Y}_k^Q = r + \gamma Q(s', \arg\max_{a \in \mathcal{A}} Q(s', a; \theta^{(k)}); \eta^{(k)}). \tag{3.8}$$

Notice that the selection of the action, in the argmax, is still due to the online weights $\theta^{(k)}$. This means that, as in Q-learning, we are still estimating the value of the greedy policy according to the current values, as defined by $\theta^{(k)}$. However, we use the second set of weights $\eta^{(k)}$ to fairly compute the value of this policy. This second set of weights can be updated symmetrically by switching the roles of θ and η.

Convergence Guarantee. For DQN, [65] characterized the approximation error of the Q-function by the sum of a statistical error and an algorithmic error, and the latter decays to zero at a geometric rate as the algorithm proceeds. The statistical error characterizes the bias and variance arising from the Q-function approximation using the neural network. [32] parametrized the Q-function by a two-layer neural network and provided a mean-squared sample complexity with sublinear convergence rate for neural TD learning. The two-layer network in [32] with width m is given by

$$F\big((s,a);\boldsymbol{W}\big) = \frac{1}{\sqrt{m}}\sum_{l=1}^{m} c_l \cdot \sigma\big(\boldsymbol{W}_l^\top (s,a)\big), \tag{3.9}$$

where the activation function $\sigma(\cdot)$ is set to be the ReLU function, and the parameter $\boldsymbol{c} = (c_1, \ldots, c_m)$ is fixed at the initial parameter during the training process and only the weights $\boldsymbol{W}$ are updated. [227] considered a more challenging setting than [32], where the input data are non-i.i.d and the neural network has multiple (more than two) layers and obtained the same sublinear convergence rate. Furthermore, [40] also employed the two-layer neural network in (3.9) and proved that two algorithms used in practice, the projection-free and max-norm regularized [85] neural TD, achieve mean-squared sample complexity of $\widetilde{\mathcal{O}}'(1/\varepsilon^6)$ and $\widetilde{\mathcal{O}}'(1/\varepsilon^4)$, respectively.

3.3 Deep Policy-based RL Algorithms

In this section we focus on deep policy-based methods, which are extensions of policy-based methods using neural network approximations. We parameterize the policy π by a neural network F with parameter $\theta = (\boldsymbol{W}, \boldsymbol{b})$, that is, $a \sim \pi(s,a;\theta) = f(F((s,a);\theta))$ for some function f. A popular choice of f is given by

$$f(F((s,a);\theta)) = \frac{\exp(\tau F((s,a);\theta))}{\sum_{a'\in\mathcal{A}} \exp(\tau F((s,a');\theta))},$$

for some parameter τ, which gives an *energy-based* policy (see, e.g. [94, 212]). The policy parameter θ is updated using the gradient ascent rule given by

$$\theta^{(n+1)} = \theta^{(n)} + \beta \widehat{\nabla_\theta J(\theta^{(n)})}, \qquad n = 0, \ldots, N-1,$$

where $\widehat{\nabla_\theta J(\theta^{(n)})}$ is an estimate of the policy gradient.

Using neural networks to parametrize the policy and/or value functions in the vanilla version of policy-based methods discussed in Section 2.4 leads to neural Actor-Critic algorithms [212], neural PPO/TRPO [141], and deep DPG (DDPG) [138]. In addition, since introducing an entropy term in the objective function encourages policy exploration [94] and speeds the learning process [95, 146] (as discussed in Section 2.5.3), there have been some recent developments in (off-policy) *soft* Actor-Critic algorithms [95, 96] using neural networks, which solve the RL problem with entropy regularization. Below we introduce the DDPG algorithm, which is one of the most popular deep policy-based methods, and which has been applied in many financial problems.

DDPG. DDPG is a model-free off-policy Actor-Critic algorithm, first introduced in [138], which combines the DQN and DPG algorithms. Since its structure is more complex than DQN and DPG, we provide the pseudocode for DDPG in Algorithm 6. DDPG extends the DQN to continuous action spaces by incorporating DPG to learn a deterministic strategy. To encourage exploration, DDPG uses the following action

$$a_t \sim \pi^D(s_t;\theta_t) + \epsilon,$$

where π^D is a deterministic policy and ϵ is a random variable sampled from some distribution $\mathcal{N}$, which can be chosen according to the environment. Note that the algorithm requires a small learning rate $\bar{\beta} \ll 1$ (see line 14 in Algorithm 6) to improve the stability of learning the target networks.

Algorithm 6 The DDPG Algorithm

Input: an actor $\pi^D(s;\theta)$, a critic network $Q(s,a;\phi)$, learning rates β and $\bar{\beta}$, initial parameters $\theta^{(0)}$ and $\phi^{(0)}$
Initialize target networks parameters $\bar{\phi}^{(0)} \leftarrow \phi^{(0)}$ and $\bar{\theta}^{(0)} \leftarrow \theta^{(0)}$,
Initialize replay buffer $\mathcal{M}$
for $n = 0, \ldots, N-1$ **do**
  Initialize state s_1
  for $t = 0, \ldots, T-1$ **do**
    Select action $a_t \sim \pi^D(s_t;\theta^{(n)}) + \epsilon_t$ with $\epsilon_t \sim \mathcal{N}$
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in $\mathcal{M}$
    Sample a random mini-batch of M transitions $\{(s_i, a_i, r_i, s_{i+1})\}_{i=1}^M$ from $\mathcal{M}$
    Set the target $y_i = r_i + \gamma Q(s_{i+1}, \pi^D(s_{i+1};\bar{\theta}^{(n)});\bar{\phi}^{(n)})$.
    Update the critic by minimizing the loss: $\phi^{(n+1)} = \phi^{(n)} - \beta\nabla_\phi \mathcal{L}_{\text{DDPG}}(\phi^{(n)})$ with $\mathcal{L}_{\text{DDPG}}(\phi^{(n)}) = \frac{1}{M}\sum_{i=1}^M (y_i - Q(s_i, a_i;\phi^{(n)}))^2$.
    Update the actor by using the sampled policy gradient: $\theta^{(n+1)} = \theta^{(n)} + \beta\nabla_\theta J(\theta^{(n)})$ with

$$\nabla_\theta J(\theta^{(n)}) \approx \frac{1}{M}\sum_{i=1}^M \nabla_a Q(s,a;\phi^{(n)})|_{s=s_i,a=a_i} \nabla_\theta \pi^D(s;\theta^{(n)})|_{s=s_i}.$$

    Update the target networks:

$$\begin{aligned} \bar{\phi}^{(n+1)} &\leftarrow \bar{\beta}\phi^{(n+1)} + (1-\bar{\beta})\bar{\phi}^{(n)} \\ \bar{\theta}^{(n+1)} &\leftarrow \bar{\beta}\theta^{(n+1)} + (1-\bar{\beta})\bar{\theta}^{(n)} \end{aligned}$$

  end for
end for

Convergence Guarantee. By parameterizing the policy and/or value functions using a two-layer neural network given in (3.9), [141] provided a mean-squared sample complexity for neural PPO and TRPO algorithms with sublinear convergence rate; [212] studied neural Actor-Critic methods where the actor updates using (1) vanilla policy gradient or (2) natural policy gradient, and in both cases the critic updates using TD(0). They proved that in case (1) the algorithm converges to a stationary point at a sublinear rate and they also established the global optimality of all stationary points under mild regularity conditions. In case (2) the algorithm was proved to achieve a mean-squared sample complexity with sublinear convergence rate. To the best of our knowledge, no convergence guarantee has been established for the DDPG (and DPG) algorithms.

4 Applications in Finance

The availability of data from electronic markets and the recent developments in RL together have led to a rapidly growing recent body of work applying RL algorithms to decision making problems in electronic markets. Examples include optimal execution, portfolio optimization, option pricing and hedging, market making, order routing, as well as robot advising.

In this section, we start with a brief overview of electronic markets and some discussion of market microstructure in Section 4.1. We then introduce several applications of RL in finance. In particular, optimal execution for a single asset is introduced in Section 4.2 and portfolio optimization problems

across multiple assets is discussed in Section 4.3. This is followed by sections on option pricing, robo-advising, and smart order routing. In each we introduce the underlying problem and basic model before looking at recent RL approaches used in tackling them.

4.1 Preliminary: Electronic Markets and Market Microstructure

Many recent decision making problems in finance are centered around electronic markets. We give a brief overview of this type of market and discuss two popular examples – central limit order books and electronic over-the-counter markets. For more in-depth discussion of electronic markets and market microstructure, we refer the reader to the books [38] and [132].

Electronic Markets. Electronic markets have emerged as popular venues for the trading of a wide variety of financial assets. Stock exchanges in many countries, including Canada, Germany, Israel, and the United Kingdom, have adopted electronic platforms to trade equities, as has Euronext, the market combining several former European stock exchanges. In the United States, electronic communications networks (ECNs) such as Island, Instinet, and Archipelago (now ArcaEx) use an electronic order book structure to trade as much as 45% of the volume on the NASDAQ stock market. Several electronic systems trade corporate bonds and government bonds (e.g., BrokerTec, MTS), while in foreign exchange, electronic systems such as EBS and Reuters dominate the trading of currencies. Eurex, the electronic Swiss-German exchange, is now the world's largest futures market, while options have been traded in electronic markets since the opening of the International Securities Exchange in 2000. Many such electronic markets are organized as electronic Limit Order Books (LOBs). In this structure, there is no designated liquidity provider such as a specialist or a dealer. Instead, liquidity arises endogenously from the submitted orders of traders. Traders who submit orders to buy or sell the asset at a particular price are said to "make" liquidity, while traders who choose to hit existing orders are said to "take" liquidity. Another major type of electronic market without a central LOB is the Over-the-Counter (OTC) market where the orders are executed directly between two parties, the dealer and the client, without the supervision of an exchange. Examples include BrokerTec and MTS for trading corporate bonds and government bonds.

Limit Order Books. An LOB is a list of orders that a trading venue, for example the NASDAQ exchange, uses to record the interest of buyers and sellers in a particular financial asset or instrument. There are two types of orders the buyers (sellers) can submit: a limit buy (sell) order with a preferred price for a given volume of the asset or a market buy (sell) order with a given volume which will be immediately executed with the best available limit sell (buy) orders. Therefore limit orders have a price guarantee but are not guaranteed to be executed, whereas market orders are executed immediately at the best available price. The lowest price of all sell limit orders is called the *ask* price, and the highest price of all buy limit orders is called the *bid* price. The difference between the ask and bid is known as the *spread* and the average of the ask and bid is known as the *mid-price*. A snapshot of an LOB[1] with 10 levels of bid/ask prices is shown in Figure 1.

[1]We use NASDAQ ITCH data taken from Lobster https://lobsterdata.com/.

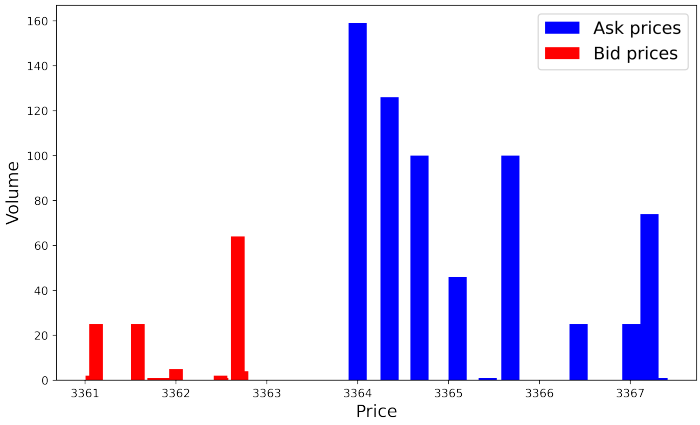

Figure 1: A snapshot of the LOB of AMZN(Amazon) stock at 10:12:52.509 am on 16 April, 2021 with ten levels of bid/ask prices.

A matching engine is used to match the incoming buy and sell orders. This typically follows the price-time priority rule [170], whereby orders are first ranked according to their price. Multiple orders having the same price are then ranked according to the time they were entered. If the price and time are the same for the incoming orders, then the larger order gets executed first. The matching engine uses the LOB to store pending orders that could not be executed upon arrival.

Over-the-counter Markets. OTC or off-exchange trading is done directly between two parties, without the supervision of an exchange. Many OTC markets organized around dealers, including corporate bond markets in many countries, have also undergone upheaval linked to electronification over the last ten years. The electronification process is dominated by Multi-dealer-to-client (MD2C) platforms enabling clients to send the same request for a quote (RFQ) to several dealers simultaneously and therefore instantly put the dealers into competition with one another. These dealers can then each provide a price to the client for the transaction (not necessarily the price the dealer has streamed). Dealers know the identity of the client (which differs from most of the systems organized around a central LOB) and the number of requested dealer prices. However, they do not see the prices that are streamed by the other dealers. They only see a composite price at the bid and offer, based on some of the best streamed prices. The client progressively receives the answers to the RFQ and can deal at any time with the dealer who has proposed the best price, or decide not to trade. Each dealer knows whether a deal was done (with her/him, but also with another dealer - without knowing the identity of this dealer) or not. If a transaction occurred, the best dealer usually knows the cover price (the second best bid price in the RFQ), if there is one. We refer the reader to [69] for a more in-depth discussion of MD2C bond trading platforms.

Market Participants. When considering different market participants it is sometimes helpful to classify them based on their objectives and trading strategies. Three primary classes are the following [38]:

- Fundamental (or noise or liquidity) traders: those who are driven by economic fundamentals outside the exchange.
- Informed traders: traders who profit from leveraging information not reflected in market prices by trading assets in anticipation of their appreciation or depreciation.
- Market makers: professional traders who profit from facilitating exchange in a particular asset and exploit their skills in executing trades.

The effects of the interactions amongst all these traders is one of the key issues studied in the field of market microstructure. How to improve trading decisions from the perspective of one class of market participants, while strategically interacting with the others, is one of the big challenges in the field. The recent literature has seen many attempts to exploit RL techniques to tackle these problems.

4.2 Optimal Execution

Optimal execution is a fundamental problem in financial modeling. The simplest version is the case of a trader who wishes to buy or sell a given amount of a single asset within a given time period. The trader seeks strategies that maximize their return from, or alternatively, minimize the cost of, the execution of the transaction.

The Almgren–Chriss Model. A classical framework for optimal execution is due to Almgren–Chriss [8]. In this setup a trader is required to sell an amount q_0 of an asset, with price S_0 at time 0, over the time period $[0, T]$ with trading decisions made at discrete time points $t = 1, \ldots, T$. The final inventory q_T is required to be zero. Therefore the goal is to determine the liquidation strategy $u_1, u_2, \ldots, u_T$, where u_t $(t = 1, 2, \ldots, T)$ denotes the amount of the asset to sell at time t. It is assumed that selling quantities of the asset will have two types of price impact – a temporary impact which refers to any temporary price movement due to the supply-demand imbalance caused by the selling, and a permanent impact, which is a long-term effect on the 'equilibrium' price due to the trading, that will remain at least for the trading period.

