Ham_Tom_01340364.pdf

by Tom Ham

Submission date: 06-Sep-2022 01:13PM (UTC+0100)
Submission ID: 185731875
File name: Ham_Tom_01340364.pdf (1.07M)
Word count: 16900
Character count: 79704

Imperial College London

Imperial College London

Department of Mathematics

Actor-Critic Reinforcement Learning Methods for Electronic Market Making

Author: Tom Ham (CID: 01340364)

A thesis submitted for the degree of

MSc in Mathematics and Finance, 2021-2022

Declaration

The work contained in this thesis is my own work unless otherwise stated.

Acknowledgements

I would like to thank Paul Bilokon for the help and support throughout writing this thesis.

Abstract

Providing liquidity to a market is both necessary for efficient price discovery, and can also be highly profitable for market making firms. Market making is becoming increasingly automated, and controlled by algorithms, and in this work, we aim to see if the reinforcement learning technique of actor-critic approaches can be applied to the problem of optimal market making. Reinforcement learning techniques have previously been applied to the problem of optimal market making, however this specific approach of actor-critic methods is less studied in the literature. In this thesis we will show that one particular actor-critic method is able to outperform a well known and well studied market making model, known as the Avellaneda-Stoikov model, on real market data. This is a result which has not been previously demonstrated.

In the following work, we introduce the theoretical framework of actor-critic methods, and apply them to both a simulated market environment, and also apply them to real market data by looking at historical Tesla order book data.

Contents

List of Figures

List of Tables

List of Algorithms

Introduction

In recent decades, technological advances have changed the way that technology is used in finance, and financial markets. It is no secret that advances in technology have been rapid, and adoption of new technologies in finance has been equally rapid. The way in which parties interact with financial markets has been fundamentally changed by the rise of *electronic trading*. As discussed by Allen *et al.* [4], the main feature of an electronic trading system is the automation of trade execution. This allows for continuous trading throughout a trading period, and removes any geographical restrictions on parties willing to trade on exchanges located far from them. Traditional trading methods (such as over the phone dealing, or floor trading) do not bestow both these advantages on traders. The adoption of electronic trading is clear, as in 2018, it is estimated that between 60-75% of all trades on US equities markets were performed by algorithms [5].

Since the financial crisis of 2007-2008, increased regulation has contributed to a surge in the collection and storage of financial data. The vast quantities of data that are now collected pertaining to finance, and financial markets means it is a prime area to explore machine learning techniques. Bilokon *et al.* [6] discuss the current widely applied uses of machine learning in finance, including fraud detection, asset pricing, and robo-advisors. One reason for the success of machine learning techniques so far when used in a financial context is that they are able to model high dimensional and incredibly complex systems. Machine learning approaches can learn behaviour that cannot easily be replicated by traditional model-based methods.

The particular branch of machine learning that will be explored in this thesis is *reinforcement learning*. This is a model free method for learning some optimal way of acting in a given situation. Reinforcement learning methods learn this by interacting with an environment, and learning through observation. The aim of this thesis will be to apply a specific kind of reinforcement learning method (known as actor-critic methods) to the problem of *market making*. A market making strategy is one in which a trader offers to simultaneously buy and sell an asset at different prices, and aims to receive two way trades so that they can profit on the difference between their buy price and their sell price.

The applications of reinforcement learning to market making have only recently started to be explored. Gašperov *et al.* [7] provide an extensive literature review on the different reinforcement learning approaches to optimal market making, and finds that research in reinforcement learning applications to market making only really started in 2018. Both [8] and [9] produce similar results, that is that reinforcement learning methods can perform as well as the Avellaneda-Stoikov market making model (a mathematical stochastic control model which will be explored in this thesis), specifically deep Q-learning techniques. However there has been significantly less research into actor-critic methods applied to market making, with Bakshaev showing that one actor-critic algorithm, *Soft Actor-Critic* can perform well on simulated order flow [10]. In Chapter 2 of this thesis, we reproduce the results found in [8] and [9], i.e. that reinforcement learning methods can perform as well as the Avellaneda-Stoikov model, however we instead use an actor-critic reinforcement learning algorithm, *Advantage Actor-Critic*. The reason for first demonstrating that actor-critic methods work on simulated environments is that we are able to use this as a foundation for showing that the results can hold on real market data.

In Chapter 3, we will use real order book data on Telsa (TSLA) to see if actor-critic algorithms can produce profitable market making strategies on real data. We compare the results of three different actor-critic algorithms, and also use an Avellaneda-Stoikov strategy on the market data to see how the reinforcement learning methods compare to the well documented and tested mathematical model. In the results, we observe that one algorithm in particular, *Deep Deterministic Policy Gradient* (DDPG) is able to outperform the Avellaneda-Stoikov strategy in terms of both mean and median wealth. We show that this actor-critic method is able to learn how to interact with the environment to achieve profitable market making, a novel result in this particular area of research. All code used to produce the results shown in this thesis can be found here: https://github.com/tomham000/actor-critic-market-making.

Chapter 1

Theory

In this chapter, we provide a background on the theory which will be relevant for the empirical results shown in Chapter 2 and Chapter 3. We start by considering the general reinforcement learning framework, which is crucial for the majority of the results in the thesis. We then consider stochastic control problems, which give us a foundation on the theory behind optimal market making. The final section will discuss electronic market making and the limit order book.

1.1 Fundamentals of Reinforcement Learning

The theory in this section is mostly adapted from Sutton and Barto 2018 [1, Chapters 3-4]. Reinforcement learning is the computational approach to learning through an agent's interaction with its environment. The general idea of a reinforcement learning problem is for an agent to learn what it should do in a given situation, i.e. learn some mapping from *states* to *actions*, in order to maximise some kind of numerical reward. Moreover, it may be the case that the agent's actions have some effect on its environment, and as with many machine learning problems, the agent is not told which actions to take, rather it must explore its possible actions and work out for itself which ones maximise the long term reward. Note this reward maximisation is not just the agent's immediate reward, but also all future subsequent rewards. Apart from an agent and an environment, a reinforcement learning system in general has three additional components:

- A policy. This is some (possibly stochastic) mapping from states to actions. In essence, this is how the agent chooses to act in a given state.
- A reward signal. This defines the goals of the agent. After each time step, the environment will send the agent a single number, known as a reward, and the agent is trying to maximise the total long-term reward it receives.
- A value function. In essence, the *value* of a state is the total amount of reward an agent expects to receive over the whole future if it starts from that state. The value function is the mapping from states to their values.

In this section, we will begin by exploring the general reinforcement learning problem formulation, the *Markov decision process* (MDP), and the fundamental classes of methods for solving MDPs.

1.1.1 Markov Decision Processes

One of the general formulations of reinforcement learning problems is the *Markov Decision Process* (MDP). MDPs model how an agent interacts with its (possibly stochastic) environment in discrete time steps

In order to explore these processes, and how they reinforcement learning has been built up using these processes, we must first define the Markov property.

Definition 1.1.1 (Markov Property)**.** Let $\mathcal{S}$ be a set of states, and let $\{S_t\}_{t\in\mathbb{N}}$ be a sequence of random states such that $S_t \in \mathcal{S}$ for all $t \geq 0$. Then a state $s \in \mathcal{S}$ has the *Markov Property* if for all $s' \in \mathcal{S}$, and rewards $r \in \mathcal{R}$,

$$\mathbb{P}(R_{t+1} = r, S_{t+1} = s' \mid S_1, ..., S_{t-1}, S_t = s) = \mathbb{P}(R_{t+1} = r, S_{t+1} = s' \mid S_t = s).$$

If a state s with the Markov property, then all future information is independent of the past given present information, and once the state is known, any history of the process up until it hits s is irrelevant.

Definition 1.1.2 (Markov Decision Process)**.** A *Markov Decision Process* (MDP) is the tuple $(\mathcal{S}, \mathcal{A}, p, \gamma)$, where

- $\mathcal{S}$ is the set of possible states.
- $\mathcal{A}$ is the set of possible actions.
- $p(r, s \mid s, a)$ is the joint probability of a reward r and next state $s' \in \mathcal{S}$, given $s \in \mathcal{S}$ and $a \in \mathcal{A}$. p is referred to as the *Markov transition density*.
- $\gamma \in [0, 1]$ is a discount factor.

all states $s \in \mathcal{S}$ are assumed to have the Markov property.

Discounted Returns

The aim of our agent is to maximise some kind of expected return at any given time. We wish to design an agent that is able to show a preference toward short term reward, as well as long term reward. This is done by defining our *discounted return*.