We write S_t for the asset price at time t. The Almgren–Chriss model assumes that the asset price evolves according to the discrete arithmetic random walk

$$S_t = S_{t-1} + \sigma\xi_t - g(u_t), \qquad t = 1, 2, \ldots, T$$

where σ is the (constant) volatility parameter, ξ_t are independent random variables drawn from a distribution with zero mean and unit variance, and g is a function of the trading strategy u_t that measures the permanent impact. The inventory process $\{q_t\}_{t=0}^{T}$ records the current holding in the asset, q_t, at each time t. Thus we have

$$q_t = q_{t-1} - u_t.$$

Selling the asset may cause a temporary price impact, that is a drop in the average price per share, thus the actual price per share received is

$$\widetilde{S}_t = S_{t-1} - h(u_t),$$

where h is a function which quantifies this temporary price impact. The cost of this trading trajectory is defined to be the difference between the initial book value and the revenue, given by $C = q_0 S_0 - \sum_{t=0}^{T} q_t \widetilde{S}_t$ with mean and variance

$$\mathbb{E}[C] = \sum_{t=1}^{T} \big(q_t g(u_t) + u_t h(u_t)\big), \qquad \mathrm{Var}(C) = \sigma^2 \sum_{t=1}^{T} q_t^2.$$

A trader thus would like to minimize her expected cost as well as the variance of the cost, which gives the optimal execution problem in this model as

$$\min_{\{u_t\}_{t=1}^{T}} \big(\mathbb{E}[C] + \lambda \mathrm{Var}(C)\big),$$

where $\lambda \in \mathbb{R}$ is a measure of risk-aversion.

When we assume that both the price impacts are linear, that is

$$g(u_t) = \gamma u_t, \qquad h(u_t) = \eta u_t,$$

where γ and η are the permanent and temporary price impact parameters, [8] derive the general solution for the Almgren–Chriss model. This is given by

$$u_j = \frac{2\sinh\left(\frac{1}{2}\kappa\right)}{\sinh(\kappa T)}\cosh\left(\kappa\left(T - t_{j-\frac{1}{2}}\right)\right)q_0, \qquad j = 1, 2, \ldots, T,$$

with

$$\kappa = \cosh^{-1}\left(\frac{\tilde{\kappa}^2}{2} + 1\right), \qquad \tilde{\kappa}^2 = \frac{\lambda\sigma^2}{\eta(1 - \frac{\gamma}{2\eta})}.$$

The corresponding optimal inventory trajectory is

$$q_j = \frac{\sinh\left(\kappa(T - t_j)\right)}{\sinh(\kappa T)}q_0, \qquad j = 0, 1, \ldots, T. \tag{4.1}$$

The above Almgren–Chriss framework for liquidating a single asset can be extended to the case of multiple assets [8, Appendix A].

This simple version of the Almgren–Chriss framework has a closed-form solution but it relies heavily on the assumptions of the dynamics and the linear form of the permanent and temporary price impact. The mis-specification of the dynamics and market impacts may lead to undesirable strategies and potential losses. In addition, the solution in (4.1) is a pre-planned strategy that does not depend on real-time market conditions. Hence this strategy may miss certain opportunities when the market moves. This motivates the use of an RL approach which is more flexible and able to incorporate market conditions when making decisions.

Evaluation Criteria and Benchmark Algorithms. Before discussing the RL approach, we introduce several widely-used criteria to evaluate the performance of execution algorithms in the literature such as the Profit and Loss (PnL), Implementation Shortfall, and the Sharp ratio. The PnL is the *final* profit or loss induced by a given execution algorithm over the whole time period, which is made up of transactions at all time points. The Implementation Shortfall [167] for an execution algorithm is defined as the difference between the PnL of the algorithm and the PnL received by trading the entire amount of the asset instantly. The Sharpe ratio [180] is defined as the ratio of expected return to standard deviation of the return; thus it measures return per unit of risk. Two popular variants of the Sharpe ratio are the differential Sharpe ratio [155] and the Sortino ratio [187].

In addition, some classical pre-specified strategies are used as benchmarks to evaluate the performance of a given RL-based execution strategy. Popular choices include executions strategies based on Time-Weighted Average Price (TWAP) and Volume-Weighted Average Price (VWAP) as well as the Submit and Leave (SnL) policy where a trader places a sell order for all shares at a fixed limit order price, and goes to the market with any unexecuted shares remaining at time T.

RL Approach. We first provide a brief overview of the existing literature on RL for optimal execution. The most popular types of RL methods that have been used in optimal execution problems are Q-learning algorithms and (double) DQN [102, 159, 241, 108, 51, 181, 158]. Policy-based algorithms are also popular in this field, including (deep) policy gradient methods [100, 241], A2C [241], PPO [51, 140], and DDPG [235]. The benchmark strategies studied in these papers include the Almgren–Chriss solution [102, 100], the TWAP strategy [159, 51, 140], the VWAP strategy [140], and the SnL policy [158, 235]. In some models the trader is allowed to buy or sell the asset at each time point

[108, 241, 217, 56], whereas there are also many models where only one trading direction is allowed [158, 102, 100, 159, 51, 181, 235, 140]. The state variables are often composed of time stamp, the market attributes including (mid-)price of the asset and/or the spread, the inventory process and past returns. The control variables are typically set to be the amount of asset (using market orders) to trade and/or the relative price level (using limit orders) at each time point. Examples of reward functions include cash inflow or outflow (depending on whether we sell or buy) [181, 158], implementation shortfall [102], profit [56], Sharpe ratio [56], return [108], and PnL [217]. Popular choices of performance measure include Implementation Shortfall [102, 100], PnL (with a penalty term of transaction cost) [159, 217, 56], trading cost [158, 181], profit [56, 108], Sharpe ratio [56, 241], Sortino ratio [241], and return [241].

We now discuss some more details of the RL algorithms and experimental settings in the above papers. For value-based algorithms, [158] provided the first large scale empirical analysis of a RL method applied to optimal execution problems. They focused on a modified Q-learning algorithm to select price levels for limit order trading, which leads to significant improvements over simpler forms of optimization such as the SnL policies in terms of trading costs. [181] proposed a risk-averse RL algorithm for optimal liquidation, which can be viewed as a generalization of [158]. This algorithm achieves substantially lower trading costs over the period when the 2010 flash-crash happened compared with the risk-neutral RL algorithm in [158]. [102] combined the Almgren–Chriss solution and the Q-learning algorithm and showed that they are able to improve the Implementation Shortfall of the Almgren–Chriss solution by up to 10.3% on average, using LOB data which includes five price levels. [159] proposed a modified double DQN algorithm for optimal execution and showed that the algorithm outperforms the TWAP strategy on seven out of nine stocks using PnL as the performance measure. They added a single one-second time step ΔT at the end of the horizon to guarantee that all shares are liquidated over $T + \Delta T$. [108] designed a trading system based on DQN which determines both the trading direction and the number of shares to trade. Their approach was shown to increase the total profits by at least four times in four different index stocks compared with a benchmark trading model which trades a fixed number of shares each time. They also used *transfer* learning to avoid overfitting, where some knowledge/information is reused when learning in a related or similar situation.

For policy-based algorithms, [56] combined deep learning with RL to determine whether to sell, hold, or buy at each time point. In the first step of their model, neural networks are used to summarize the market features and in the second step the RL part makes trading decisions. The proposed method was shown to outperform several other deep learning and deep RL models in terms of PnL, total profits, and Sharpe ratio. They suggested that in practice, the Sharpe ratio was more recommended as the reward function compared with total profits. [235] used the DDPG algorithm for optimal execution over a short time horizon (two minutes) and designed a network to extract features from the market data. Experiments on real LOB data with 10 price levels show that the proposed approach significantly outperforms the existing methods, including the SnL policy (as baseline), the Q-learning algorithm, and the method in [102]. [140] proposed an adaptive framework based on PPO with neural networks including LSTM and fully-connected networks, and showed that the framework outperforms the baseline models including TWAP and VWAP, as well as several deep RL models on most of 14 US equities. [100] applied the (vanilla) policy gradient method to the LOB data of five stocks in different sectors and showed that they improve the Implementation Shortfall of the Almgren–Chriss solution by around 20%. [130] used neural networks to learn the mapping between the risk-aversion parameter and the optimal control with potential market impacts incorporated.

For comparison between value-based and policy-based algorithms, [51] explored double DQN and PPO algorithms in different market environments – when the benchmark TWAP is optimal, PPO is shown to converge to TWAP whereas double DQN may not; when TWAP is not optimal, both algorithms outperform this benchmark. [241] showed that DQN, policy gradient, and A2C outperform several baseline models including classical time-series momentum strategies, on test data of 50 liquid

futures contracts. Both continuous and discrete action spaces were considered in their work. They observed that DQN achieves the best performance and the second best is the A2C approach.

In addition, model-based RL algorithms have also been used for optimal execution. [217] built a profitable electronic trading agent to place buy and sell orders using model-based RL, which outperforms two benchmarks strategies in terms of PnL on LOB data. They used a recurrent neural network to learn the state transition probability. We note that multi-agent RL has also been used to address the optimal execution problem, see for example [17, 114].

4.3 Portfolio Optimization

In portfolio optimization problems, a trader needs to select and trade the best portfolio of assets in order to maximize some objective function, which typically includes the expected return and some measure of the risk. The benefit of investing in such portfolios is that the diversification of investments achieves higher return per unit of risk than only investing in a single asset (see, e.g. [245]).

Mean-Variance Portfolio Optimization. The first significant mathematical model for portfolio optimization is the *Markowitz* model [144], also called the *mean-variance* model, where an investor seeks a portfolio to maximize the expected total return for any given level of risk measured by variance. This is a single-period optimization problem and is then generalized to multi-period portfolio optimization problems in [157, 97, 175, 150, 190, 134]. In this mean-variance framework, the risk of a portfolio is quantified by the variance of the wealth and the optimal investment strategy is then sought to maximize the final wealth penalized by a variance term. The mean-variance framework is of particular interest because it not only captures both the portfolio return and the risk, but also suffers from the *time-inconsistency* problem [193, 205, 221]. That is the optimal strategy selected at time t is no longer optimal at time $s > t$ and the Bellman equation does not hold. A breakthrough was made in [134], in which they were the first to derive the analytical solution to the discrete-time multi-period mean-variance problem. They applied an *embedding* approach, which transforms the mean-variance problem to an LQ problem where classical approaches can be used to find the solution. The same approach was then used to solve the continuous-time mean-variance problem [244]. In addition to the embedding scheme, other methods including the consistent planning approach [18, 26] and the dynamically optimal strategy [165] have also been applied to solve the time-inconsistency problem arising in the mean-variance formulation of portfolio optimization.

Here we introduce the multi-period mean-variance portfolio optimization problem as given in [134]. Suppose there are n risky assets in the market and an investor enters the market at time 0 with initial wealth x_0. The goal of the investor is to reallocate his wealth at each time point $t = 0, 1, \ldots, T$ among the n assets to achieve the optimal trade off between the return and the risk of the investment. The random rates of return of the assets at t is denoted by $\boldsymbol{e}_t = (e_t^1, \ldots, e_t^n)^\top$, where e_t^i $(i = 1, \ldots, n)$ is the rate of return of the i-th asset at time t. The vectors $\boldsymbol{e}_t$, $t = 0, 1, \ldots, T-1$, are assumed to be statistically independent (this independence assumption can be relaxed, see, e.g. [221]) with known mean $\boldsymbol{\mu}_t = (\mu_t^1, \ldots, \mu_t^n)^\top \in \mathbb{R}^n$ and known standard deviation σ_t^i for $i = 1, \ldots, n$ and $t = 0, \ldots, T-1$. The covariance matrix is denoted by $\boldsymbol{\Sigma}_t \in \mathbb{R}^{n \times n}$, where $[\boldsymbol{\Sigma}_t]_{ii} = (\sigma_t^i)^2$ and $[\boldsymbol{\Sigma}_t]_{ij} = \rho_t^{ij} \sigma_t^i \sigma_t^j$ for $i, j = 1, \ldots, n$ and $i \neq j$, where ρ_t^{ij} is the correlation between assets i and j at time t. We write x_t for the wealth of the investor at time t and u_t^i $(i = 1, \ldots, n-1)$ for the amount invested in the i-th asset at time t. Thus the amount invested in the n-th asset is $x_t - \sum_{i=1}^{n-1} u_t^i$. An investment strategy is denoted by $\boldsymbol{u}_t = (u_t^1, u_t^2, \ldots, u_t^{n-1})^\top$ for $t = 0, 1, \ldots, T-1$, and the goal is to find an optimal strategy such that the portfolio return is maximized while minimizing the risk of the investment, that is,

$$\max_{\{\boldsymbol{u}_t\}_{t=0}^{T-1}} \mathbb{E}[x_T] - \phi \mathrm{Var}(x_T), \tag{4.2}$$

subject to

$$x_{t+1} = \sum_{i=1}^{n-1} e_t^i u_t^i + \left(x_t - \sum_{i=1}^{n-1} u_t^i \right) e_t^n, \qquad t = 0, 1, \ldots, T-1, \tag{4.3}$$

where ϕ is a weighting parameter balancing risk, represented by the variance of x_T, and return. As mentioned above, this framework is embedded into an LQ problem in [134], and the solution to the LQ problem gives the solution to the above problem (4.2)-(4.3). The analytical solution derived in [134] is of the following form

$$\boldsymbol{u}_t^*(x_t) = \alpha_t x_t + \beta_t,$$

where α_t and β_t are explicit functions of $\boldsymbol{\mu}_t$ and $\boldsymbol{\Sigma}_t$, which are omitted here and can be found in [134].

The above framework has been extended in different ways, for example, the risk-free asset can also be involved in the portfolio, and one can maximize the cumulative form of (4.2) rather than only focusing on the final wealth x_T. For more details about these variants and solutions, see [221]. In addition to the mean-variance framework, other major paradigms in portfolio optimization are the Kelly Criterion and Risk Parity. We refer to [176] for a review of these optimal control frameworks and popular model-free RL algorithms for portfolio optimization.

Note that the classical stochastic control approach for portfolio optimization problems across multiple assets requires both a realistic representation of the temporal dynamics of individual assets, as well as an adequate representation of their co-movements. This is extremely difficult when the assets belong to different classes (for example, stocks, options, futures, interest rates and their derivatives). On the other hand, the model-free RL approach does not rely on the specification of the joint dynamics across assets.