Definition 1.1.3 (Returns)**.** The infinite horizon discounted return at time t is defined as

$$G_t := \sum_{k=t+1}^{\infty} \gamma^{k-t-1} R_k.$$

If instead, the task is episodic with some (possibly random) terminal time T, then the discounted return at time t is defined as

$$G_t := \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k.$$

Here, $\gamma \in [0, 1]$ is known as the *discount factor*, and is a measure of how much the agent values long term rewards relative to short term rewards. If γ is close to 0, then the agent is short sighted; it will value short term rewards higher than long term rewards, and vice versa if γ is close to 1. Notice that if $T = \infty$, then $\gamma \neq 1$, otherwise the sum may not converge. One key property of returns is that

$$G_t = R_{t+1} + \gamma G_{t+1}.$$

This iterative definition proves helpful in calculating quantities of interest, and will be explored in the next section.

1.1.2 Policies and State-Value Functions

Most reinforcement learning algorithms involve estimating some *value function*. These functions of states (or state-action pairs) are measurements of how good it is for the agent to be in a particular state, or how good an action is when it is performed in a given state. The way in which we quantify "how good" is by looking at the expected future returns. It is important to note however that the future expected returns strongly depend on how the agent will act in the future, and this motivates the idea of a *policy*.

Definition 1.1.4 (Policy)**.** A *policy* π is a mapping from states $s \in \mathcal{S}$ to probabilities of selecting actions $a \in \mathcal{A}$. Specifically, at time t,

$$\pi(a \mid s) := \mathbb{P}(A_t = a \mid S_t = s).$$

With policies defined, we can now consider the value of a state, and state-action pairs for an agent who is following some policy π.

Definition 1.1.5 (State-value function)**.** Suppose an agent is following a policy π. Then the *state-value function* $v_\pi : \mathcal{S} \to \mathbb{R}$ is given by

$$v_\pi(s) := \mathbb{E}[G_t \mid S_t = t]$$

Definition 1.1.6 (Action-value function)**.** Suppose an agent is following a policy π. Then the *action-value function* $q_\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is given by

$$q_\pi(s, a) := \mathbb{E}[G_t \mid S_t = t, A_t = a]$$

For small state/action spaces, one may be able to keep track of state/action values tabularly, however for more complex problems, it is often convenient to parameterise these functions, e.g. with a neural network. These methods will be explored in detail later in this thesis. We now derive the *Bellman equation* for v_π. We have that for every $s \in \mathcal{S}$,

$$\begin{aligned}
& v_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s] \\
\implies & v_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \\
\implies & v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a \mid s) \sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r \mid s, a)\big(r + \gamma \mathbb{E}_\pi[G_{t+1} \mid S_{t+1} = s']\big) \\
\implies & v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a \mid s) \sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r \mid s, a)\big(r + \gamma v_\pi(s')\big]\big). \qquad (1.1.1)
\end{aligned}$$

(1.1.1) is known as the *Bellman equation for* v_π, and shows the relationship between a state's value, and the values of all possible successor states. We can also similarly derive the Bellman equation for q_π, as

$$q_\pi(s, a) = \sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r \mid s, a) + \gamma \sum_{a \in \mathcal{A}} \pi(a' \mid s') q_\pi(s', a'). \qquad (1.1.2)$$

1.1.3 Dynamic Programming

Dynamic programming (DP) in the context of reinforcement learning is the collection of algorithms to compute optimal policies given some exact model of an environment, for example an MDP. In reality, DP algorithms are not particularly useful in actually solving problems, due to both their large computational requirements, and assumption of a perfectly known environment. However, DP does provide an important foundation for more advanced techniques in reinforcement learning.

Policy Evaluation

Policy evaluation is the process of computing the state-value function v_π for some policy π. If the dynamics of the environment are known, then (1.1.1) is a system of $|\mathcal{S}|$ linear equations, which can be solved by numerous methods. For our purposes, we use an iterative solution method, i.e. we find a sequence of value functions $v_0, v_1, ...$, each $v_i : \mathcal{S} \to \mathbb{R}$. The initial v_0 can be arbitrarily chosen, so long as $v_0(s) = 0$ for every terminal state s. Subsequent approximations are obtained by using the Bellman equation, yielding the following update rule.

$$v_{k+1}(s) = \sum_{a \in \mathcal{A}(s)} \pi(a \mid s) \sum_{s' \in \mathcal{S}, r \in \mathcal{R}} p(s', r \mid s, a)\left[r + \gamma v_k(s')\right]. \qquad (1.1.3)$$

$v_k = v_p i$ is a fixed point of this update rule due to the equality in the Bellman equation. The sequence $\{v_k\}_{k \geq 0}$ has been shown to converge to v_π as $k \to \infty$ [11]. Algorithm 1.1 shows the full iterative policy evaluation algorithm.

Algorithm 1.1 Iterative Policy Evaluation

Input: π, the policy to be evaluated.
  Initialise an array $V(s) = 0$, for all $s \in \mathcal{S}$.
  $\Delta \leftarrow 0$
  repeat
    $\Delta \leftarrow 0$
    for each $s \in \mathcal{S}$ **do**
      $v \leftarrow V(s)$
      $V(s) \leftarrow \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a)[r + \gamma V(s')]$
      $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
    end for
  until $\Delta < \theta$ (a small positive number)
Output: $V(s) \approx v_\pi(s)$

Policy Improvement

Finding the value function, as described in 1.1.3 for some given policy is useful as it allows us to find better policies. This process is known as *policy improvement*. Suppose our agent (following policy π) is in some state $s \in \mathcal{S}$. Suppose they select some action $a \in \mathcal{A}$, which may not be part of their current policy π, and thereafter follows its original policy. The value of this behaviour is,

$$q_\pi(s, a) = \sum_{s' \in \mathcal{S}, r \in \mathcal{R}} p(s', r \mid s, a) \left[r + \gamma v_\pi(s') \right].$$

We are interested in knowing whether this new value is greater than or less than the old value, $v_\pi(s)$. If $q(s, a) > v_\pi(s)$, then selecting a when in state s is always better than the current policy, hence that new policy would be better overall than π. This motivates the more general result of the *policy improvement theorem*. Let π, π' be any pair of deterministic policies such that for all $s \in \mathcal{S}$,

$$q_\pi(s, \pi'(s)) \geq v_\pi(s).$$

Then π' must be at least as good as π, i.e. for all $s \in \mathcal{S}$,

$$v_{\pi'}(s) \geq v_\pi(s).$$

So far, we have seen that if we have π and its value function v_π, we can evaluate a change in policy at one specific state s to a particular action a. The extension would be to consider changes at all states and to all actions, by choosing the best action at each state s, according to $q_\pi(s, a)$. i.e, we consider the new *greedy* policy, π' which is given by

$$\begin{aligned} \pi'(s) &= \arg\max_{a \in \mathcal{A}(s)} q_\pi(s, a) \\ &= \arg\max_{a \in \mathcal{A}(s)} \sum_{s' \in \mathcal{S}, r \in \mathcal{R}} p(s', r \mid s, a) \left[r + \gamma v_\pi(s') \right]. \end{aligned} \tag{1.1.4}$$

This greedy policy takes the best short term action, after looking ahead by one step, according to v_π. This process, i.e. making a new policy that improves on the old policy by making it greedy with respect to v_π is called *policy improvement*. Note that if $v_\pi = v_{\pi'}$, then for all $s \in \mathcal{S}$,

$$v_{\pi'}(s) = \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}, r \in \mathcal{R}} p(s', r \mid s, a) \left[r + \gamma v_\pi(s') \right] \tag{1.1.5}$$

$$= \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}, r \in \mathcal{R}} p(s', r \mid s, a) \left[r + \gamma v_{\pi'}(s') \right]. \tag{1.1.6}$$

Notice that (1.1.5) is exactly the Bellman optimality equation, an alternate version of the Bellman equation. Hence overall, $v_{\pi'} = v^*$, so both π and π' must be optimal policies. This means that policy improvement will give us a strictly better policies until the policy is optimal. Note that here we have only considered deterministic policies, but this theory can very easily be extended to stochastic policies, with minimal changes.