RL Approach. Both value-based methods such as Q-learning [60, 166], SARSA [166], and DQN [163], and policy-based algorithms such as DPG and DDPG [224, 110, 236, 137, 3] have been applied to solve portfolio optimization problems. The state variables are often composed of time, asset prices, asset past returns, current holdings of assets, and remaining balance. The control variables are typically set to be the amount/proportion of wealth invested in each component of the portfolio. Examples of reward signals include portfolio return [110, 166, 236], (differential) Sharpe ratio [60, 166], and profit [60]. The benchmark strategies include Constantly Rebalanced Portfolio (CRP) [236, 110] where at each period the portfolio is rebalanced to the initial wealth distribution among the assets, and the buy-and-hold or do-nothing strategy [163, 3] which does not take any action but rather holds the initial portfolio until the end. The performance measures studied in these papers include the Sharpe ratio [236, 211, 224, 110, 137, 163, 208], the Sortino ratio [236], portfolio returns [208, 224, 137, 163, 236, 3], portfolio values [110, 224, 166], and cumulative profits [60]. Some models incorporate the transaction costs [110, 137, 60, 163, 236, 3] and investments in the risk-free asset [236, 211, 208, 60, 110].

For value-based algorithms, [60] considered the portfolio optimization problems of a risky asset and a risk-free asset. They compared the performance of the Q-learning algorithm and a Recurrent RL (RRL) algorithm under three different value functions including the Sharpe ratio, differential Sharpe ratio, and profit. The RRL algorithm is a policy-based method which uses the last action as an input. They concluded that the Q-learning algorithm is more sensitive to the choice of value function and has less stable performance than the RRL algorithm. They also suggested that the (differential) Sharpe ratio is preferred rather than the profit as the reward function. [166] studied a two-asset personal retirement portfolio optimization problem, in which traders who manage retirement funds are restricted from making frequent trades and they can only access limited information. They tested the performance of three algorithms: SARSA and Q-learning methods with discrete state and action space that maximize either the portfolio return or differential Sharpe ratio, and a TD learning method with discrete state space and continuous action space that maximizes the portfolio return. The TD method

learns the portfolio returns by using a linear regression model and was shown to outperform the other two methods in terms of portfolio values. [163] proposed a portfolio trading strategy based on DQN which chooses to either hold, buy or sell a pre-specified quantity of the asset at each time point. In their experiments with two different three-asset portfolios, their trading strategy is superior to four benchmark strategies including the do-nothing strategy and a random strategy (take an action in the feasible space randomly) using performance measures including the cumulative return and the Sharpe ratio.

For policy-based algorithms, [110] proposed a framework combining neural networks with DPG. They used a so-called Ensemble of Identical Independent Evaluators (EIIE) topology to predict the potential growth of the assets in the immediate future using historical data which includes the highest, lowest, and closing prices of portfolio components. The experiments using real cryptocurrency market data showed that their framework achieves higher Sharpe ratio and cumulative portfolio values compared with three benchmarks including CRP and several published RL models. [224] explored the DDPG algorithm for the portfolio selection of 30 stocks, where at each time point, the agent can choose to buy, sell, or hold each stock. The DDPG algorithm was shown to outperform two classical strategies including the min-variance portfolio allocation method [231] in terms of several performance measures including final portfolio values, annualized return, and Sharpe ratio, using historical daily prices of the 30 stocks. [137] considered the DDPG, PPO, and policy gradient method with an adversarial learning scheme which learns the execution strategy using noisy market data. They applied the algorithms for portfolio optimization to five stocks using market data and the policy gradient method with adversarial learning scheme achieves higher daily return and Sharpe ratio than the benchmark CRP strategy. They also showed that the DDPG and PPO algorithms fail to find the optimal policy even in the training set. [236] proposed a model using DDPG, which includes prediction of the assets future price movements based on historical prices and synthetic market data generation using a *Generative Adversarial Network* (GAN) [84]. The model outperforms the benchmark CRP and the model considered in [110] in terms of several performance measures including Sharpe ratio and Sortino ratio. Using Actor-Critic methods, [3] combined the mean-variance framework (the actor determines the policy using the mean-variance framework) and the Kelly Criterion framework (the critic evaluates the policy using their growth rate). They studied eight policy-based algorithms including DPG, DDPG, and PPO, among which DPG was shown to achieve the best performance.

In addition to the above discrete-time models, [211] studied the entropy-regularized continuous-time mean-variance framework with one risky asset and one risk-free asset. They proved that the optimal policy is Gaussian with decaying variance and proposed an Exploratory Mean-Variance (EMV) algorithm, which consists of three procedures: policy evaluation, policy improvement, and a self-correcting scheme for learning the Lagrange multiplier. They showed that this EMV algorithm outperforms two benchmarks, including the analytical solution with estimated model parameters obtained from the classical maximum likelihood method [33, Section 9.3.2] and a DDPG algorithm, in terms of annualized sample mean and sample variance of the terminal wealth, and Sharpe ratio in their simulations. The continuous-time framework in [211] was then generalized in [208] to large scale portfolio selection setting with d risky assets and one risk-free asset. The optimal policy is shown to be multivariate Gaussian with decaying variance. They tested the performance of the generalized EMV algorithm on price data from the stocks in the S&P 500 with $d \geq 20$ and it outperforms several algorithms including DDPG.

4.4 Option Pricing and Hedging

Understanding how to price and hedge financial derivatives is a cornerstone of modern mathematical and computational finance due to its importance in the finance industry. A financial derivative is a contract that derives its value from the performance of an underlying entity. For example, a call or

put *option* is a contract which gives the holder the right, but not the obligation, to buy or sell an underlying asset or instrument at a specified strike price prior to or on a specified date called the expiration date. Examples of option types include European options which can only be exercised at expiry, and American options which can be exercised at any time before the option expires.

The Black-Scholes Model. One of the most important mathematical models for option pricing is the Black-Scholes or Black-Scholes-Merton (BSM) model [27, 149], in which we aim to find the price of a European option $V(S_t, t)$ $(0 \leq t \leq T)$ with underlying stock price S_t, expiration time T, and payoff at expiry $P(S_T)$. The underlying stock price S_t is assumed to be non-dividend paying and to follow a geometric Brownian motion

$$dS_t = \mu S_t dt + \sigma S_t dW_t,$$

where μ and σ are called the drift and volatility parameters of the underlying asset, and $W = \{W_t\}_{0 \leq t \leq T}$ is a standard Brownian motion defined on a filtered probability space $(\Omega, \mathcal{F}, \{\mathcal{F}\}_{0 \leq t \leq T}, \mathbb{P})$. The key idea of the BSM is that European options can be perfectly replicated by a continuously rebalanced portfolio of the underlying asset and a riskless asset, under the assumption that there are no market frictions, and that trading can take place continuously and in arbitrarily small quantities. If the derivative can be replicated, by analysing the cost of the hedging strategy, the derivative's price must satisfy the following Black-Scholes partial differential equation

$$\frac{\partial V}{\partial t}(S_t, t) + \frac{1}{2}\sigma^2 S_t^2 \frac{\partial^2 V}{\partial S^2}(S_t, t) + r S_t \frac{\partial V}{\partial S}(S_t, t) - r V(S_t, t) = 0, \tag{4.4}$$

with terminal condition $V(S_T, T) = P(S_T)$ and where r is the known (constant) risk-free interest rate. When we have a *call* option with payoff $P(S_T) = \max(S_T - K, 0)$, giving the option buyer the right to buy the underlying asset, the solution to the Black-Scholes equation is given by

$$V(S_t, t) = N(d_1) S_t - N(d_2) K e^{-r(T-t)},$$

where $N(\cdot)$ is the standard normal cumulative distribution function, and d_1 and d_2 are given by

$$d_1 = \frac{1}{\sigma\sqrt{T-t}} \left(\ln\left(\frac{S_t}{K}\right) + \left(r + \frac{\sigma^2}{2}\right)(T-t) \right), \qquad d_2 = d_1 - \sigma\sqrt{T-t}.$$

We refer to the survey [30] for details about extensions of the BSM model, other classical option pricing models, and numerical methods such as the Monte Carlo method.

In a complete market one can *hedge* a given derivative contract by buying and selling the underlying asset in the right way to eliminate risk, which ensures there is a unique price for the derivative. In the Black-Scholes analysis *delta* hedging is used, in which we hedge the risk of a call option by shorting Δ_t units of the underlying asset with $\Delta_t := \frac{\partial V}{\partial S}(S_t, t)$ (the sensitivity of the option price with respect to the asset price). It is also possible to use financial derivatives to *hedge* against the volatility of given positions in the underlying assets. However, in practice, we can only rebalance portfolios at discrete time points and frequent transactions may incur high costs. Therefore an optimal hedging strategy depends on the tradeoff between the hedging error and the transaction costs. It is worth mentioning that this is in a similar spirit to the mean-variance portfolio optimization framework introduced in Section 4.3. Much effort has been made to include the transaction costs using classical approaches such as dynamic programming, see, for example [133, 70, 103]. We also refer to [82] for the details about delta hedging under different model assumptions.

However, the above discussion addresses option pricing and hedging in a model-based way since there are strong assumptions made on the asset price dynamics and the form of transaction costs. In practice some assumptions of the BSM model are not realistic since: 1) transaction costs due to

commissions, market impact, and non-zero bid-ask spread exist in the real market; 2) the volatility is not constant; 3) short term returns typically have a heavy tailed distribution [48, 42]. Thus the resulting prices and hedges may suffer from model mis-specification when the real asset dynamics is not exactly as assumed and the transaction costs are difficult to model. Thus we will focus on a model-free RL approach that can address some of these issues.

RL Approach. RL methods including (deep) Q-learning [99, 59, 35, 136, 98], PPO [59], and DDPG [35] have been applied to find hedging strategies and price financial derivatives. The state variables often include asset price, current positions, option strikes, and time remaining to expiry. The control variable is often set to be the change in holdings. Examples of reward functions are (risk-adjusted) expected wealth/return [99, 98, 59, 120] (as in the mean-variance portfolio optimization), option payoff [136], and (risk-adjusted) hedging cost [35]. The benchmarks for pricing models are typically the BSM model [99, 98] and binomial option pricing model [61] (introduced in [50]). The learned hedging strategies are typically compared to a discrete approximation to the delta hedging strategy for the BSM model [35, 59, 31, 34], or the hedging strategy for the Heston model [31]. In contrast to the BSM model where the volatility is assumed to be constant, the Heston model assumes that the volatility of the underlying asset follows a particular stochastic process which leads to a semi-analytical solution. The performance measures for the hedging strategy used in RL papers include (expected) hedging cost/error/loss [35, 120, 31, 34], PnL [59, 120, 34], and average payoff [136]. Some practical issues have been taken into account in RL models, including transaction costs [31, 59, 35, 120] and position constraints such as round lotting [59] (where a round lot is a standard number of securities to be traded, such as 100 shares) and limits on the trading size (for example buy or sell up to 100 shares) [120].

For European options, [99] developed a discrete-time option pricing model called the QLBS (Q-Learner in Black-Scholes) model based on Fitted Q-learning (see Section 3.2). This model learns both the option price and the hedging strategy in a similar spirit to the mean-variance portfolio optimization framework. [98] extended the QLBS model in [99] by using Fitted Q-learning. They also investigated the model in a different setting where the agent infers the risk-aversion parameter in the reward function using observed states and actions. However, [99] and [98] did not consider transaction costs in their analysis. By contrast, [31] used deep neural networks to approximate an optimal hedging strategy under market frictions, including transaction costs, and *convex risk measures* [118] such as conditional value at risk. They showed that their method can accurately recover the optimal hedging strategy in the Heston model [104] without transaction costs and it can be used to numerically study the impact of proportional transaction costs on option prices. The learned hedging strategy is also shown to outperform delta hedging in the (daily recalibrated) BSM model in a simple setting, this is hedging an at-the-money European call option on the S&P 500 index. The method in [31] was extended in [37] to price and hedge a very large class of derivatives including vanilla options and exotic options in more complex environments (e.g. stochastic volatility models and jump processes). [120] found the optimal hedging strategy by optimizing a simplified version of the mean-variance objective function (4.2), subject to discrete trading, round lotting, and nonlinear transaction costs. They showed in their simulations that the learned hedging strategy achieves a much lower cost, with no significant difference in volatility of total PnL, compared to the delta hedging strategy. [59] extended the framework in [120] and tested the performance of DQN and PPO for European options with different strikes. Their simulation results showed that these models are superior to delta hedging in general, and out of all models, PPO achieves the best performance in terms of PnL, training time, and the amount of data needed for training. [34] formulated the optimal hedging problem as a Risk-averse Contextual k-Armed Bandit (R-CMAB) model (see the discussion of the contextual bandit problem in Section 2.2) and proposed a deep CMAB algorithm involving Thompson Sampling [200]. They showed that their algorithm outperforms DQN in terms of sample efficiency and hedging error when compared to delta

hedging (in the setting of the BSM model). Their learned hedging strategy was also shown to converge to delta hedging in the absence of risk adjustment, discretization error and transaction costs. [35] considered Q-learning and DDPG for the problem of hedging a short position in a call option when there are transaction costs. The objective function is set to be a weighted sum of the expected hedging cost and the standard deviation of the hedging cost. They showed that their approach achieves a markedly lower expected hedging cost but with a slightly higher standard deviation of the hedging cost when compared to delta hedging. In their simulations, the stock price is assumed to follow either geometric Brownian motion or a stochastic volatility model.

For American options, the key challenge is to find the optimal exercise strategy which determines when to exercise the option as this determines the price. [136] used the Least-Squares Policy Iteration (LSPI) algorithm [126] and the Fitted Q-learning algorithm to learn the exercise policy for American options. In their experiments for American put options using both real and synthetic data, the two algorithms gain larger average payoffs than the benchmark Longstaff-Schwartz method [143], which is the standard Least-Squares Monte Carlo algorithm. [136] also analyzed their approach from a theoretical perspective and derived a high probability, finite-time bound on their method. [61] then extended the work in [136] by combining random forests, a popular machine learning technique, with Monte Carlo simulation for pricing of both American options and *convertible bonds*, which are corporate bonds that can be converted to the stock of the issuing company by the bond holder. They showed that the proposed algorithm provides more accurate prices than several other methods including LSPI, Fitted Q-learning, and the Longstaff-Schwartz method.

4.5 Market Making

A market maker in a financial instrument is an individual trader or an institution that provides liquidity to the market by placing buy and sell limit orders in the LOB for that instrument while earning the bid-ask spread.