Policy Iteration

One natural idea following both policy evaluation, and policy improvement is to improve some policy π using v_π, yielding a new policy π'. We then evaluate π' to compute $v_{\pi'}$, and then improve it to yield π'', and so on, each policy an improvement on the last. This process is known as *policy iteration*, and is displayed graphically below.

$$\pi_0 \xrightarrow{E} v_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} v_{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} ... \xrightarrow{I} \pi_* \xrightarrow{E} v_{\pi_*}$$

1.1.4 Monte Carlo Methods for Reinforcement Learning

One of the key differences between Monte Carlo (MC) methods and dynamic programming approaches are that Monte Carlo methods do not assume complete knowledge of the environment. The only thing that Monte Carlo methods require are experience, i.e. samples of states, actions and rewards from actual or simulated interaction with the environment. This approach means that no prior knowledge of the environment's dynamics are needed, but optimal behaviour can still be attained. These methods require episodic tasks, and when an episode is complete, value estimates and polices are changed. Monte Carlo methods sample and average returns from each state-action pair, and average rewards for each action.

Suppose for some $s \in \mathcal{S}$, we wish to estimate $v_\pi(s)$, the value of s under π. Suppose we have a set of episodes obtained by following π, each episode passing through s. Each occurrence of state s in each episode is called a *visit* to s. It is possible for s to be visited multiple times in a single episode, but we denote the first time s is visited as the *first visit* to s. The two MC methods are known as *first visit MC method*, and *every-visit MC method*. The former estimates $v_\pi(s)$ as the average of total discounted rewards following the first visits to s, whereas the latter uses every visit to s. First-visit MC is more widely studied in literature, however both methods are very similar, only differing slightly in some theoretical properties. Below shows the first visit MC algorithm for estimating v_π. First-visit MC converges to $v_\pi(s)$ as the number of visits to s goes to infinity.

Algorithm 1.2 First-visit MC Evaluation

Input: a policy π to be evaluated
  initialise $V(s)$ arbitrarily for all $s \in \mathcal{S}$
  initialise $Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$
  while $V(s)$ not converged **do**
    Generate an episode following π, giving $S_0, A_0, R_1, S_1, A_1, R_2, ..., S_{T-1}, A_{T-1}, R_T$
    $G \leftarrow 0$
    for each step of episode $t = T-1, T-2, ..., 0$ **do**
      $G \leftarrow \gamma G + R_{t+1}$
      if not S_t appears in $S_0, ..., S_{t-1}$ **then**
        Append G to $Returns(S_t)$
        $V(S_t) \leftarrow$ average$(Returns(S_t))$
      end if
    end for
  end while
Output: $V(s) \approx v_\pi$

1.1.5 Temporal Difference Methods

Temporal difference (TD) learning combines the ideas from MC methods, and DP methods. Similar to MC methods, TD learning uses direct interaction with the environment with no model of the environment's dynamics. We begin as usual by looking at policy evaluation, that is, finding v_π for some given policy π. The main difference between TD and MC methods, are that while MC methods must wait until the end of an episode to update $V(S_t)$, TD methods can update their estimates after a single time step. At time $t+1$, they make an update using the observed reward,

R_{t+1}, and the estimate $V(S_{t+1})$. The simplest TD update rule is as follows,

$$V(S_t) \leftarrow V(S_t) + \alpha\big[R_{t+1} + \gamma V(S_{t+1} - V(S_t)\big]. \tag{1.1.7}$$

This update is immediately made on the transition from S_t to S_{t+1}, when R_{t+1} is received. Note that this TD method is known as *TD(0)*, or one-step TD, and is a special case of the TD(λ) method. For more details on the generalised TD methods, see [1, Chapter 12]. Algorithm 1.3 shows the full one-step TD method algorithm.

Algorithm 1.3 Tabular one-step temporal difference for estimating v_π

Input: A policy π to be evaluated
Input: Step-size parameter $\alpha \in (0, 1]$
  Initialise $V(s)$ for all $s \in \mathcal{S}$ arbitrarily, except $V(\hat{s}) = 0$ for all terminal states $\hat{s}$
  while V(s) not converged **do**
    Initialise S
    while S not terminal **do**
      $A \leftarrow$ action given by π for S
      Take action A, observe R, S'
      $V(S) \leftarrow V(S) + \alpha\big[R + \gamma V(S') - V(S)\big]$
      $S \leftarrow S'$
    end while
  end while
Output: $V(s) \approx v_\pi(s)$

1.2 Policy Gradient Methods

This section on policy gradient methods is adapted from [1, Chapter 13]. So far we have considered only *action-value methods*. They have all learned the values associated with actions, and selected an action based on these action values. In this section, we consider methods that learn a policy which is parameterised by some vector of parameters, $\boldsymbol{\theta} \in \mathbb{R}^{d'}$, known as *policy gradient methods*. These policies can select actions without needing an action value function, however a value function may still be used to learn the parameter $\boldsymbol{\theta}$. We denote $\pi(a \mid s, \boldsymbol{\theta}) = \mathbb{P}(A_t = a \mid S_t = s, \boldsymbol{\theta}_t = \boldsymbol{\theta})$ as the probability an action a is chosen at time t given the environment is in state s, and the parameter vector is $\boldsymbol{\theta}$. If a method also uses some value function $\hat{v}$, then the weight vector for this function is denoted as $\boldsymbol{w} \in \mathbb{R}^d$, i.e. $\hat{v}(s, \boldsymbol{w})$. Policy gradient methods aim to learn the policy parameter $\boldsymbol{\theta}$ based on the gradient of a performance measure $J(\boldsymbol{\theta})$. The methods will aim to maximise this performance measure J, so gradient ascent is used to update the parameters, i.e.

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha\widehat{\nabla J(\boldsymbol{\theta}_t)}. \tag{1.2.1}$$

Any method which uses this general approach is called a policy gradient method, however methods that also approximate both a policy *and* a value function are known as *actor-critic methods*, and will be discussed in Section 1.2.2. Policy gradient methods require $\pi(a \mid s, \boldsymbol{\theta})$ to be differentiable with respect to $\boldsymbol{\theta}$, i.e $\nabla_{\boldsymbol{\theta}}\pi(a \mid s, \boldsymbol{\theta})$ exists and is finite for all $s \in \mathcal{S}$, $a \in \mathcal{A}$, and $\boldsymbol{\theta} \in \mathbb{R}^{d'}$.

Example - Policy Parameterisation for Discrete Action Spaces

When the action space $\mathcal{A}$ is discrete, and not too large, one common parameterisation of $\pi(a \mid s, \boldsymbol{\theta})$ is to create some kind of numerical preference, $h(s, a, \boldsymbol{\theta}) \in \mathbb{R}$ for all state-action pairs. The actions which have the largest preferences, h, are assigned the largest probabilities of being selected. One example of such selection is the exponential softmax, i.e.

$$\pi(a \mid s, \boldsymbol{\theta}) = \frac{e^{h(s,a,\boldsymbol{\theta})}}{\sum_{k \in \mathcal{A}} e^{h(s,k,\boldsymbol{\theta})}}.$$

It is important to note that the parameterisation is arbitrary, i.e. $\boldsymbol{\theta}$ may be a vector of weights and biases used in a neural network; the neural network would then assign the probabilities to state-action pairs. There are many such possibilities for this parameterisation.

1.2.1 REINFORCE Algorithm

In this section, we will discuss the simplest policy gradient algorithm, REINFORCE. For notational simplicity, from now on we will assume that every episode begins in some deterministic state, $s_0 \in \mathcal{S}$, however the theory can be extended with ease to stochastic starting states. We also assume that the discount factor $\gamma = 1$, but again, this is for convenience and does not reduce the generality of results. We let our performance function, J, be defined as

$$J(\boldsymbol{\theta}) := v_{\pi_{\boldsymbol{\theta}}}(s_0),$$

the value of the initial state in the episode. Here, $v_{\pi_{\boldsymbol{\theta}}}$ is the true value function for $\pi_{\boldsymbol{\theta}}$, the policy parameterised by $\boldsymbol{\theta}$. Intuitively, this makes sense as a performance function, as we are trying to maximise the total discounted reward that we receive given that we start in s_0. To develop the REINFORCE algorithm, we must first introduce the *policy gradient theorem* below.