The objective in market making is different from problems in optimal execution (to target a position) or portfolio optimization (for long-term investing). Instead of profiting from identifying the correct price movement direction, the objective of a market maker is to profit from earning the bid-ask spread without accumulating undesirably large positions (known as inventory) [89]. A market maker faces three major sources of risk [92]. The inventory risk [11] refers to the risk of accumulating an undesirable large net inventory, which significantly increases volatility due to market movements. The execution risk [124] is the risk that limit orders may not get filled over a desired horizon. Finally, the adverse selection risk refers to the situation where there is a directional price movement that sweeps through the limit orders submitted by the market marker such that the price does not revert back by the end of the trading horizon. This may lead to a huge loss as the market maker in general needs to clear their inventory at the end of the horizon (typically the end of the day to avoid overnight inventory).

Stochastic Control Approach. Traditionally the theoretical study of market making strategies follows a stochastic control approach, where the LOB dynamics are modeled directly by some stochastic process and an optimal market making strategy that maximizes the market maker's expected utility can be obtained by solving the Hamilton–Jacobi–Bellman equation. See [11], [90] and [160], and [38, Chapter 10] for examples.

We follow the framework in [11] as an example to demonstrate the control formulation in continuous time and discuss its advantages and disadvantages.

Consider a high-frequency market maker trading on a single stock over a finite horizon T. Suppose the mid-price of this stock follows an arithmetic Brownian motion in that

$$dS_t = \sigma dW_t, \tag{4.5}$$

where σ is a constant and $W = \{W_t\}_{0\leq t\leq T}$ is a standard Brownian motion defined on a filtered probability space $(\Omega, \mathcal{F}, \{\mathcal{F}\}_{0\leq t\leq T}, \mathbb{P})$.

The market maker will continuously propose bid and ask prices, denoted by S_t^b and S_t^a respectively, and hence will buy and sell shares according to the rate of arrival of market orders at the quoted prices. Her inventory, which is the number of shares she holds, is therefore given by

$$q_t = N_t^b - N_t^a, \tag{4.6}$$

where N^b and N^a are the point processes (independent of W) giving the number of shares the market maker respectively bought and sold. Arrival rates obviously depend on the prices S_t^b and S_t^a quoted by the market maker and we assume that the intensities λ^b and λ^a associated respectively to N^b and N^a depend on the difference between the quoted prices and the reference price (i.e. $\delta_t^b = S_t - S_t^b$ and $\delta_t^a = S_t - S_t^a$) and are of the following form

$$\lambda^b(\delta^b) = A\exp(-k\delta^b) \text{ and } \lambda^a(\delta^a) = A\exp(-k\delta^a), \tag{4.7}$$

where A and k are positive constants that characterize the liquidity of the stock. Consequently the cash process of the market maker follows

$$dX_t = (S_t + \delta_t^a)dN_t^a - (S_t - \delta_t^b)dN_t^b. \tag{4.8}$$

Finally, the market maker optimizes a constant absolute risk aversion (CARA) utility function:

$$V(s,x,q,t) = \sup_{\{\delta_u^a,\delta_u^b\}_{t\leq u\leq T}\in\mathcal{U}} \mathbb{E}\Big[-\exp(-\gamma(X_T + q_T S_T))\Big| X_t = x, S_t = s, \text{ and } q_t = q\Big], \tag{4.9}$$

where $\mathcal{U}$ is the set of predictable processes bounded from below, γ is the absolute risk aversion coefficient characterizing the market maker, X_T is the amount of cash at time T and $q_T S_T$ is the evaluation of the (signed) remaining quantity of shares in the inventory at time T.

By applying the dynamic programming principle, the value function V solves the following Hamilton–Jacobi–Bellman equation:

$$\begin{cases} \partial_t V + \frac{1}{2}\partial_{ss} V + \max_{\delta^b} \lambda^b(\delta^b)[V(s, x-s+\delta^b, q+1, t) - u(s,x,q,t)] \\ \qquad\qquad + \max_{\delta^a} \lambda^a(\delta^a)[V(s, x+s+\delta^a, q-1, t) - u(s,x,q,t)] = 0, \\ V(s,x,q,T) = -\exp(-\gamma(x+qs)). \end{cases} \tag{4.10}$$

While (4.10), derived in [11], admits a (semi) closed-form solution which leads to nice insights about the problem, it all builds on the full analytical specification of the market dynamics. In addition, there are very few utility functions (e.g. exponential (CARA), power (CRRA), and quadratic) known in the literature that could possibly lead to closed-form solutions. The same issues arise in other work along this line ([90] and [160], and [38, Chapter 10]) where strong model assumptions are made about the prices or about the LOB or both. This requirement of full analytical specification means these papers are quite removed from realistic market making, as financial markets do not conform to any simple parametric model specification with fixed parameters.

RL Approach. For market making problems with an RL approach, most of the developments are centered around value-based methods such as the Q-learning algorithm [2, 188] and SARSA [188]. The state variables are often composed of bid and ask prices, current holdings of assets, order-flow imbalance,volatility, and some sophisticated market indices. The control variables are typically set to be the spread to post a pair of limit buy and limit sell orders. Examples of reward include PnL with inventory cost [2, 188, 189] or Implementation Shortfall with inventory cost [80].

The first RL method for market making was explored by [43] and the authors applied three RL algorithms and tested them in simulation environments: a Monte Carlo method, a SARSA method and an actor-critic method. For all three methods, the state variables include inventory of the market-maker, order imbalance on the market, and market quality measures. The actions are the changes in the bid/ask price to post the limit orders and the sizes of the limit buy/sell orders. The reward function is set as a linear combination of profit (to maximize), inventory risk (to minimize), and market qualities (to maximize). The authors found that SARSA and Monte Carlo methods work well in a simple simulation environment. The actor-critic method is more plausible in complex environments and generates stochastic policies that correctly adjust bid/ask prices with respect to order imbalance and effectively control the trade-off between the profit and the quoted spread (defined as the price difference to post limit buy orders and limit sell orders). Furthermore, the stochastic policies are shown to outperform deterministic policies in achieving a lower variance of the resulting spread. Later on, [2] designed a group of "spread-based" market making strategies parametrized by a minimum quoted spread. The strategies bet on the mean-reverting behavior of the mid-price and utilize the opportunities when the mid-price deviates from the price during the previous period. Then an online algorithm is used to pick in each period a minimum quoted spread. The states of the market maker are the current inventory and price data. The actions are the quantities and at what prices to offer in the market. It is assumed that the market maker interacts with a continuous double auction via an order book. The market maker can place both market and limit orders and is able to make and cancel orders after every price fluctuation. The authors provided structural properties of these strategies which allows them to obtain low regret relative to the best such strategy in hindsight which maximizes the realized rewards. [188] generalized the results in [43] and adopted several reinforcement learning algorithms (including Q-learning and SARSA) to improve the decisions of the market maker. In their framework, the action space contains ten actions. The first nine actions correspond to a pair of orders with a particular spread and bias in their prices. The final action allows the agent to clear their inventory using a market order. The states can be divided into two groups: agent states and market states. Agent states include inventory level and active quoting distances and market states include market (bid-ask) spread, mid-price move, book/queue imbalance, signed volume, volatility, and relative strength index. The reward function is designed as the sum of a symmetrically dampened PnL and an inventory cost. The idea of the symmetric damping is to disincentivize the agent from trend chasing and direct the agent towards spread capturing. A simulator of a financial market is constructed via direct reconstruction of the LOB from historical data. Although, since the market is reconstructed from historical data, simulated orders placed by an agent cannot impact the market. The authors compared their algorithm to a modified version of [2] and showed a significant empirical improvement from [2].

Some recent papers have focused on improving the robustness of market markers' strategies with respect to adversarial and volatile market conditions. [80] used perturbations by an opposing agent - the adversary - to render the market maker more robust to model uncertainty and consequently improve generalization. Meanwhile, [80] incorporated additional predictive signals in the states to improve the learning outcomes of market makers. In particular, the states include the agent's inventory, price range and trend predictions. In a similar way to the setting in [2] and [188], actions are represented by a certain choice of bid/ask orders relative to the current best bid/ask. The reward function both incentivizes spread-capturing (making round-trips) and discourages holding inventory. The first part of the reward is inspired by utility functions containing a running inventory-based penalty with an absolute value inventory penalty term which has a convenient Value at Risk (VAR) interpretation. The second part is a symmetrically dampened PnL which follows [188]. Experimental results on historical data demonstrate the superior reward-to-risk performance of the proposed framework over several standard market making benchmarks. More specifically, in these experiments, the resulting reinforcement learning agent achieves between 20-30% higher terminal wealth than the benchmarks while being exposed to only around 60% of their inventory risks. Also, [189] applied Adversarial

Reinforcement Learning (ARL) to a zero-sum game version of the control formulation (4.5)-(4.10). The adversary acts as a proxy for other market participants that would like to profit at the market maker's expense.

In addition to designing learning algorithms for the market maker in an unknown financial environment, RL algorithms can also be used to solve high-dimensional control problems for the market maker or to solve the control problem with the presence of a time-dependent rebate in the full information setting. In particular, [91] focused on a setting where a market maker needs to decide the optimal bid and ask quotes for a given universe of bonds in an OTC market. This problem is high-dimensional and other classical numerical methods including finite differences are inapplicable. The authors proposed a model-based Actor-Critic-like algorithm involving a deep neural network to numerically solve the problem. Similar ideas have been applied to market making problems in dark pools [16] (see the discussion on dark pools in Section 4.7). With the presence of a time-dependent rebate, there is no closed-form solution for the associated stochastic control problem of the market maker. Instead, [238] proposed a Hamiltonian-guided value function approximation algorithm to solve for the numerical solutions under this scenario.

Multi-agent RL algorithms are also used to improve the strategy for market making with a particular focus on the impact of competition from other market makers or the interaction with other types of market participant. See [76] and [164].

4.6 Robo-advising

Robo-advisors, or automated investment managers, are a class of financial advisers that provide online financial advice or investment management with minimal human intervention. They provide digital financial advice based on mathematical rules or algorithms. Robo-advisors have gained widespread popularity and emerged prominently as an alternative to traditional human advisers in recent years. The first robo-advisors were launched after the 2008 financial crisis when financial services institutions were facing the ensuing loss of trust from their clients. Examples of pioneering robo-advising firms include Betterment and Wealthfront. As of 2020, the value of assets under robo-management is highest in the United States and exceeded $650 billion [36].

The robo-advisor does not know the client's risk preference in advance but learns it while interacting with the client. The robo-advisor then improves its investment decisions based on its current estimate of the client's risk preference. There are several challenges in the application of robo-advising. Firstly, the client's risk preference may change over-time and may depend on the market returns and economic conditions. Therefore the robo-advisor needs to determine a frequency of interaction with the client that ensures a high level of consistency in the risk preference when adjusting portfolio allocations. Secondly, the robo-advisor usually faces a dilemma when it comes to either catering to the client's wishes, that is, investing according to the client's risk preference, or going against the client's wishes in order to seek better investment performance. Finally, there is also a subtle trade-off between the rate of information acquisition from the client and the accuracy of the acquired information. On the one hand, if the interaction does not occur at all times, the robo-advisor may not always have access to up-to-date information about the client's profile. On the other hand, information communicated to the robo-advisor may not be representative of the client's true risk aversion as the client is subject to behavioral biases.

Stochastic Control Approach. To address the above mentioned challenges, [36] proposed a stochastic control framework with four components: (i) a regime switching model of market returns, (ii) a mechanism of interaction between the client and the robo-advisor, (iii) a dynamic model (i.e., risk aversion process) for the client's risk preferences, and (vi) an optimal investment criterion. In this framework, the robo-advisor interacts repeatedly with the client and learns about changes in her risk

profile whose evolution is specified by (iii). The robo-advisor adopts a multi-period mean-variance investment criterion with a finite investment horizon based on the estimate of the client's risk aversion level. The authors showed the stock market allocation resulting from the dynamic mean-variance optimization consists of two components where the first component is akin to the standard single period Markowitz strategy whereas the second component is intertemporal hedging demand, which depends on the relationship between the current market return and future portfolio returns.

Note that although [36] focused on the stochastic control approach to tackle the robo-advising problem, the framework is general enough that some components (for example the mean-variance optimization step) may be replaced by an RL algorithm and the dependence on model specification can be potentially relaxed.

RL Approach. There are only a few references on robo-advising with an RL approach since this is still a relatively new topic. We review each paper with details. The first RL algorithm for a robo-advisor was proposed by [9] where the authors designed an exploration-exploitation algorithm to learn the investor's risk appetite over time by observing her portfolio choices in different market environments. The set of various market environments of interest is formulated as the state space $\mathcal{S}$. In each period, the robo-advisor places an investor's capital into one of several pre-constructed portfolios which can be viewed as the action space $\mathcal{A}$. Each portfolio decision reflects the robo-advisor's belief concerning the investor's true risk preference from a discrete set of possible risk aversion parameters $\Theta = \{\theta_i\}_{1 \leq i \leq |\Theta|}$. The investor interacts with the robo-advisor by portfolio selection choices, and such interactions are used to update the robo-advisor's estimate of the investor's risk profile. The authors proved that, with high probability, the proposed exploration-exploitation algorithm performs near optimally with the number of time steps depending polynomially on various model parameters.

[209] proposed an investment robo-advising framework consisting of two agents. The first agent, an inverse portfolio optimization agent, infers an investor's risk preference and expected return directly from historical allocation data using online inverse optimization. The second agent, a deep RL agent, aggregates the inferred sequence of expected returns to formulate a new multi-period mean-variance portfolio optimization problem that can be solved using a deep RL approach based on the DDPG method. The proposed investment pipeline was applied to real market data from April 1, 2016 to February 1, 2021 and was shown to consistently outperform the S&P 500 benchmark portfolio that represents the aggregate market optimal allocation.

As mentioned earlier in this subsection, learning the client's risk preference is challenging as the preference may depend on multiple factors and may change over time. [237] was dedicated to learning the risk preferences from investment portfolios using an inverse optimization technique. In particular, the proposed inverse optimization approach can be used to measure time varying risk preferences directly from market signals and portfolios. This approach is developed based on two methodologies: convex optimization based modern portfolio theory and learning the decision-making scheme through inverse optimization.

4.7 Smart Order Routing

In order to execute a trade of a given asset, market participants may have the opportunity to split the trade and submit orders to different venues, including both lit pools and dark pools, where this asset is traded. This could potentially improve the overall execution price and quantity. Both the decision and hence the outcome are influenced by the characteristics of different venues as well as the structure of transaction fees and rebates across different venues.

Dark Pools vs. Lit Pools. Dark pools are private exchanges for trading securities that are not accessible by the investing public. Also known as "dark pools of liquidity", the name of these exchanges

is a reference to their complete lack of transparency. Dark pools were created in order to facilitate block trading by institutional investors who did not wish to impact the markets with their large orders and obtain adverse prices for their trades. According to recent Securities and Exchange Commission (SEC) data, there were 59 registered Alternative Trading Systems (a.k.a. the "Dark Pools") with the SEC as of May 2021 of which there are three types: (1) Broker-Dealer-Owned Dark Pools, (2) Agency Broker or Exchange-Owned Dark Pools, and (3) Electronic Market Makers Dark Pools. Lit pools are effectively the opposite of dark pools. Unlike dark pools, where prices at which participants are willing to trade are not revealed, lit pools do display bid offers and ask offers in different stocks. Primary exchanges operate in such a way that available liquidity is displayed at all times and form the bulk of the lit pools available to traders.

For smart order routing (SOR) problems, the most important characteristics of different dark pools are the chances of being matched with a counterparty and the price (dis)advantages whereas the relevant characteristics of lit pools include the order flows, queue sizes, and cancellation rates.

There are only a few references on using a data-driven approach to tackle SOR problems for dark pool allocations and for allocations across lit pools. We will review each of them with details.

Allocation Across Lit Pools. The SOR problem across multiple lit pools (or primary exchanges) was first studied in [49] where the authors formulated the SOR problem as a convex optimization problem.

Consider a trader who needs to buy S shares of a stock within a short time interval $[0, T]$. At time 0, the trader may submit K limit orders with $L_k \geq 0$ shares to exchanges $k = 1, \cdots, K$ (joining the queue of the best bid price level) and also market orders for $M \geq 0$ shares. At time T if the total executed quantity is less than S the trader also submits a market order to execute the remaining amount. The trader's order placement decision is thus summarized by a vector $X = (M, L_1, \cdots, L_K) \in \mathbb{R}_+^{K+1}$ of order sizes. It is assumed for simplicity that a market order of any size up to S can be filled immediately at any single exchange. Thus a trader chooses the cheapest venue for his market orders.

Limit orders with quantities $(L_1, \cdots, L_K)$ join queues of $(Q_1, \cdots, Q_K)$ orders in K limit order books, where $Q_k \geq 0$. Then the filled amounts at the end of the horizon T can be written as a function of their initial queue position and future order flow:

$$\min(\max(\xi_k - Q_k, 0), L_k) = (\xi_k - Q_k)_+ - (\xi_k - Q_k - L_k)_+ \tag{4.11}$$

where the notation $(x)_+ = \max(0, x)$. Here $\xi_k := D_k + C_k$ is the total outflow from the front of the k-th queue which consists of order cancellations C_k that occurred before time T from queue positions in front of an order and market orders D_k that reach the k-th exchange before T. Note that the fulfilled amounts are random because they depend on queue outflows $\xi = (\xi_1, \cdots, \xi_K)$ during $[0, T]$, which are modeled as random variables with a distribution F.

Using the mid-quote price as a benchmark, the execution cost relative to the mid-quote for an order allocation $X = (M, L_1, ..., L_K)$ is defined as:

$$V_{\text{execution}}(X, \xi) := (h + f)M - \sum_{k=1}^{K} (h + r_k)((\xi_k - Q_k)_+ - (\xi_k - Q_k - L_k)_+), \tag{4.12}$$

where h is one-half of the bid-ask spread at time 0, f is a fee for market orders and r_k are effective rebates for providing liquidity by submitting limit orders on exchanges $k = 1, \cdots, K$.

Penalties for violations of the target quantity in both directions are included:

$$V_{\text{penalty}}(X, \xi) := \lambda_u(S - A(X, \xi))_+ + \lambda_o(A(X, \xi) - S)_+, \tag{4.13}$$

where $\lambda_o \geq 0$ and $\lambda_u \geq 0$ are, respectively, the penalty for overshooting and undershooting, and $A(X,\xi) = M + \sum_{k=1}^{K}(h+r_k)((\xi_k - Q_k)_+ - (\xi_k - Q_k - L_k)_+)$ is the total number of shares bought by the trader during $[0,T)$.

The impact cost is paid on all orders placed at times 0 and T, irrespective of whether they are filled, leading to the following total impact:

$$V_{\text{impact}}(X,\xi) = \theta(M + L_k + (S - A(X,\xi)_+)) \tag{4.14}$$

where $\theta > 0$ is the impact coefficient.

Finally, the cost function is defined as the summation of all three pieces $V(X,\xi) := V_{\text{execution}}(X,\xi) + V_{\text{penalty}}(X,\xi) + V_{\text{impact}}(X,\xi)$. [49, Proposition 4] provides optimality conditions for an order allocation $X^* = (M^*, L_1, \cdots, L_K)$. In particular, (semi)-explicit model conditions are given for when $L_k^* > 0$ ($M^* > 0$), i.e., when submitting limit orders to venue k (when submitting market orders) is optimal.

In a different approach to the single-period model introduced above, [15] formulated the SOR problem as an order allocation problem across multiple lit pools over multiple trading periods. Each venue is characterized by a bid-ask spread process and an imbalance process. The dependencies between the imbalance and spread at the venues are considered through a covariance matrix. A Bayesian learning framework for learning and updating the model parameters is proposed to take into account possibly changing market conditions. Extensions to include short/long trading signals, market impact or hidden liquidity are also discussed.

Allocation Across Dark Pools. As discussed at the beginning of Section 4.7, dark pools are a type of stock exchange that is designed to facilitate large transactions. A key aspect of dark pools is the *censored feedback* that the trader receives. At every iteration the trader has a certain number V_t of shares to allocate amongst K different dark pools with v_t^i the volume allocated to dark pool i. The dark pool i trades as many of the allocated shares v_t^i as it can with the available liquidity s_t^i. The trader only finds out how many of these allocated shares were successfully traded at each dark pool (i.e., $\min(s_t^i, v_t^i)$), but not how many would have been traded if more were allocated (i.e., s_t^i).

Based on this property of censored feedback, [75] formulated the allocation problem across dark pools as an online learning problem under the assumption that s_t^i is an i.i.d. sample from some unknown distribution P_i and the total allocation quantity V_t is an i.i.d. sample from an unknown distribution Q with V_t upper bounded by a constant $V > 0$ almost surely. At each iteration t, the learner allocates the orders greedily according to the estimate $\widehat{P}_i^{(t-1)}$ of the distribution P_i for all dark pools $i = 1, 2, \cdots, K$ derived from previous iteration $t-1$. Then the learner can update the estimation $\widehat{P}_i^{(t)}$ of the distribution P_i with a modified version of the Kaplan-Meier estimate (a non-parametric statistic used to estimate the cumulative probability) with the new censored observation $\min(s_t^i, v_t^i)$ from iteration t. The authors then proved that for any $\varepsilon > 0$ and $\delta > 0$, with probability $1 - \delta$ (over the randomness of draws from Q and $\{P_i\}_{i=1}^K$), after running for a time polynomial in $K, V, 1/\varepsilon$, and $\ln(1/\delta)$, the algorithm makes an ε-optimal allocation on each subsequent time step with probability at least $1 - \varepsilon$.

The setup of [75] was generalized in [5] where the authors assumed that the sequences of volumes V_t and available liquidities $\{s_t^i\}_{i=1}^K$ are chosen by an adversary who knows the previous allocations of their algorithm. An exponentiated gradient style algorithm was proposed and shown to enjoy an optimal regret guarantee $\mathcal{O}(V\sqrt{T \ln K})$ against the best allocation strategy in hindsight.

5 Further Developments for Mathematical Finance and Reinforcement Learning

The general RL algorithms developed in the machine learning literature are good starting points for use in financial applications. A possible drawback is that such general RL algorithms tend to learn "more information" than actually required for a particular application. On the other hand, the stochastic control approach to many financial decision making problems may suffer from the risk of model misspecification. However, it may capture the essential features of a given financial application from a modeling perspective, in terms of the dynamics and the reward function.

One promising direction for RL in finance is to develop an even closer integration of the modeling techniques (the domain knowledge) from the stochastic control literature and key components of a given financial application (for example the adverse selection risk for market-making problems and the execution risk for optimal liquidation problems) with the learning power of the RL algorithms. This line of developing a more integrated framework is interesting from both theoretical and applications perspectives. From the application point of view, a modified RL algorithm, with designs tailored to one particular financial application, could lead to better empirical performance. This could be verified by comparison with existing algorithms on the available datasets. In addition, financial applications motivate potential new frameworks and playgrounds for RL algorithms. Carrying out the convergence and sample complexity analysis for these modified algorithms would also be a meaningful direction in which to proceed. Many of the papers referenced in this review provide great initial steps in this direction. We list the following future directions that the reader may find interesting.

Risk-aware or Risk-sensitive RL. Risk arises from the uncertainties associated with future events, and is inevitable since the consequences of actions are uncertain at the time when a decision is made. Many decision-making problems in finance lead to trading strategies and it is important to account for the risk of the proposed strategies (which could be measured for instance by the maximum draw-down, the variance or the 5% percentile of the PnL distribution) and/or the risk from the market environment such as the adverse selection risk.

Hence it would be interesting to include risk measures in the design of RL algorithms for financial applications. The challenge of risk-sensitive RL lies both in the non-linearity of the objective function with respect to the reward and in designing a risk-aware exploration mechanism.

RL with risk-sensitive utility functions has been studied in several papers without regard to specific financial applications. The work of [151] proposes TD(0) and Q-learning-style algorithms that transform temporal differences instead of cumulative rewards, and proves their convergence. Risk-sensitive RL with a general family of utility functions is studied in [182], which also proposes a Q-learning algorithm with convergence guarantees. The work of [62] studies a risk-sensitive policy gradient algorithm, though with no theoretical guarantees. [68] considers the problem of risk-sensitive RL with exponential utility and proposes two efficient model-free algorithms, Risk-sensitive Value Iteration (RSVI) and Risk-sensitive Q-learning (RSQ), with a near-optimal sample complexity guarantee. [203] developed a martingale approach to learn policies that are sensitive to the uncertainty of the rewards and are meaningful under some market scenarios. Another line of work focuses on constrained RL problems with different risk criteria [4, 46, 47, 57, 198, 242]. Very recently, [107] proposed a robust risk-aware reinforcement learning framework via robust optimization and with a rank dependent expected utility function. Financial applications such as statistical arbitrage and portfolio optimization are discussed with detailed numerical examples. However, there is no sample complexity or asymptotic convergence studied for the proposed algorithm.

Offline Learning and Online Exploration. Online learning requires updating of algorithm parameters in real-time and this is impractical for many financial decision-making problems, especially in the high-frequency regime. The most plausible setting is to collect data with a pre-specified exploration scheme during trading hours and update the algorithm with the new collected data after the close of trading. This is closely related to the translation of online learning to offline regression [186] and RL with batch data [45, 78, 79, 172]. However, these developments focus on general methodologies without being specifically tailored to financial applications.

Learning with a Limited Exploration Budget. Exploration can help agents to find new policies to improve their future cumulative rewards. However, too much exploration can be both time consuming and computation consuming, and in particular, it may be very costly for some financial applications. Additionally, exploring black-box trading strategies may need a lot of justification within a financial institution and hence investors tend to limit the effort put into exploration and try to improve performance as much as possible within a given budget for exploration. This idea is similar in spirit to conservative RL where agents explore new strategies to maximize revenue whilst simultaneously maintaining their revenue above a fixed baseline, uniformly over time [220]. This is also related to the problem of information acquisition with a cost which has been studied for economic commodities [169] and operations management [115]. It may also be interesting to investigate such costs for decision-making problems in financial markets.

Learning with Multiple Objectives. In finance, a common problem is to choose a portfolio when there are two conflicting objectives - the desire to have the expected value of portfolio returns be as high as possible, and the desire to have risk, often measured by the standard deviation of portfolio returns, be as low as possible. This problem is often represented by a graph in which the efficient frontier shows the best combinations of risk and expected return that are available, and in which indifference curves show the investor's preferences for various risk-expected return combinations. Decision makers sometimes combine both criteria into a single objective function consisting of the difference of the expected reward and a scalar multiple of the risk. However, it may well not be in the best interest of a decision maker to combine relevant criteria in a linear format for certain applications. For example, market makers on the OTC market tend to view criteria such as turn around time, balance sheet constraints, inventory cost, profit and loss as separate objective functions. The study of multi-objective RL is still at a preliminary stage and relevant references include [243] and [234].

Learning to Allocate Across Lit Pools and Dark Pools. Online optimization methods explored in [5] and [75] for dark pool allocations can be viewed as a single-period RL algorithm and the Bayesian framework developed in [15] for allocations across lit pools may be classified as a model-based RL approach. However, there is currently no existing work on applying multi-period and model-free RL methods to learn how to route orders across both dark pools and lit pools. We think this might be an interesting direction to explore as agents sometimes have access to both lit pools and dark pools and these two contrasting pools have quite different information structures and matching mechanisms.

Robo-advising in a Model-free Setting. As introduced in Section 4.6, [9] considered learning within a set of m pre-specified investment portfolios, and [209] and [237] developed learning algorithms and procedures to infer risk preferences, respectively, under the framework of Markowitz mean-variance portfolio optimization. It would be interesting to consider a model-free RL approach where the robo-advisor has the freedom to learn and improve decisions beyond a pre-specified set of strategies or the Markowitz framework.

Sample Efficiency in Learning Trading Strategies. In recent years, sample complexity has been studied extensively to understand modern reinforcement learning algorithms (see Sections 2-3). However, most RL algorithms still require a large number of samples to train a decent trading algorithm, which may exceed the amount of relevant available historical data. Financial time series are known to be non-stationary [105], and hence historical data that are further away in time may not be helpful in training efficient learning algorithms for the current market environment. This leads to important questions of designing more sample-efficient RL algorithms for financial applications or developing good market simulators that could generate (unlimited) realistic market scenarios [218].

Transfer Learning and Cold Start for Learning New Assets. Financial institutions or individuals may change their baskets of assets to trade over time. Possible reasons may be that new assets (for example cooperative bonds) are issued from time to time or the investors may switch their interest from one sector to another. There are two interesting research directions related to this situation. When an investor has a good trading strategy, trained by an RL algorithm for one asset, how should they transfer the experience to train a trading algorithm for a "similar" asset with fewer samples? This is closely related to transfer learning [202, 161]. To the best of our knowledge, no study for financial applications has been carried out along this direction. Another question is the cold-start problem for newly issued assets. When we have very limited data for a new asset, how should we initialize an RL algorithm and learn a decent strategy using the limited available data and our experience (i.e., the trained RL algorithm or data) with other longstanding assets?

Acknowledgement We thank Xuefeng Gao, Anran Hu, Wenpin Tang, Zhuoran Yang, Junzi Zhang and Zeyu Zheng for helpful discussions and comments on this survey.