Theorem 1.2.1 (Policy gradient theorem)**.** *If $J(\boldsymbol{\theta}) := v_{\pi_{\boldsymbol{\theta}}}(s_0)$, then*

$$\nabla J(\boldsymbol{\theta}) = \mathbb{E}_\pi \left[\sum_{a \in \mathcal{A}} q_\pi(S_t, a) \nabla_{\boldsymbol{\theta}} \pi(a \mid S_t, \boldsymbol{\theta}) \right]. \tag{1.2.2}$$

Proof. See [1, Chapter 13, Page 325] □

To continue our derivation of the REINFORCE algorithm, we will further manipulate (1.2.2). We have

$$\nabla J(\boldsymbol{\theta}) = \mathbb{E}_\pi \left[\sum_{a \in \mathcal{A}} q_\pi(S_t, a) \nabla_{\boldsymbol{\theta}} \pi(a \mid S_t, \boldsymbol{\theta}) \right]$$

$$\implies \nabla J(\boldsymbol{\theta}) = \mathbb{E}_\pi \left[\sum_{a \in \mathcal{A}} \pi(a \mid S_t, \boldsymbol{\theta}) q_\pi(S_t, a) \frac{\nabla_{\boldsymbol{\theta}} \pi(a \mid S_t, \boldsymbol{\theta})}{\pi(a \mid S_t, \boldsymbol{\theta})} \right]. \tag{1.2.3}$$

Notice that we can replace a by the sample $A_t \sim \pi(a \mid S_t, \boldsymbol{\theta})$, the action taken at time t in (1.2.3) to give

$$\nabla J(\boldsymbol{\theta}) = \mathbb{E}_\pi \left[q_\pi(S_t, A_t) \frac{\nabla_{\boldsymbol{\theta}} \pi(A_t \mid S_t, \boldsymbol{\theta})}{\pi(A_t \mid S_t, \boldsymbol{\theta})} \right]$$

$$\implies \nabla J(\boldsymbol{\theta}) = \mathbb{E}_\pi \left[G_t \frac{\nabla_{\boldsymbol{\theta}} \pi(A_t \mid S_t, \boldsymbol{\theta})}{\pi(A_t \mid S_t, \boldsymbol{\theta})} \right], \tag{1.2.4}$$

where we have used the fact that $\mathbb{E}_\pi[G_t \mid S_t, A_t] = q_\pi(S_t, A_t)$ in (1.2.4). We now use this directly in our policy gradient ascent update rule, (1.2.1) to give the update rule for REINFORCE,

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha G_t \frac{\nabla_{\boldsymbol{\theta}} \pi(A_t \mid S_t, \boldsymbol{\theta})}{\pi(A_t \mid S_t, \boldsymbol{\theta})}. \tag{1.2.5}$$

If we now take a step back and consider what this update rule means intuitively. Each increment is proportional to the total return multiplied by some vector. This vector is the direction (in the $\boldsymbol{\theta}$-space) that most increases the probability to repeat A_t on future visits to S_t. The update increases proportionally to the total return, which is intuitive as it causes the parameter to move more in directions that favour actions which produce the highest return. The update also increases inversely proportionally to the action probability. This makes sense as actions with a high selection probability are at an advantage, as the updates will occur more often in their direction. Before we introduce the pseudocode for REINFORCE, recall that $\nabla \log(x) = \frac{\nabla x}{x}$, so we can rewrite the update rule (1.2.5) as

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha G_t \nabla_{\boldsymbol{\theta}} \log(\pi(A_t \mid S_t, \boldsymbol{\theta})).$$

In the literature, $\nabla_{\boldsymbol{\theta}} \log(\pi(A_t \mid S_t, \boldsymbol{\theta}))$ is known as the *eligibility vector*. Algorithm 1.4 shows the full pseudocode for the REINFORCE algorithm.

Algorithm 1.4 REINFORCE

Input: a differentiable policy parameterisation, $\pi(a \mid s, \boldsymbol{\theta})$
  Initialise policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$, e.g. to $\mathbf{0}$.
  for each episode **do**
    Generate an episode $S_0, A_0, R_1, ..., S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot, \boldsymbol{\theta})$
    for each step of the episode, $t = 0, ..., T-1$ **do**
      $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$
      $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \gamma^t G \nabla_{\boldsymbol{\theta}} \log(\pi(A_t \mid S_t, \boldsymbol{\theta}))$
    end for
  end for

1.2.2 Actor-Critic Methods

In Chapter 3, actor-critic methods will be the focal point of the results that are presented. Actor-critic methods are policy gradient methods driven by two separate structures; the *actor*, which is used to select which actions to take, and the *critic*, which evaluates the actor's action choices using an estimated value function. The critic produces a *TD error*, which is a scalar signal that drives the learning of both the actor and the critic. Figure 1.1 shows a graphical representation of the actor-critic framework.

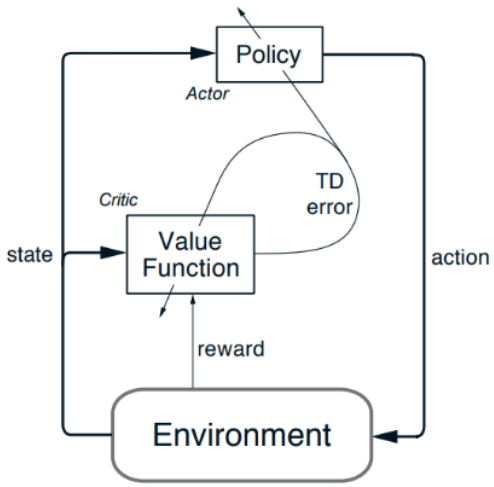

Figure 1.1: A graphical depiction of the actor-critic framework [1, Figure 11.1, page 258]

.

After each action selection, the critic (state-value function) evaluates the new state to see whether the reward and the discounted value of the new state are an overestimate or an underestimate of the value of the old state. The TD error is defined as

$$\delta_t = R_{t+1} + \gamma V_t(s_{S+1}) - V(S_t),$$

where V_t is the critic's value function at time t. If $\delta_t > 0$, then the action performed better than expected, and the actor should increase its tendency to select that action in the future (and vice versa if $\delta_t < 0$). Hence the TD error, δ_t is used to evaluate the action A_t taken while in state S_t.

1.2.3 Advantage Actor-Critic

We now introduce our first actor-critic algorithm, *advantage actor-critic* (A2C). This actor-critic method was first introduced by Mnih *et al.* in 2016 [12]. Firstly, we must define the *advantage* function.

Definition 1.2.2. The *advantage* of the action $a \in \mathcal{A}$ in state $s \in \mathcal{S}$ under policy π is given by

$$A^{\pi}(s,a) := q_{\pi}(s,a) - v_{\pi}(s).$$

Intuitively, this is the difference between the q-value of the action it took, and the probability-weighted average of the other actions it could have taken in state s. To continue, we must first introduce a more general version of the policy gradient theorem.

Theorem 1.2.3 (Policy gradient theorem with baselines)**.** *Let $b : \mathcal{S} \to \mathbb{R}$ be a* baseline*, i.e. some function on the state space. Then*

$$\nabla J(\boldsymbol{\theta}) = \mathbb{E}_{\pi}\left[\sum_{a \in \mathcal{A}} \left(q_{\pi}(S_t,a) - b(S_t)\right) \nabla_{\boldsymbol{\theta}} \pi(a \mid S_t, \boldsymbol{\theta})\right]. \tag{1.2.6}$$

Proof. See [1, Chapter 13, Page 329] □

Notice that our definition of the advantage function can be applied directly to (1.2.6), with v_{π} as the baseline, so that we obtain

$$\nabla J(\boldsymbol{\theta}) = \mathbb{E}_{\pi}\left[\sum_{a \in \mathcal{A}} A^{\pi}(S_t,a) \nabla_{\boldsymbol{\theta}} \pi(a \mid S_t, \boldsymbol{\theta})\right]. \tag{1.2.7}$$

Using identical steps as those in the previous section, it can also be shown that

$$\nabla J(\boldsymbol{\theta}) = \mathbb{E}_{\pi}\left[A^{\pi}(S_t,A_t) \nabla_{\boldsymbol{\theta}} \log(\pi_{\boldsymbol{\theta}}(A_t \mid S_t; \boldsymbol{\theta}))\right] \tag{1.2.8}$$

At this point, we introduce our actor and our critic. The actor is the policy $\pi_{\boldsymbol{\theta}}(\cdot \mid \cdot, \boldsymbol{\theta})$, parameterised by $\boldsymbol{\theta}$, and the critic is the state-value function $v_{\boldsymbol{\varphi}}(\cdot; \boldsymbol{\varphi})$, parameterised by $\boldsymbol{\varphi}$. One way of estimating the advantage function, such that it is equal in expectation to the true advantage function, is to use the *n-step advantage estimate*. This estimate is given by

$$\hat{A}_{\boldsymbol{\theta},\boldsymbol{\varphi}}(s,a) = \sum_{k=0}^{n-1} \gamma^k R_{t+k+1} + \gamma^n v_{\boldsymbol{\varphi}}(S_{t+n+1}) - v_{\boldsymbol{\varphi}}(S_t). \tag{1.2.9}$$

Using this advantage estimate along with (1.2.8), our update rule for $\boldsymbol{\theta}$ becomes

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \eta \nabla_{\boldsymbol{\theta}} \log(\pi_{\boldsymbol{\theta}}(A_t \mid S_t; \boldsymbol{\theta})) \left(\sum_{k=0}^{n-1} \gamma^k R_{t+k+1} + \gamma^n v_{\boldsymbol{\varphi}}(S_{t+n+1}) - v_{\boldsymbol{\varphi}}(S_t)\right).$$

We now have our actor update rule, and the only remaining step is to find the critic update rule. For A2C, when updating the critic's parameters, $\boldsymbol{\varphi}$, we simply minimise the square error between each state's estimated value, and its true observed value. Algorithm 1.5 shows the full pseudocode for the A2C algorithm.

1.3 State of the Art Actor-Critic Algorithms

For our empirical results, we will consider three state of the art reinforcement learning algorithms, and compare their results to each other, and to previous reinforcement learning techniques, such as those described in section 1.1. These algorithms are *soft actor-critic* (SAC), an off-policy stochastic policy optimisation algorithm, *proximal policy optimisation* (PPO), a policy improvement algorithm that improves by taking small steps, and finally *deep deterministic policy gradient* (DDPG),which simultaneously learns both a Q-function, and a policy. We explore each of these in more detail in the upcoming sections.

Algorithm 1.5 Advantage Actor-Critic

Initialise actor $\pi_{\boldsymbol{\theta}}$, parameterised by $\boldsymbol{\theta}$ and critic $v_{\boldsymbol{\varphi}}$, parameterised by $\boldsymbol{\varphi}$
Initialise $\boldsymbol{\theta}$, $\boldsymbol{\varphi}$ randomly
for each episode **do**
  $d\boldsymbol{\theta} \leftarrow 0$, $d\boldsymbol{\varphi} \leftarrow 0$
  Initialise environment, observe S_0
  repeat
    Take action A_t according to $\pi_{\boldsymbol{\theta}}$
    Receive reward R_{t+1}, observe state S_{t+1}
  until S_t terminal
  $R = 0$ if S_t is terminal, else $R = v_{\boldsymbol{\varphi}}(S_t)$
  for each $i \in \{t-1, ..., 0\}$ **do**
    $R \leftarrow R_i + \gamma R$
    Accumulate policy gradient using critic, $d\boldsymbol{\theta} \leftarrow d\boldsymbol{\theta} + \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(A_i \mid s_i, \boldsymbol{\theta})(R - v_{\boldsymbol{\varphi}}(S_i))$
    Accumulate critic gradient $d\boldsymbol{\varphi} \leftarrow d\boldsymbol{\varphi} + \nabla_{\boldsymbol{\varphi}}(R - v_{\boldsymbol{\varphi}}(S_i))^2$
  end for
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \eta d\boldsymbol{\theta}$
  $\boldsymbol{\varphi} \leftarrow \boldsymbol{\varphi} + \eta d\boldsymbol{\varphi}$
end for

1.3.1 Soft Actor-Critic

Soft actor-critic (SAC) is an off-policy, maximum entropy reinforcement learning algorithm introduced by Haarnoja *et al.* [13]. This paper discusses two major problems with traditional deep reinforcement learning methods. Firstly, the poor sample efficiency of even simple tasks; millions of environment interactions may be needed to solve even simple tasks, and this becomes even more challenging with complex environments. Secondly, these methods can be unstable with respect to hyperparameters, e.g. learning rate, environment parameters etc. This means that hyperparameter tuning must be performed, which can be expensive depending on the problem/environment. The motivation of SAC is to design an efficient (low environment steps) and stable (robust to hyperparameter changes) algorithm for both discrete and continuous action spaces.

To setup SAC, we first consider an agent who maximises a new objective function,

$$J(\pi) = \sum_{t=0}^{T} \mathbb{E}_{\pi}[R_t + \alpha \mathcal{H}(\pi(\cdot \mid S_t))], \tag{1.3.1}$$

where $\mathcal{H}(\pi)$ is the entropy (differential entropy) of the discrete (continuous) policy π, defined as

$$\mathcal{H}(X) = \mathbb{E}[-\log(f(X))],$$

where f is the probability mass/density function of X, depending if X is discrete/continuous. (1.3.1) also has a *temperature parameter* α, which determines how important the entropy term is compared to the reward signal, and determines how stochastic the optimal policy is. Note that as $\alpha \to 0$, we recover the conventional objective function.

Soft Policy Iteration

The SAC algorithm is derived starting from a maximum entropy variant of policy iteration, known as *soft policy iteration*. For this we have two new quantities of interest, the soft Q-value, and the soft state-value function. Analogously to the standard setting, the soft Q-value tells us the expected return from a given state/action pair, and the soft state-value function tells us the expected return from a given state. The soft Q-value can be computed using the standard update rule,

$$Q^{k+1}(s_t, a_t) \leftarrow r_t + \gamma \mathbb{E}_{\pi}[V(s_{t+1})] \tag{1.3.2}$$

where V is the *soft state-value function*, defined as

$$V(s_t) := \mathbb{E}_{\pi}[Q(s_t, A_t) - \log(\pi(A_t \mid s_t))]. \tag{1.3.3}$$

This update rule for Q^k will converge to the *soft Q-value*, and a proof of this convergence can be found in [13].

Details of Soft Actor-Critic Algorithm

Like standard actor-critic algorithms, SAC uses function approximators for both Q and the policy, and optimises both functions via stochastic gradient descent. The algorithm uses a parameterised state-value function, V_ψ, soft Q-function, Q_θ, and policy π_ϕ. We hence have three networks parameterised by ψ, θ, and ϕ respectively. The state-value network minimises the loss

$$J_V(\psi) = \mathbb{E}_{s_t \sim \mathcal{D}} \left[\frac{1}{2} \left(V_\psi(s_t) - \mathbb{E}_{\pi_\phi}[Q_(s_t, a_t) - \log(\pi_\phi(a_t \mid s_t))] \right)^2 \right], \tag{1.3.4}$$

where D is the distribution of previously observed states and actions. The gradient of (1.3.4) can be estimated using the unbiased estimator

$$\hat{\nabla} J_\psi(V) = \nabla_\psi V_\psi(s_t)(V_\psi(s_t) - Q_\theta(s_t, a_t) + \log(\pi_\theta(a_t, s_t)).$$

Similarly, using appropriate loss functions J and J_π as described in [13], we obtain gradient estimators for J and J_π with respect to θ and ϕ respectively as

$$\hat{\nabla}_\theta J_Q(\theta) = \nabla_\theta Q_\theta(a_t, s_t)(Q_\theta(s_t, a_t) - r_t - \gamma V_{\bar{\psi}}(s_{t+1})), \tag{1.3.5}$$

where $V_{\bar{\psi}}$ is a target value network, where $\bar{\psi}$ is an EWMA of all of the previous network weights. This has been show to stabilise training in [14]. Finally,

$$\hat{\nabla}_\phi J_\pi(\phi) = \nabla_\phi \log(\pi_\phi(a_t \mid s_t)) + (\nabla_{a_t} \log_\phi(a_t \mid s_t) - \nabla_{a_t} Q(s_t, a_t)) \nabla_\phi f_\phi(\epsilon_t; s_t). \tag{1.3.6}$$

Algorithm 1.6 Soft Actor-Critic

```
Initialise parameter vectors ψ, ψ̄, θ, φ.
Initialise replay buffer D.
for each iteration do
    Initialise S
    for each environment step do
        a_t ~ π_φ(a_t | s_t)
        Take action a_t, observe s_{t+1}, r_{t+1}
        D ← D ∪ {(s_t, a_t, r + t + 1, s_{t+1})}
    end for
    for each gradient step do
        ψ ← ψ − λ_V ∇̂_ψ J_V(ψ)
        θ ← θ − λ_Q ∇̂_θ J_Q(θ)
        φ ← φ − λ_π ∇̂_φ J_π(φ)
        ψ̄ ← τψ + (1 − τ)ψ̄
    end for
end for
```