References

[1] Y. Abbasi-Yadkori, P. Bartlett, K. Bhatia, N. Lazic, C. Szepesvari, and G. Weisz, *Politex: Regret bounds for policy iteration using expert prediction*, in International Conference on Machine Learning, PMLR, 2019, pp. 3692–3702.

[2] J. D. Abernethy and S. Kale, *Adaptive market making via online learning*, in NIPS, Citeseer, 2013, pp. 2058–2066.

[3] A. M. Aboussalah, *What is the value of the cross-sectional approach to deep reinforcement learning?*, Available at SSRN, (2020).

[4] J. Achiam, D. Held, A. Tamar, and P. Abbeel, *Constrained policy optimization*, in International Conference on Machine Learning, PMLR, 2017, pp. 22–31.

[5] A. Agarwal, P. Bartlett, and M. Dama, *Optimal allocation strategies for the dark pool problem*, in Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, JMLR Workshop and Conference Proceedings, 2010, pp. 9–16.

[6] A. Agarwal, S. Kakade, and L. F. Yang, *Model-based reinforcement learning with a generative model is minimax optimal*, in Conference on Learning Theory, PMLR, 2020, pp. 67–83.

[7] A. Agarwal, S. M. Kakade, J. D. Lee, and G. Mahajan, *On the theory of policy gradient methods: Optimality, approximation, and distribution shift*, Journal of Machine Learning Research, 22 (2021), pp. 1–76.

[8] R. Almgren and N. Chriss, *Optimal execution of portfolio transactions*, Journal of Risk, 3 (2001), pp. 5–40.

[9] H. Alsabah, A. Capponi, O. Ruiz Lacedelli, and M. Stern, *Robo-advising: Learning investors' risk preferences via portfolio choices*, Journal of Financial Econometrics, 19 (2021), pp. 369–392.

[10] K. Asadi and M. L. Littman, *An alternative softmax operator for reinforcement learning*, in International Conference on Machine Learning, PMLR, 2017, pp. 243–252.

[11] M. Avellaneda and S. Stoikov, *High-frequency trading in a limit order book*, Quantitative Finance, 8 (2008), pp. 217–224.

[12] M. G. Azar, R. Munos, and B. Kappen, *On the sample complexity of reinforcement learning with a generative model*, arXiv preprint arXiv:1206.6461, (2012).

[13] M. G. Azar, R. Munos, and H. J. Kappen, *Minimax PAC bounds on the sample complexity of reinforcement learning with a generative model*, Machine Learning, 91 (2013), pp. 325–349.

[14] M. G. Azar, I. Osband, and R. Munos, *Minimax regret bounds for reinforcement learning*, in International Conference on Machine Learning, PMLR, 2017, pp. 263–272.

[15] B. Baldacci and I. Manziuk, *Adaptive trading strategies across liquidity pools*, arXiv preprint arXiv:2008.07807, (2020).

[16] B. Baldacci, I. Manziuk, T. Mastrolia, and M. Rosenbaum, *Market making and incentives design in the presence of a dark pool: A deep reinforcement learning approach*, arXiv preprint arXiv:1912.01129, (2019).

[17] W. Bao and X.-y. Liu, *Multi-agent deep reinforcement learning for liquidation strategy analysis*, arXiv preprint arXiv:1906.11046, (2019).

[18] S. Basak and G. Chabakauri, *Dynamic mean-variance asset allocation*, The Review of Financial Studies, 23 (2010), pp. 2970–3016.

[19] M. Basei, X. Guo, A. Hu, and Y. Zhang, *Logarithmic regret for episodic continuous-time linear-quadratic reinforcement learning over a finite-time horizon*, Available at SSRN 3848428, (2021).

[20] C. L. Beck and R. Srikant, *Error bounds for constant step-size Q-learning*, Systems & Control Letters, 61 (2012), pp. 1203–1208.

[21] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, *The Arcade Learning Environment: An evaluation platform for general agents*, Journal of Artificial Intelligence Research, 47 (2013), pp. 253–279.

[22] D. A. Berry and B. Fristedt, *Bandit problems: Sequential allocation of experiments (Monographs on Statistics and Applied Probability)*, London: Chapman and Hall, 5 (1985), pp. 7–7.

[23] J. Bhandari and D. Russo, *Global optimality guarantees for policy gradient methods*, arXiv preprint arXiv:1906.01786, (2019).

[24] J. Bhandari, D. Russo, and R. Singal, *A finite time analysis of temporal difference learning with linear function approximation*, in Conference on Learning Theory, PMLR, 2018, pp. 1691–1692.

[25] S. Bhatnagar, *An actor–critic algorithm with function approximation for discounted cost constrained Markov decision processes*, Systems & Control Letters, 59 (2010), pp. 760–766.

[26] T. Bjork and A. Murgoci, *A general theory of Markovian time inconsistent stochastic control problems*, Available at SSRN 1694759, (2010).

[27] F. Black and M. Scholes, *The pricing of options and corporate liabilities*, Journal of Political Economy, 81 (1973), pp. 637–654.

[28] S. J. Bradtke and A. G. Barto, *Linear least-squares algorithms for temporal difference learning*, Machine learning, 22 (1996), pp. 33–57.

[29] R. I. Brafman and M. Tennenholtz, *R-max-a general polynomial time algorithm for near-optimal reinforcement learning*, Journal of Machine Learning Research, 3 (2002), pp. 213–231.

[30] M. Broadie and J. B. Detemple, *Anniversary article: Option pricing: Valuation models and applications*, Management Science, 50 (2004), pp. 1145–1177.

[31] H. Buehler, L. Gonon, J. Teichmann, and B. Wood, *Deep hedging*, Quantitative Finance, 19 (2019), pp. 1271–1291.

[32] Q. Cai, Z. Yang, J. Lee, and Z. Wang, *Neural temporal-difference learning converges to global optima*, in Advances in Neural Information Processing Systems, 2019.

[33] J. Y. Campbell, A. W. Lo, and A. C. MacKinlay, *The econometrics of financial markets*, Princeton University Press, 1997.

[34] L. Cannelli, G. Nuti, M. Sala, and O. Szehr, *Hedging using reinforcement learning: Contextual k-armed bandit versus Q-learning*, arXiv preprint arXiv:2007.01623, (2020).

[35] J. Cao, J. Chen, J. Hull, and Z. Poulos, *Deep hedging of derivatives using reinforcement learning*, The Journal of Financial Data Science, 3 (2021), pp. 10–27.

[36] A. Capponi, S. Olafsson, and T. Zariphopoulou, *Personalized robo-advising: Enhancing investment through client interaction*, Management Science, (2021).

[37] A. Carbonneau and F. Godin, *Equal risk pricing of derivatives with deep hedging*, Quantitative Finance, 21 (2021), pp. 593–608.

[38] Á. Cartea, S. Jaimungal, and J. Penalva, *Algorithmic and high-frequency trading*, Cambridge University Press, 2015.

[39] Á. Cartea, S. Jaimungal, and L. Sánchez-Betancourt, *Deep reinforcement learning for algorithmic trading*, Available at SSRN, (2021).

[40] S. Cayci, S. Satpathi, N. He, and R. Srikant, *Sample complexity and overparameterization bounds for projection-free neural TD learning*, arXiv preprint arXiv:2103.01391, (2021).

[41] S. Cen, C. Cheng, Y. Chen, Y. Wei, and Y. Chi, *Fast global convergence of natural policy gradient methods with entropy regularization*, arXiv preprint arXiv:2007.06558, (2020).

[42] A. Chakraborti, I. M. Toke, M. Patriarca, and F. Abergel, *Econophysics review: I. empirical facts*, Quantitative Finance, 11 (2011), pp. 991–1012.

[43] N. T. Chan and C. Shelton, *An electronic market-maker*, (2001).

[44] A. Charpentier, R. Elie, and C. Remlinger, *Reinforcement learning in economics and finance*, Computational Economics, (2021), pp. 1–38.

[45] J. Chen and N. Jiang, *Information-theoretic considerations in batch reinforcement learning*, in International Conference on Machine Learning, PMLR, 2019, pp. 1042–1051.

[46] Y. Chow, M. Ghavamzadeh, L. Janson, and M. Pavone, *Risk-constrained reinforcement learning with percentile risk criteria*, The Journal of Machine Learning Research, 18 (2017), pp. 6070–6120.

[47] Y. Chow, A. Tamar, S. Mannor, and M. Pavone, *Risk-sensitive and robust decision-making: A CVaR optimization approach*, in NIPS'15, MIT Press, 2015, pp. 1522–1530.

[48] R. Cont, *Empirical properties of asset returns: stylized facts and statistical issues*, Quantitative Finance, 1 (2001), p. 223.

[49] R. Cont and A. Kukanov, *Optimal order placement in limit order markets*, Quantitative Finance, 17 (2017), pp. 21–39.

[50] J. C. Cox, S. A. Ross, and M. Rubinstein, *Option pricing: A simplified approach*, Journal of Financial Economics, 7 (1979), pp. 229–263.

[51] K. Dabérius, E. Granat, and P. Karlsson, *Deep execution-value and policy based reinforcement learning for trading and beating market benchmarks*, Available at SSRN 3374766, (2019).

[52] B. Dai, A. Shaw, L. Li, L. Xiao, N. He, Z. Liu, J. Chen, and L. Song, *SBEED: Convergent reinforcement learning with nonlinear function approximation*, in International Conference on Machine Learning, PMLR, 2018, pp. 1125–1134.

[53] G. Dalal, B. Szörényi, G. Thoppe, and S. Mannor, *Finite sample analyses for TD(0) with function approximation*, in 32th AAAI Conference on Artificial Intelligence, 2018.

[54] C. Dann and E. Brunskill, *Sample complexity of episodic fixed-horizon reinforcement learning*, in NIPS'15, Cambridge, MA, USA, 2015, MIT Press, pp. 2818–2826.

[55] C. Dann, T. Lattimore, and E. Brunskill, *Unifying PAC and regret: Uniform PAC bounds for episodic reinforcement learning*, in Proceedings of the 31st International Conference on Neural Information Processing Systems, NIPS'17, 2017, pp. 5717–5727.

[56] Y. Deng, F. Bao, Y. Kong, Z. Ren, and Q. Dai, *Deep direct reinforcement learning for financial signal representation and trading*, IEEE Transactions on Neural Networks and Learning Systems, 28 (2017), pp. 653–664.

[57] D. Ding, X. Wei, Z. Yang, Z. Wang, and M. Jovanovic, *Provably efficient safe exploration via primal-dual policy optimization*, in International Conference on Artificial Intelligence and Statistics, PMLR, 2021, pp. 3304–3312.

[58] M. F. Dixon, I. Halperin, and P. Bilokon, *Machine Learning in Finance*, Springer, 2020.

[59] J. Du, M. Jin, P. N. Kolm, G. Ritter, Y. Wang, and B. Zhang, *Deep reinforcement learning for option replication and hedging*, The Journal of Financial Data Science, 2 (2020), pp. 44–57.

[60] X. Du, J. Zhai, and K. Lv, *Algorithm trading using Q-learning and recurrent reinforcement learning*, Positions, 1 (2016), p. 1.

[61] B. Dubrov, *Monte Carlo simulation with machine learning for pricing American options and convertible bonds*, Available at SSRN 2684523, (2015).

[62] H. Eriksson and C. Dimitrakakis, *Epistemic risk-sensitive reinforcement learning*, arXiv preprint arXiv:1906.06273, (2019).

[63] E. Even-Dar, Y. Mansour, and P. Bartlett, *Learning rates for Q-learning*, Journal of Machine Learning Research, 5 (2003).

[64] J. Fan, C. Ma, and Y. Zhong, *A selective overview of deep learning*, Statistical Science, 36 (2021).

[65] J. Fan, Z. Wang, Y. Xie, and Z. Yang, *A theoretical analysis of deep Q-learning*, in Learning for Dynamics and Control, PMLR, 2020, pp. 486–489.

[66] A. M. Farahmand, M. Ghavamzadeh, C. Szepesvàri, and S. Mannor, *Regularized policy iteration*, in Advances in Neural Information Processing Systems 21 - Proceedings of the 2008 Conference, 01 2008, pp. 441–448.

[67] M. Fazel, R. Ge, S. M. Kakade, and M. Mesbahi, *Global convergence of policy gradient methods for the linear quadratic regulator*, in International Conference on Machine Learning, PMLR, 2018, pp. 1467–1476.

[68] Y. Fei, Z. Yang, Y. Chen, Z. Wang, and Q. Xie, *Risk-sensitive reinforcement learning: Near-optimal risk-sample tradeoff in regret*, in NeurIPS, 2020.

[69] J.-D. Fermanian, O. Guéant, and A. Rachez, *Agents' Behavior on Multi-Dealer-to-Client Bond Trading Platforms*, CREST, Center for Research in Economics and Statistics, 2015.

[70] S. Figlewski, *Options arbitrage in imperfect markets*, The Journal of Finance, 44 (1989), pp. 1289–1311.

[71] T. G. Fischer, *Reinforcement learning in financial markets-a survey*, tech. rep., FAU Discussion Papers in Economics, 2018.

[72] V. François-Lavet, P. Henderson, R. Islam, M. G. Bellemare, and J. Pineau, *An introduction to deep reinforcement learning*, Foundations and Trends in Machine Learning, 11 (2018), pp. 219–354.

[73] V. François-Lavet, G. Rabusseau, J. Pineau, D. Ernst, and R. Fonteneau, *On overfitting and asymptotic bias in batch reinforcement learning with partial observability*, Journal of Artificial Intelligence Research, 65 (2019), pp. 1–30.

[74] Z. Fu, Z. Yang, and Z. Wang, *Single-timescale actor-critic provably finds globally optimal policy*, in International Conference on Learning Representations, 2021.

[75] K. Ganchev, Y. Nevmyvaka, M. Kearns, and J. W. Vaughan, *Censored exploration and the dark pool problem*, Communications of the ACM, 53 (2010), pp. 99–107.

[76] S. Ganesh, N. Vadori, M. Xu, H. Zheng, P. Reddy, and M. Veloso, *Reinforcement learning for market making in a multi-agent dealer market*, arXiv preprint arXiv:1911.05892, (2019).

[77] X. Gao, Z. Q. Xu, and X. Y. Zhou, *State-dependent temperature control for langevin diffusions*, arXiv preprint arXiv:2011.07456, (2020).

[78] Z. Gao, Y. Han, Z. Ren, and Z. Zhou, *Batched multi-armed bandits problem*, in Advances in Neural Information Processing Systems, vol. 32, 2019.