1.3.2 Proximal Policy Optimisation

Proximal Policy Optimisation (PPO) is a reinforcement learning algorithm proposed by Schulman *et al.* [2], and similarly to SAC, attempts to find a balance between simple implementation, sample efficiency, and robustness to hyperparameter tuning. It is an extension of the *Trust Region Policy Optimisation* (TPRO) algorithm, introduced by Schulman *et al.* [15]. PPO performs an update at each step that minimizes a cost function while also making sure that the new policy is very similar to the previous one. While technically a class of different algorithms, this section will briefly outline

the most commonly used version of PPO, using a *clipped surrogate objective* function. Recall the gradient estimator used in the REINFORCE algorithm, (1.2.8)

$$\hat{g} = \mathbb{E}_\pi \left[\hat{A}(S_t, A_t) \nabla_{\boldsymbol{\theta}} \log(\pi_{\boldsymbol{\theta}}(A_t \mid S_t; \boldsymbol{\theta})) \right],$$

where $\hat{A}$ is some estimate of the advantage function, for example the n-step advantage estimate defined in (1.2.9). Let $\boldsymbol{\theta}'$ be the parameters of the previous actor (policy). We then define the *probability ratio*, $r_t(\boldsymbol{\theta})$ as

$$r_t(\boldsymbol{\theta}) := \frac{\pi(A_t \mid S_t; \boldsymbol{\theta})}{\pi(A_t \mid S_t; \boldsymbol{\theta}')}.$$

TPRO maximises a so-called *surrogate* objective function,

$$L(\boldsymbol{\theta}) := \mathbb{E} \left[r_t(\boldsymbol{\theta}) \hat{A}_t \right],$$

where again, $\hat{A}$ is some estimate of the advantage function. However, to avoid unnecessarily large policy changes, [2] suggests maximising a *clipped* version of this objective function, as follows:

$$L^{\text{CLIP}}(\boldsymbol{\theta}) := \mathbb{E} \left[\min \left(r_t(\boldsymbol{\theta}) \hat{A}_t,\ \text{clip}(r_t(\boldsymbol{\theta}), 1-\epsilon, 1+\epsilon) \hat{A}_t \right) \right],$$

where,

$$\text{clip}(x, a, b) := \begin{cases} a & \text{if } x < a \\ x & \text{if } a \leq x \leq b \\ b & \text{if } x > b \end{cases},$$

and ϵ is some hyperparameter, with [2] suggesting $\epsilon = 0.2$ as a sensible value.

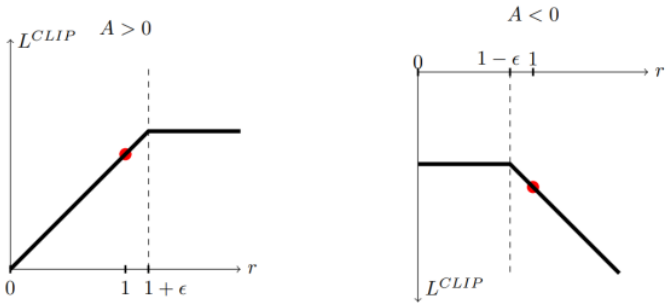

Figure 1.2: Clipped surrogate objective function used for PPO Algorithm. [2, Page 3]

Figure 1.2 shows that the change in probability ratio is clipped if it improves the value of the objective function, and is not clipped when it worsens the objective function. This gives a pessimistic bound on the objective function, and leads to smaller changes in the policy. The implementation of PPO is essentially very similar to REINFORCE except L^{CLIP} is maximised instead of the standard policy gradient objective function. Hence we do not include its pseudocode in this section.

1.3.3 Deep Deterministic Policy Gradient

Deep Deterministic Policy Gradient (DDPG) was first proposed by Lillicrap *et al.* [16]. It is an actor-critic approach to the DPG algorithm proposed by Silver *et al.* [17]. DPG keeps track of an actor, represented by $\mu(s \mid \theta^\mu)$, parameterised by the network θ^μ. As with a normal actor, this specifies a policy, mapping states to actions. The critic, $Q(s, a)$ is learned as in standard

Q-learning, with the Bellman equation.

Algorithm 1.7 Deep Deterministic Policy Gradient

Initialise actor $\mu(s \mid \theta^\mu)$ and critic $Q(s, a \mid \theta^Q)$ with parameters θ^μ and θ^Q.
Initialise target network μ' and Q' with weights $\theta^{\mu'} \leftarrow \theta^\mu$, and $\theta^{Q'} \leftarrow \theta^Q$
Initialise replay buffer $\mathcal{R}$.
for each episode **do**
  Initialise a random process $\mathcal{N}$
  Observe S_0
  for each environment step **do**
    Select action $A_t = \mu(S_t \mid \theta^\mu) + \mathcal{N}_t$
    Take action A_t, observe S_{t+1}, R_{t+1}
    $\mathcal{R} \leftarrow \mathcal{R} \cup \{(S_t, A_t, R_{t+1}, S_{t+1})\}$
    Sample a random minibatch of N transitions $(S_i, A_i, R_{i+1}, S_{i+1})$ from $\mathcal{R}$
    Set $y_i = R_{i+1} + \gamma Q'(S_{i+1}, \mu'(S_{i+1} \mid \theta^{\mu'}) \mid \theta^{Q'})$
    Update critic by minimising loss $L := \frac{1}{N}\sum_{i=1}^{N}(y_i - Q(S_i, A_i \mid \theta^Q))^2$
    Update actor policy using the sampled policy gradient

$$\nabla_{\theta^\mu} \approx \frac{1}{N}\sum_{i=1}^{N} \nabla_a Q(s, a \mid \theta^Q)|_{s=S_i, a=\mu(S_i)} \nabla_{\theta^\mu} \mu(s \mid \theta^\mu)|_{s=S_i}$$

    Update the target networks

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$

$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  end for
end for

1.4 Stochastic Processes and Control

We will now switch gears and move onto the theory of stochastic control. Chapter 2 will combine both the theory of reinforcement learning and stochastic control in a practical setting. The theory in this section is adapted from [18, Chapter 5]. Stochastic control problems regularly appear in applications to financial mathematics. As an example, in 1971, Merton created and solved the optimal investment problem, concerned with an investor choosing a proportion of his wealth to allocate into stocks vs a risk-free asset, using the techniques of stochastic control [19]. The goal of stochastic control problems is in general to maximise some expected payoff function by selecting some strategy which affects the dynamics of the stochastic system. In this thesis, we will set up the problem of optimal market making (to be discussed in Section 1.5) as a stochastic control problem, find some optimal strategy through standard techniques, and also find a strategy through a machine learning approach. The main objects of consideration in the realm of stochastic control are those of *controlled diffusions*, defined below.

Definition 1.4.1 (Controlled diffusion)**.** A controlled diffusion, $\{X_t^\pi\}_{t>0}$ is a stochastic process defined by

$$dX_t^\pi = \mu(t, X_t^\pi, \pi_t)dt + \sigma(t, X_t^\pi, \pi_t)dB_t, \tag{1.4.1}$$

where $\{\pi_t\}_{t>0}$ is a process adapted to $\mathcal{F}_t$, known as the *control process* (or more simply, control). μ and σ are real valued functions of time, the controlled diffusion, and the control process. $\{B_t\}$ is a standard Brownian motion.

Note that each control, π gives rise to a different controlled diffusion, X_t^π, and in the general case, (1.4.1) is not an SDE, and $\{X_t^\pi\}$ does not satisfy the Markov property in general. For

our purposes here, we will restrict any selection of a control π to be a function of two variables, $(t,x) \longrightarrow \pi(t,x)$, so that the control process is $\pi(t,X_t)$, and the controlled diffusion is given by

$$dX_t^\pi = \mu(t,X_t^\pi,\pi(t,X_t))dt + \sigma(t,X_t^\pi,\pi(t,X_t))dB_t. \tag{1.4.2}$$

Any control of this form is known as a *Markov control*, and (1.4.2) is an SDE, whose solution has the Markov property. The general problem we attempt to solve is maximising an expectation of an additive payoff. We first define the *performance criterion*.

Definition 1.4.2 (Performance criterion)**.** The performance criterion, $J^\pi(x)$ is given by:

$$J^\pi(x) := \mathbb{E}\left[e^{-\int_0^T r_s ds}\psi(X_T^\pi) + \int_0^T e^{-\int_0^t r_s ds}\gamma_t dt\right],$$

where $r_s = r(s,X_s^\pi,\pi_s)$, and $\gamma_t = \gamma(t,X_t^\pi,\pi_t)$.