[79] E. Garcelon, M. Ghavamzadeh, A. Lazaric, and M. Pirotta, *Conservative exploration in reinforcement learning*, in International Conference on Artificial Intelligence and Statistics, PMLR, 2020, pp. 1431–1441.

[80] B. Gašperov and Z. Kostanjčar, *Market making with signals through deep reinforcement learning*, IEEE Access, 9 (2021), pp. 61611–61622.

[81] M. Geist, B. Scherrer, and O. Pietquin, *A theory of regularized Markov decision processes*, in International Conference on Machine Learning, PMLR, 2019, pp. 2160–2169.

[82] A. Giurca and S. Borovkova, *Delta hedging of derivatives using deep reinforcement learning*, Available at SSRN 3847272, (2021).

[83] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*, MIT press, 2016.

[84] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, *Generative adversarial nets*, Advances in Neural Information Processing Systems, 27 (2014).

[85] I. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio, *Maxout networks*, in International Conference on Machine Learning, PMLR, 2013, pp. 1319–1327.

[86] G. J. Gordon, *Stable fitted reinforcement learning*, in Advances in Neural Information Processing Systems, 1996, pp. 1052–1058.

[87] J. Grau-Moya, F. Leibfried, and P. Vrancx, *Soft Q-learning with mutual-information regularization*, in International Conference on Learning Representations, 2018.

[88] S. Gu, T. Lillicrap, I. Sutskever, and S. Levine, *Continuous deep Q-learning with model-based acceleration*, in International Conference on Machine Learning, PMLR, 2016, pp. 2829–2838.

[89] O. Guéant, C.-A. Lehalle, and J. Fernandez-Tapia, *Optimal portfolio liquidation with limit orders*, SIAM Journal on Financial Mathematics, 3 (2012), pp. 740–764.

[90] ——, *Dealing with the inventory risk: A solution to the market making problem*, Mathematics and Financial Economics, 7 (2013), pp. 477–507.

[91] O. Guéant and I. Manziuk, *Deep reinforcement learning for market making in corporate bonds: beating the curse of dimensionality*, Applied Mathematical Finance, 26 (2019), pp. 387–452.

[92] F. Guilbaud and H. Pham, *Optimal high-frequency trading with limit and market orders*, Quantitative Finance, 13 (2013), pp. 79–94.

[93] X. Guo, A. Hu, and Y. Zhang, *Reinforcement learning for linear-convex models with jumps via stability analysis of feedback controls*, arXiv preprint arXiv:2104.09311, (2021).

[94] T. Haarnoja, H. Tang, P. Abbeel, and S. Levine, *Reinforcement learning with deep energy-based policies*, in International Conference on Machine Learning, PMLR, 2017, pp. 1352–1361.

[95] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, *Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor*, in International Conference on Machine Learning, PMLR, 2018, pp. 1861–1870.

[96] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, et al., *Soft actor-critic algorithms and applications*, arXiv preprint arXiv:1812.05905, (2018).

[97] N. H. Hakansson, *Multi-period mean-variance analysis: Toward a general theory of portfolio choice*, The Journal of Finance, 26 (1971), pp. 857–884.

[98] I. Halperin, *The QLBS Q-learner goes NuQlear: Fitted Q iteration, inverse RL, and option portfolios*, Quantitative Finance, 19 (2019), pp. 1543–1553.

[99] ——, *QLBS: Q-learner in the Black-Scholes (-Merton) worlds*, The Journal of Derivatives, 28 (2020), pp. 99–122.

[100] B. Hambly, R. Xu, and H. Yang, *Policy gradient methods for the noisy linear quadratic regulator over a finite horizon*, SIAM Journal on Control and Optimization, 59 (2021), pp. 3359–3391.

[101] H. Hasselt, *Double Q-learning*, Advances in Neural Information Processing Systems, 23 (2010), pp. 2613–2621.

[102] D. Hendricks and D. Wilcox, *A reinforcement learning extension to the Almgren-Chriss framework for optimal trade execution*, in 2014 IEEE Conference on Computational Intelligence for Financial Engineering & Economics (CIFEr), IEEE, 2014, pp. 457–464.

[103] P. Henrotte, *Transaction costs and duplication strategies*, Graduate School of Business, Stanford University, (1993).

[104] S. Heston, *A closed-form solution for options with stochastic volatility with applications to bond and currency options*, Review of Financial Studies, 6 (1993), pp. 327–343.

[105] N. E. Huang, M.-L. Wu, W. Qu, S. R. Long, and S. S. Shen, *Applications of Hilbert–Huang transform to non-stationary financial time series analysis*, Applied Stochastic Models in Business and Industry, 19 (2003), pp. 245–268.

[106] O. Ian, V. R. Benjamin, and R. Daniel, *(More) Efficient reinforcement learning via posterior sampling*, in Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2, NIPS'13, 2013, pp. 3003–3011.

[107] S. Jaimungal, S. M. Pesenti, Y. S. Wang, and H. Tatsat, *Robust risk-aware reinforcement learning*, Available at SSRN 3910498, (2021).

[108] G. Jeong and H. Y. Kim, *Improving financial trading decisions using deep Q-learning: Predicting the number of shares, action strategies, and transfer learning*, Expert Systems with Applications, 117 (2019), pp. 125–138.

[109] J. Jiang, B. T. Kelly, and D. Xiu, *(Re-) Imag (in) ing price trends*, Chicago Booth Research Paper, (2020).

[110] Z. Jiang, D. Xu, and J. Liang, *A deep reinforcement learning framework for the financial portfolio management problem*, arXiv preprint arXiv:1706.10059, (2017).

[111] C. Jin, Z. Allen-Zhu, S. Bubeck, and M. I. Jordan, *Is Q-learning provably efficient?*, in Advances in Neural Information Processing Systems, vol. 31, 2018.

[112] C. Jin, Z. Yang, Z. Wang, and M. I. Jordan, *Provably efficient reinforcement learning with linear function approximation*, in Conference on Learning Theory, PMLR, 2020, pp. 2137–2143.

[113] S. M. Kakade, *A natural policy gradient*, Advances in Neural Information Processing Systems, 14 (2001).

[114] M. Karpe, J. Fang, Z. Ma, and C. Wang, *Multi-agent reinforcement learning in a realistic limit order book market simulation*, in Proceedings of the First ACM International Conference on AI in Finance, ICAIF '20, 2020.

[115] T. T. Ke, Z.-J. M. Shen, and J. M. Villas-Boas, *Search for information on multiple products*, Management Science, 62 (2016), pp. 3576–3603.

[116] M. Kearns and S. Singh, *Near-optimal reinforcement learning in polynomial time*, Machine Learning, 49 (2002), pp. 209–232.

[117] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, in Proceedings of the 3rd International Conference on Learning Representations (ICLR), 2015.

[118] S. Klöppel and M. Schweizer, *Dynamic indifference valuation via convex risk measures*, Mathematical Finance, 17 (2007), pp. 599–627.

[119] S. Koenig and R. G. Simmons, *Complexity analysis of real-time reinforcement learning*, in AAAI, 1993, pp. 99–107.

[120] P. N. Kolm and G. Ritter, *Dynamic replication and hedging: A reinforcement learning approach*, The Journal of Financial Data Science, 1 (2019), pp. 159–171.

[121] ——, *Modern perspectives on reinforcement learning in finance*, Modern Perspectives on Reinforcement Learning in Finance (September 6, 2019). The Journal of Machine Learning in Finance, 1 (2020).

[122] V. Konda, *Actor-critic algorithms*, PhD thesis, MIT, 2002.

[123] V. R. Konda and J. N. Tsitsiklis, *Actor-critic algorithms*, in Advances in Neural Information Processing Systems, 2000, pp. 1008–1014.

[124] C. Kühn and M. Stroh, *Optimal portfolios of a small investor in a limit order market: A shadow price approach*, Mathematics and Financial Economics, 3 (2010), pp. 45–72.

[125] H. Kumar, A. Koppel, and A. Ribeiro, *On the sample complexity of actor-critic method for reinforcement learning with function approximation*, arXiv preprint arXiv:1910.08412, (2019).

[126] M. G. Lagoudakis and R. Parr, *Least-squares policy iteration*, The Journal of Machine Learning Research, 4 (2003), pp. 1107–1149.

[127] C. Lakshminarayanan and C. Szepesvari, *Linear stochastic approximation: How far does constant step-size and iterate averaging go?*, in International Conference on Artificial Intelligence and Statistics, PMLR, 2018, pp. 1347–1355.

[128] T. Lattimore and M. Hutter, *PAC bounds for discounted MDPs*, in International Conference on Algorithmic Learning Theory, Springer, 2012, pp. 320–334.

[129] T. Lattimore, C. Szepesvari, and G. Weisz, *Learning with good feature representations in bandits and in RL with a generative model*, in International Conference on Machine Learning, PMLR, 2020, pp. 5662–5670.

[130] L. Leal, M. Laurière, and C.-A. Lehalle, *Learning a functional control for high-frequency finance*, arXiv preprint arXiv:2006.09611, (2020).

[131] Y. LeCun and Y. Bengio, *Convolutional networks for images, speech, and time series*, The Handbook of Brain Theory and Neural Networks, 3361 (1995), p. 1995.

[132] C.-A. Lehalle and S. Laruelle, *Market microstructure in practice*, World Scientific, 2018.

[133] H. E. Leland, *Option pricing and replication with transactions costs*, The Journal of Finance, 40 (1985), pp. 1283–1301.

[134] D. Li and W.-L. Ng, *Optimal dynamic portfolio selection: Multiperiod mean-variance formulation*, Mathematical Finance, 10 (2000), pp. 387–406.

[135] L. Li, *A unifying framework for computational reinforcement learning theory*, Rutgers The State University of New Jersey-New Brunswick, 2009.

[136] Y. Li, C. Szepesvari, and D. Schuurmans, *Learning exercise policies for American options*, in Artificial Intelligence and Statistics, PMLR, 2009, pp. 352–359.

[137] Z. Liang, H. Chen, J. Zhu, K. Jiang, and Y. Li, *Adversarial deep reinforcement learning in portfolio management*, arXiv preprint arXiv:1808.09940, (2018).

[138] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, *Continuous control with deep reinforcement learning*, in 4th International Conference on Learning Representations (ICLR), 2016.

[139] L.-J. Lin, *Self-improving reactive agents based on reinforcement learning, planning and teaching*, Machine Learning, 8 (1992), pp. 293–321.

[140] S. Lin and P. A. Beling, *An end-to-end optimal trade execution framework based on proximal policy optimization*, in IJCAI, 2020, pp. 4548–4554.

[141] B. Liu, Q. Cai, Z. Yang, and Z. Wang, *Neural trust region/proximal policy optimization attains globally optimal policy*, in Advances in Neural Information Processing Systems, vol. 32, 2019.

[142] Y. Liu, K. Zhang, T. Basar, and W. Yin, *An improved analysis of (variance-reduced) policy gradient and natural policy gradient methods*, in NeurIPS, 2020.

[143] F. A. Longstaff and E. S. Schwartz, *Valuing American options by simulation: A simple least-squares approach*, The Review of Financial Studies, 14 (2001), pp. 113–147.

[144] H. M. Markowitz, *Portfolio selection*, Journal of Finance, 7 (1952), pp. 77–91.

[145] A. massoud Farahmand, M. Ghavamzadeh, C. Szepesvàri, and S. Mannor, *Regularized fitted Q-iteration for planning in continuous-space Markovian decision problems*, in 2009 American Control Conference, IEEE, 2009, pp. 725–730.

[146] J. Mei, C. Xiao, C. Szepesvari, and D. Schuurmans, *On the global convergence rates of softmax policy gradient methods*, in International Conference on Machine Learning, PMLR, 2020, pp. 6820–6829.

[147] F. S. Melo and M. I. Ribeiro, *Q-learning with linear function approximation*, in International Conference on Computational Learning Theory, Springer, 2007, pp. 308–322.

[148] T. L. Meng and M. Khushi, *Reinforcement learning in financial markets*, Data, 4 (2019), p. 110.

[149] R. C. Merton, *Theory of rational option pricing*, The Bell Journal of Economics and Management Science, (1973), pp. 141–183.

[150] R. C. Merton and P. A. Samuelson, *Fallacy of the log-normal approximation to optimal portfolio decision-making over many periods*, Journal of Financial Economics, 1 (1974), pp. 67–94.

[151] O. Mihatsch and R. Neuneier, *Risk-sensitive reinforcement learning*, Machine Learning, 49 (2002), pp. 267–290.

[152] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, *Asynchronous methods for deep reinforcement learning*, in International Conference on Machine Learning, PMLR, 2016, pp. 1928–1937.

[153] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, *Playing atari with deep reinforcement learning*, arXiv preprint arXiv:1312.5602, (2013).

[154] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al., *Human-level control through deep reinforcement learning*, Nature, 518 (2015), pp. 529–533.

[155] J. Moody, L. Wu, Y. Liao, and M. Saffell, *Performance functions and reinforcement learning for trading systems and portfolios*, Journal of Forecasting, 17 (1998), pp. 441–470.

[156] A. Mosavi, Y. Faghan, P. Ghamisi, P. Duan, S. F. Ardabili, E. Salwana, and S. S. Band, *Comprehensive review of deep reinforcement learning methods and applications in economics*, Mathematics, 8 (2020), p. 1640.

[157] J. Mossin, *Optimal multiperiod portfolio policies*, The Journal of Business, 41 (1968), pp. 215–229.

[158] Y. Nevmyvaka, Y. Feng, and M. Kearns, *Reinforcement learning for optimized trade execution*, in Proceedings of the 23rd International Conference on Machine Learning, 2006, pp. 673–680.

[159] B. Ning, F. H. T. Ling, and S. Jaimungal, *Double deep Q-learning for optimal execution*, arXiv preprint arXiv:1812.06600, (2018).

[160] A. A. Obizhaeva and J. Wang, *Optimal trading strategy and supply/demand dynamics*, Journal of Financial Markets, 16 (2013), pp. 1–32.

[161] S. J. Pan and Q. Yang, *A survey on transfer learning*, IEEE Transactions on Knowledge and Data Engineering, 22 (2009), pp. 1345–1359.

[162] M. Papini, D. Binaghi, G. Canonaco, M. Pirotta, and M. Restelli, *Stochastic variance-reduced policy gradient*, in International Conference on Machine Learning, PMLR, 2018, pp. 4026–4035.

[163] H. Park, M. K. Sim, and D. G. Choi, *An intelligent financial portfolio trading strategy using deep Q-learning*, Expert Systems with Applications, 158 (2020), p. 113573.

[164] Y. Patel, *Optimizing market making using multi-agent reinforcement learning*, arXiv preprint arXiv:1812.10252, (2018).