In this thesis, we will use the convention that $r \equiv \gamma \equiv 0$. We can hence rewrite

$$J^\pi(x) = \mathbb{E}[\psi((X_T^\pi)].$$

1.4.1 Dynamic Programming Principle

The two key tools in solving stochastic control problems are the *dynamic programming principle* (DPP), and the related non-linear PDE, known as the *Hamilton-Jacobi-Bellman* (HJB) equation. The DPP is a way of solving stochastic control problems by working backwards from the terminal state, and the HJB equation can be viewed as the infinitesimal version of this principle. Before exploring these two important tools, we first define admissible controls.

Definition 1.4.3 (Admissible controls)**.** The set of admissible controls is defined as

$$\mathcal{A}_{s,t} := \{\pi : [s,t]\times\mathbb{R} \to \mathbb{R} : (1.4.1) \text{ has a unique square-integrable strong solution}\}.$$

We will only consider controls π from this set to ensure that any solutions are well defined. Our main objective is to find the two following quantities:

$$H(x) := \sup_{\pi\in\mathcal{A}_{0,T}} \mathbb{E}(\psi(X_T^\pi)) = \sup_{\pi\in\mathcal{A}_{0,T}} J^\pi(x), \tag{1.4.3}$$

which will be referred to as the *value function*, and

$$\pi^* \in \mathcal{A}_{0,T} \text{ such that } H(x) = J^{\pi^*}(x) \text{ (if it exists)}, \tag{1.4.4}$$

a control that maximises our value function. This is known as the *optimal control*. The first part of our approach will be to define a modified version of H in the following way.

(i)
$$J^\pi(t,X_t) = \mathbb{E}[\psi(X_T^\pi)|\mathcal{F}_t].$$

(ii)
$$H(t,X_t) = \sup_{\pi\in\mathcal{A}_{t,T}} \mathbb{E}[\psi(X_T^\pi)|\mathcal{F}_t] = \sup_{\pi\in\mathcal{A}_{t,T}} J^\pi(t,X_t).$$

We refer to $H(t,x)$ as the *dynamic value function*. Notice that $H(0,x) = H(x)$. The dynamic programming principle is the relation between the dynamic function at different points in time.

Theorem 1.4.4 (Dynamic Programming Principle)**.**

$$H(s,X_s) = \sup_{\pi\in\mathcal{A}_{s,t}} \mathbb{E}[H(t,X_t^\pi)|\mathcal{F}_s] \qquad \textit{for all } s \le t \le T. \tag{1.4.5}$$

Remark 1.4.5. In plain words, the dynamic programming principle says that if you know how to act optimally between t and T, then you only need to know how to act optimally between s and t

Proof. See A.1. □

1.4.2 Hamilton-Jacobi-Bellman Equation

In order to set up our final tool for solving stochastic control problems, it will first be helpful to recall the Feynman-Kac formula. We will then adapt this formula to a controlled diffusion environment to yield the HJB equation.

Theorem 1.4.6 (Feynman-Kac)**.** *Let X satisfy*

$$dX_t = \mu(t, X_t)dt + \sigma(t, X_t)dB_t.$$

Let $f : [0, T] \times \mathbb{R} \to \mathbb{R}$ be a function satisfying

$$f(t, X_t) = \mathbb{E}(\varphi(X_T)|\mathcal{F}_t).$$

Assume $f \in \mathcal{C}^{1,2}$, then:

$$\frac{\partial f}{\partial t} + \mu(t, x)\frac{\partial f}{\partial x} + \frac{1}{2}\sigma^2(t, x)\frac{\partial^2 f}{\partial x^2} = 0,$$

$$f(T, x) = \varphi(x).$$

We are now ready to introduce the final tool, the HJB equation.

Theorem 1.4.7 (Hamilton-Jacobi-Bellman Equation)**.** *Let X^π be a controlled diffusion satisfying*

$$dX_t^\pi = \mu(t, X_t^\pi, \pi_t)dt + \sigma(t, X_t^\pi, \pi_t)dB_t.$$

Let $H : [0, T] \times \mathbb{R} \to \mathbb{R}$ be a function satisfying

$$H(t, X_t) = \sup_{\pi \in \mathcal{A}_{t,T}} \mathbb{E}(\varphi(X_T^\pi)|\mathcal{F}_t)$$

Assume $H \in \mathcal{C}^{1,2}$, then:

$$\sup_{\pi \in \mathbb{R}} \left\{ \frac{\partial H}{\partial t} + \mu(t, x, \pi)\frac{\partial H}{\partial x} + \frac{1}{2}\sigma^2(t, x, \pi)\frac{\partial^2 H}{\partial x^2} \right\} = 0, \tag{1.4.6}$$

$$H(T, x) = \varphi(x).$$

Additionally, suppose $\pi^* : [0, T] \times \mathbb{R} \to \mathbb{R}$ attains the supremum in (1.4.6) for all t, x. Then π^* is an optimal Markov control.

Proof. See A.2. □

Notice that most terms in this equation, and the terminal condition are exactly the same as in the standard diffusion setting, the only difference being that we now have a free variable π, over which we take a supremum. We are essentially maximising the equation with respect to the π argument. We are aiming to find some maximiser, π^* which attains the supremum, and this control will again depend on t and X_t^π.

1.5 Electronic Market Making

The US Securities and Exchange Commission (SEC) defines a market maker as "a firm that stands ready to buy or sell a stock at publicly quoted prices." [20]. According to the New York Stock Exchange (NYSE), in 2021, the average daily trading volume of US cash equities was almost $580 billion [21], and in order to provide enough liquidity to enable these vast trading volumes, market makers are an essential feature of modern markets. They allow for higher price visibility for investors, and also allow trades to be made at a reasonable price instantly, increasing market efficiency.

1.5.1 The Limit Order Book

The following description of the limit order book is adapted from Bouchard *et al.* [22, Chapter 3]. Modern exchanges adopt an automated approach to matching buyers and sellers through the use of a *limit order book* (LOB). These markets are driven by incoming orders, which may be of two types.

- Limit orders - these orders specify the highest price the trader is willing to buy at (known as a bid price), or the lowest price a trader is willing to sell as (known as the ask price).
- Market orders - these orders leave the price unspecified.

Once a limit order is placed, it must wait until it is *matched*, or until the order sender cancels the order. However a market order is immediately executed against the best limit order currently in the LOB. To formalise the definition, an order x is associated with the following four attributes.

- The sign, $\epsilon_x \in \{-1, 1\}$, representing whether x is a buy order ($\epsilon_x = 1$), or a sell order ($\epsilon_x = -1$).
- The price, $p_x \in \mathbb{R}^+$.
- The volume $v_x \in \mathbb{Z}^+$[1].
- The timestamp $t_x \in \mathbb{R}^+$.

When a buy order x arrives, the exchange's matching algorithm will attempt to find a sell order y, with $p_x \geq p_y$. If such an order is found, and if $v_y \geq v_x$, then the order is fully matched and the trade is made. However if $v_x > v_y$, the remaining volume $v_x - v_y$ must be matched against another order. If such a y cannot be found, or the full volume cannot be matched against existing LOB orders, then x becomes an active order in the LOB.

Best Bid/Ask

We denote the LOB state at time t, i.e. the set of all orders in the LOB at time t as $\mathcal{L}_t$. The set of buy orders is denoted as

$$\mathcal{B}_t := \{x \in \mathcal{L}_t : \epsilon_x = 1\},$$

and the set of sell orders as

$$\mathcal{S}_t := \{x \in \mathcal{L}_t : \epsilon_x = -1\},$$

then the *best bid* (or bid price) at t is defined as

$$b_t := \max_{x \in \mathcal{B}_t} p_x,$$

and the *best ask* (or ask price) at t is defined as

$$a_t := \min_{x \in \mathcal{S}_t} p_x.$$

a_t and b_t are the prices that a trader can immediately buy or sell at respectively. We also define the *mid-price* as

$$m_t := \frac{b_t + a_t}{2},$$

the average of the bid and ask prices. Figure 1.3 shows an example snapshot of a LOB, where $b_t = 27.47$, and $a_t = 27.49$.

[1] Some exchanges will allow for fractional volumes if the asset price is large.