[165] J. L. Pedersen and G. Peskir, *Optimal mean-variance portfolio selection*, Mathematics and Financial Economics, 11 (2017), pp. 137–160.

[166] P. C. Pendharkar and P. Cusatis, *Trading financial indices with reinforcement learning agents*, Expert Systems with Applications, 103 (2018), pp. 1–13.

[167] A. F. Perold, *The implementation shortfall: Paper versus reality*, Journal of Portfolio Management, 14 (1988), p. 4.

[168] J. Peters and S. Schaal, *Natural actor-critic*, Neurocomputing, 71 (2008), pp. 1180–1190.

[169] L. Pomatto, P. Strack, and O. Tamuz, *The cost of information*, arXiv preprint arXiv:1812.04211, (2018).

[170] T. Preis, *Price-time priority and pro rata matching in an order book model of financial markets*, in Econophysics of Order-driven Markets, Springer, 2011, pp. 65–72.

[171] M. L. Puterman, *Markov decision processes: discrete stochastic dynamic programming*, John Wiley & Sons, 2014.

[172] Z. Ren and Z. Zhou, *Dynamic batch learning in high-dimensional sparse linear contextual bandits*, arXiv preprint arXiv:2008.11918, (2020).

[173] M. Riedmiller, *Neural fitted Q iteration–first experiences with a data efficient neural reinforcement learning method*, in European Conference on Machine Learning, Springer, 2005, pp. 317–328.

[174] H. Robbins, *Some aspects of the sequential design of experiments*, Bulletin of the American Mathematical Society, 58 (1952), pp. 527–535.

[175] P. A. Samuelson, *Lifetime portfolio selection by dynamic stochastic programming*, Stochastic Optimization Models in Finance, (1975), pp. 517–524.

[176] Y. Sato, *Model-free reinforcement learning for financial portfolios: A brief survey*, arXiv preprint arXiv:1904.04973, (2019).

[177] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, *Trust region policy optimization*, in International Conference on Machine Learning, PMLR, 2015, pp. 1889–1897.

[178] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, *Proximal policy optimization algorithms*, arXiv preprint arXiv:1707.06347, (2017).

[179] L. Shani, Y. Efroni, and S. Mannor, *Adaptive trust region policy optimization: Global convergence and faster rates for regularized MDPs*, in Proceedings of the AAAI Conference on Artificial Intelligence, vol. 34, 2020, pp. 5668–5675.

[180] W. F. Sharpe, *Mutual fund performance*, The Journal of Business, 39 (1966), pp. 119–138.

[181] Y. Shen, R. Huang, C. Yan, and K. Obermayer, *Risk-averse reinforcement learning for algorithmic trading*, in 2014 IEEE Conference on Computational Intelligence for Financial Engineering & Economics (CIFEr), IEEE, 2014, pp. 391–398.

[182] Y. Shen, M. J. Tobia, T. Sommer, and K. Obermayer, *Risk-sensitive reinforcement learning*, Neural Computation, 26 (2014), pp. 1298–1328.

[183] Z. Shen, A. Ribeiro, H. Hassani, H. Qian, and C. Mi, *Hessian aided policy gradient*, in International Conference on Machine Learning, PMLR, 2019, pp. 5729–5738.

[184] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, et al., *Mastering chess and shogi by self-play with a general reinforcement learning algorithm*, arXiv preprint arXiv:1712.01815, (2017).

[185] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, *Deterministic policy gradient algorithms*, in International Conference on Machine Learning, PMLR, 2014, pp. 387–395.

[186] D. Simchi-Levi and Y. Xu, *Bypassing the monster: A faster and simpler optimal algorithm for contextual bandits under realizability*, Available at SSRN 3562765, (2020).

[187] F. A. Sortino and L. N. Price, *Performance measurement in a downside risk framework*, The Journal of Investing, 3 (1994), pp. 59–64.

[188] T. Spooner, J. Fearnley, R. Savani, and A. Koukorinis, *Market making via reinforcement learning*, in International Foundation for Autonomous Agents and Multiagent Systems, AAMAS '18, 2018, pp. 434–442.

[189] T. Spooner and R. Savani, *Robust Market Making via Adversarial Reinforcement Learning*, in Proc. of the 29th International Joint Conference on Artificial Intelligence, IJCAI-20, 7 2020, pp. 4590–4596.

[190] M. Steinbach, *Markowitz revisited: Mean-variance models in financial portfolio analysis*, Society for Industrial and Applied Mathematics, 43 (2001), pp. 31–85.

[191] A. L. Strehl, L. Li, and M. L. Littman, *Reinforcement learning in finite MDPs: PAC analysis*, Journal of Machine Learning Research, 10 (2009).

[192] A. L. Strehl and M. L. Littman, *A theoretical analysis of model-based interval estimation*, in Proceedings of the 22nd International Conference on Machine Learning, ICML'05, Association for Computing Machinery, 2005, pp. 856–863.

[193] R. H. Strotz, *Myopia and inconsistency in dynamic utility maximization*, The Review of Economic Studies, 23 (1955), pp. 165–180.

[194] I. Sutskever, J. Martens, and G. E. Hinton, *Generating text with recurrent neural networks*, in ICML, 2011.

[195] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, MIT press, 2018.

[196] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, *Policy gradient methods for reinforcement learning with function approximation*, in Advances in Neural Information Processing Systems, 2000, pp. 1057–1063.

[197] I. Szita and C. Szepesvári, *Model-based reinforcement learning with nearly tight exploration complexity bounds*, in ICML, 2010, pp. 1031–1038.

[198] A. Tamar, Y. Chow, M. Ghavamzadeh, and S. Mannor, *Policy gradient for coherent risk measures*, in Advances in Neural Information Processing Systems, vol. 28, 2015.

[199] W. Tang, P. Y. Zhang, and X. Y. Zhou, *Exploratory HJB equations and their convergence*, arXiv preprint arXiv:2109.10269, (2021).

[200] W. R. Thompson, *On the likelihood that one unknown probability exceeds another in view of the evidence of two samples*, Biometrika, 25 (1933), pp. 285–294.

[201] T. Tieleman and G. Hinton, *Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude*, COURSERA: Neural Networks for Machine Learning, 4 (2012), pp. 26–31.

[202] L. Torrey and J. Shavlik, *Transfer learning*, in Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques, IGI global, 2010, pp. 242–264.

[203] N. Vadori, S. Ganesh, P. Reddy, and M. Veloso, *Risk-sensitive reinforcement learning: A martingale approach to reward uncertainty*, arXiv preprint arXiv:2006.12686, (2020).

[204] H. Van Hasselt, A. Guez, and D. Silver, *Deep reinforcement learning with double Q-learning*, in Proceedings of the AAAI Conference on Artificial Intelligence, vol. 30, 2016.

[205] E. Vigna, *On time consistency for mean-variance portfolio selection*, Collegio Carlo Alberto Notebook, 476 (2016).

[206] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, et al., *Grandmaster level in starcraft ii using multi-agent reinforcement learning*, Nature, 575 (2019), pp. 350–354.

[207] U. Von Luxburg and B. Schölkopf, *Statistical learning theory: Models, concepts, and results*, in Handbook of the History of Logic, vol. 10, Elsevier, 2011, pp. 651–706.

[208] H. Wang, *Large scale continuous-time mean-variance portfolio allocation via reinforcement learning*, Available at SSRN 3428125, (2019).

[209] H. Wang and S. Yu, *Robo-advising: Enhancing investment with inverse optimization and deep reinforcement learning*, arXiv preprint arXiv:2105.09264, (2021).

[210] H. Wang, T. Zariphopoulou, and X. Zhou, *Exploration versus exploitation in reinforcement learning: A stochastic control approach*, J. Mach. Learn. Res., 21 (2020), pp. 1–34.

[211] H. Wang and X. Y. Zhou, *Continuous-time mean–variance portfolio selection: A reinforcement learning framework*, Mathematical Finance, 30 (2020), pp. 1273–1308.

[212] L. Wang, Q. Cai, Z. Yang, and Z. Wang, *Neural policy gradient methods: Global optimality and rates of convergence*, in International Conference on Learning Representations, 2020.

[213] Y. Wang, K. Dong, X. Chen, and L. Wang, *Q-learning with UCB exploration is sample efficient for infinite-horizon MDP*, in International Conference on Learning Representations, 2020.

[214] Z. Wang, V. Bapst, N. Heess, V. Mnih, R. Munos, K. Kavukcuoglu, and N. Freitas, *Sample efficient actor-critic with experience replay*, in International Conference on Learning Representations (ICLR), 2017, pp. 1–13.

[215] C. J. Watkins and P. Dayan, *Q-learning*, Machine learning, 8 (1992), pp. 279–292.

[216] C.-Y. Wei, M. J. Jahromi, H. Luo, H. Sharma, and R. Jain, *Model-free reinforcement learning in infinite-horizon average-reward Markov decision processes*, in International Conference on Machine Learning, PMLR, 2020, pp. 10170–10180.

[217] H. Wei, Y. Wang, L. Mangu, and K. Decker, *Model-based reinforcement learning for predictions and control for limit order books*, arXiv preprint arXiv:1910.03743, (2019).

[218] M. Wiese, R. Knobloch, R. Korn, and P. Kretschmer, *Quant GANs: Deep generation of financial time series*, Quantitative Finance, 20 (2020), pp. 1419–1440.

[219] R. J. Williams, *Simple statistical gradient-following algorithms for connectionist reinforcement learning*, Machine Learning, 8 (1992), pp. 229–256.

[220] Y. Wu, R. Shariff, T. Lattimore, and C. Szepesvári, *Conservative bandits*, in International Conference on Machine Learning, PMLR, 2016, pp. 1254–1262.

[221] H. Xiao, Z. Zhou, T. Ren, Y. Bai, and W. Liu, *Time-consistent strategies for multi-period mean-variance portfolio optimization with the serially correlated returns*, Communications in Statistics-Theory and Methods, 49 (2020), pp. 2831–2868.

[222] H. Xiong, T. Xu, Y. Liang, and W. Zhang, *Non-asymptotic convergence of adam-type reinforcement learning algorithms under Markovian sampling*, Proceedings of the AAAI Conference on Artificial Intelligence, 35 (2021), pp. 10460–10468.

[223] H. Xiong, L. Zhao, Y. Liang, and W. Zhang, *Finite-time analysis for double Q-learning*, Advances in Neural Information Processing Systems, 33 (2020).

[224] Z. Xiong, X.-Y. Liu, S. Zhong, H. Yang, and A. Walid, *Practical deep reinforcement learning approach for stock trading*, arXiv preprint arXiv:1811.07522, (2018).

[225] P. Xu, F. Gao, and Q. Gu, *An improved convergence analysis of stochastic variance-reduced policy gradient*, in Uncertainty in Artificial Intelligence, PMLR, 2020, pp. 541–551.

[226] P. Xu, F. Gao, and Q. Gu, *Sample efficient policy gradient methods with recursive variance reduction*, in International Conference on Learning Representations, 2020.

[227] P. Xu and Q. Gu, *A finite-time analysis of Q-learning with neural network function approximation*, in International Conference on Machine Learning, PMLR, 2020, pp. 10555–10565.

[228] T. Xu, Z. Wang, and Y. Liang, *Improving sample complexity bounds for (natural) actor-critic algorithms*, in Advances in Neural Information Processing Systems, vol. 33, 2020, pp. 4358–4369.

[229] ——, *Non-asymptotic convergence analysis of two time-scale (natural) actor-critic algorithms*, arXiv preprint arXiv:2005.03557, (2020).

[230] T. Xu, Z. Yang, Z. Wang, and Y. Liang, *Doubly robust off-policy actor-critic: Convergence and optimality*, arXiv preprint arXiv:2102.11866, (2021).

[231] H. Yang, X.-Y. Liu, and Q. Wu, *A practical machine learning approach for dynamic stock recommendation*, in 2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE), IEEE, 2018, pp. 1693–1697.

[232] L. Yang and M. Wang, *Sample-optimal parametric Q-learning using linearly additive features*, in International Conference on Machine Learning, PMLR, 2019, pp. 6995–7004.

[233] L. Yang and M. Wang, *Reinforcement learning in feature space: Matrix bandit, kernels, and regret bound*, in International Conference on Machine Learning, PMLR, 2020, pp. 10746–10756.

[234] R. Yang, X. Sun, and K. Narasimhan, *A generalized algorithm for multi-objective reinforcement learning and policy adaptation*, in Advances in Neural Information Processing Systems, vol. 32, 2019.

[235] Z. Ye, W. Deng, S. Zhou, Y. Xu, and J. Guan, *Optimal trade execution based on deep deterministic policy gradient*, in Database Systems for Advanced Applications, Springer International Publishing, 2020, pp. 638–654.

[236] P. Yu, J. S. Lee, I. Kulyatin, Z. Shi, and S. Dasgupta, *Model-based deep reinforcement learning for dynamic portfolio optimization*, arXiv preprint arXiv:1901.08740, (2019).

[237] S. Yu, H. Wang, and C. Dong, *Learning risk preferences from investment portfolios using inverse optimization*, arXiv preprint arXiv:2010.01687, (2020).

[238] G. Zhang and Y. Chen, *Reinforcement learning for optimal market making with the presence of rebate*, Available at SSRN 3646753, (2020).

[239] J. Zhang, J. Kim, B. O'Donoghue, and S. Boyd, *Sample efficient reinforcement learning with REINFORCE*, in Proceedings of the AAAI Conference on Artificial Intelligence, vol. 35, 2021, pp. 10887–10895.

[240] K. Zhang, A. Koppel, H. Zhu, and T. Basar, *Global convergence of policy gradient methods to (almost) locally optimal policies*, SIAM Journal on Control and Optimization, 58 (2020), pp. 3586–3612.

[241] Z. Zhang, S. Zohren, and S. Roberts, *Deep reinforcement learning for trading*, The Journal of Financial Data Science, 2 (2020), pp. 25–40.

[242] L. Zheng and L. Ratliff, *Constrained upper confidence reinforcement learning*, in Learning for Dynamics and Control, PMLR, 2020, pp. 620–629.

[243] D. Zhou, J. Chen, and Q. Gu, *Provable multi-objective reinforcement learning with generative models*, arXiv preprint arXiv:2011.10134, (2020).

[244] X. Y. Zhou and D. Li, *Continuous-time mean-variance portfolio selection: A stochastic LQ framework*, Applied Mathematics and Optimization, 42 (2000), pp. 19–33.

[245] E. Zivot, *Introduction to Computational Finance and Financial Econometrics*, Chapman & Hall Crc, 2017.

[246] S. Zou, T. Xu, and Y. Liang, *Finite-sample analysis for SARSA with linear function approximation*, Advances in Neural Information Processing Systems, 32 (2019), pp. 8668–8678.