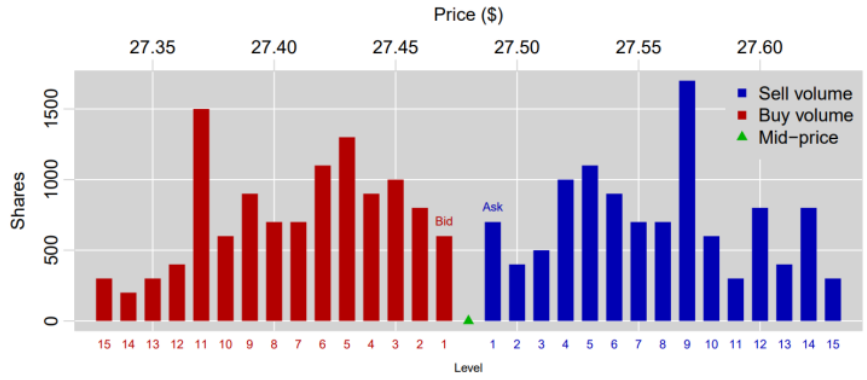

Figure 1.3: Top 15 levels of the Nasdaq LOB of Twitter, Inc. (TWTR) on 9 September 2015 at 3:26:22.84pm (ET). [3, Page 18]

.

1.5.2 The Avellaneda-Stoikov Market Making Model

One of the most frequently used stochastic models for market making is the Avellaneda-Stoikov model. In this section, we outline the dynamics of this model proposed in [23].

Controlled Diffusion Dynamics

Firstly, it is assumed that an asset mid-market price evolves according to the stochastic differential equation,

$$dS_t = \sigma dW_t. \tag{1.5.1}$$

Here $\{W_t\}$ is a standard Brownian motion, σ a constant, and the asset has initial value $S_0 = s$. This price process represents the value of the agent's assets at the final period, but it is important to note that trades can't be executed at this price. We see that in this model, Avellaneda and Stoikov assume no interest in the money market, and implicitly assume that the agent has no opinion on the drift or autocorrelation structure for the asset price. Suppose that the market maker (MM) wishes to quote a bid price of p^b and an ask price of p^a. Then we define the distances

$$\delta^b := s - p^b,$$

and

$$\delta^a := p^a - s,$$

where s is the current asset price. It is assumed that the MM's buy orders are filled at Poisson rate $\lambda^b(\delta^b)$, and the MM's sell orders are filled at Poisson rate $\lambda^a(\delta^a)$. Note λ^b and λ^a are functions of the distances defined above respectively. The reasoning behind Poisson order arrival rates is based on two empirical facts; the distribution of the size of market orders follows a power law [24, 25], and that the limit order depth changes proportionally to the size of the market order [26]. It is argued in [23] that these order arrival rates can be assumed to take the form

$$\lambda^a(\delta) = \lambda^b(\delta) = Ae^{-k\delta}. \tag{1.5.2}$$

Under all these assumptions, the agent's cash process, $\{X_t\}_{t>0}$ has the dynamics

$$dX_t = p^a dN_t^a - p^b dN_t^b, \tag{1.5.3}$$

where $\{N_t^a\}$ and $\{N_t^b\}$ are Poisson processes with intensities $\lambda^a(\delta^a)$ and $\lambda^b(\delta^b)$ respectively. The MM's inventory process, $\{Q_t\}_{t>0}$ representing the total volume of asset the MM holds at time t is given by

$$Q_t = N_t^b - N_t^a. \tag{1.5.4}$$

Dynamic Value Function

Avellaneda and Stoikov suggest the MM has the dynamic value function

$$H(s,x,q,t) = \sup_{\delta^a,\delta^b} \left\{ \mathbb{E}\left[-\exp(-\gamma W_T) \mid \mathcal{F}_t \right] \right\}, \tag{1.5.5}$$

where $W_t := X_t + Q_t S_t$ is the wealth at time t. Notice at $t = T$, we have

$$H(s,x,q,T) = -\exp(-\gamma(x+qs)).$$

This is known as *constant absolute risk aversion* (CARA) utility, and is studied in [27].

Solving the System Using the Hamilton-Jacobi-Bellman Equation

To begin finding our optimal controls, δ^{a*}, and δ^{b*}, we must write down the HJB equation for our stochastic control system. This equation is as follows:

$$\sup_{\delta^a,\delta^b} \left\{ \frac{\partial H}{\partial t} + \frac{\sigma^2}{2}\frac{\partial^2 H}{\partial s^2} + \mathcal{L}^Q H \right\} = 0, \tag{1.5.6}$$

where $\mathcal{L}^Q$ is the infinitesimal generator of $\{Q_t\}$. Since $\{Q_t\}$ is a linear combination of two jump processes (namely $\{N_t^b\}$ and $\{N_t^a\}$), it is itself a jump process, hence $\mathcal{L}^Q$ is a finite difference operator. Hence the HJB equation for our system becomes

$$\begin{aligned} \sup_{\delta^a,\delta^b} \Bigg\{ &\frac{\partial H}{\partial t} + \frac{\sigma^2}{2}\frac{\partial^2 H}{\partial s^2} + \lambda^b(\delta^b)\Big(H(s,x-s+\delta^b,q+1,t) - H(s,x,q,t)\Big) \\ &+ \lambda^a(\delta^a)\Big(H(s,x+s-\delta^a,q-1,t) - H(s,x,q,t)\Big)\Bigg\} = 0, \end{aligned} \tag{1.5.7}$$

with terminal condition $H(s,x,q,T) = -\exp(-\gamma(x+qs))$, the CARA utility function. The solution to this non-linear PDE is continuous in s, x, and t, and depends on the discrete values q. We can simplify the problem using the ansatz

$$H(s,x,q,t) = -\exp(-\gamma(x + v(s,q,t))), \tag{1.5.8}$$

where v is some function not dependent on x. Directly substituting this into (1.5.7), and rearranging yields

$$\begin{aligned} &\frac{\partial v}{\partial t} + \frac{\sigma^2}{2}\frac{\partial^2 v}{\partial s^2} - \gamma\frac{\sigma^2}{2}\left(\frac{\partial v}{\partial s}\right)^2 + \sup_{\delta^b}\left\{\frac{\lambda^b(\delta^b)}{\gamma}\Big(1 - \exp\big(-\gamma(s-\delta^b - v(s,q+1,t) + v(s,q,t))\big)\Big)\right\} \\ &+ \sup_{\delta^a}\left\{\frac{\lambda^a(\delta^a)}{\gamma}\Big(1 - \exp\big(\gamma(s+\delta^a + v(s,q-1,t) - v(s,q,t))\big)\Big)\right\}. \end{aligned} \tag{1.5.9}$$

(1.5.9) is a highly non-linear PDE, and is computationally hard to solve. Avellaneda and Stoikov suggest an asymptotic expansion of v in the inventory variable q. After these expansions are computed, the PDE is then solved approximately, as there is no closed form solution. Hence the controls proposed by Avellaneda-Stoikov are only semi-optimal. After solving the PDE approximately, we obtain the two semi-optimal controls,

$$\delta^{b*}(t,Q_t) \approx \gamma\sigma^2(T-t)Q_t + \frac{1}{\gamma}\log\left(1+\frac{\gamma}{k}\right), \tag{1.5.10}$$

$$\delta^{a*}(t,Q_t) \approx -\gamma\sigma^2(T-t)Q_t + \frac{1}{\gamma}\log\left(1+\frac{\gamma}{k}\right), \tag{1.5.11}$$

and the semi-optimal bid/ask prices are given by

$$p_t^{b*} \approx S_t - Q_t\gamma\sigma^2(T-t) - \frac{\gamma}{2}\sigma^2(T-t) - \frac{1}{\gamma}\log\left(1+\frac{\gamma}{k}\right)$$

$$p_t^{a*} \approx S_t - Q_t \gamma \sigma^2 (T-t) + \frac{\gamma}{2}\sigma^2 (T-t) + \frac{1}{\gamma}\log\left(1+\frac{\gamma}{k}\right)$$

These controls show that as the inventory increases, both the bid and ask get lower, i.e. the market is skewed downwards as we would expect. The opposite is true if the inventory decreases. If $Q_t = 0$, the spread is symmetric about the mid-price. If volatility is higher, the market maker will quote a wider market.

Chapter 2

Reinforcement Learning Approach to Avellaneda-Stoikov Market Making

We now wish to apply the reinforcement learning techniques discussed in Section 1.1 to the market making problem discussed in Section 1.5. Gorse and Lim observe that reinforcement learning approaches can outperform Avellaneda-Stoikov market making [9], so we wish to see if this can also be achieved with actor-critic reinforcement learning algorithms. In this chapter, we will consider the performance of three agents. One acts optimally according to Avellaneda-Stoikov, one will use the traditional temporal difference learning approach of tabular Q-learning, and one will use the actor-critic algorithm, A2C, described in Section 1.2.3. The aim of this chapter is essentially to prove the concept that actor-critic algorithms can perform well on a theoretical foundation on the market making problem, so that in Chapter 3, we can apply them to real data.

2.1 Environment

The reinforcement learning algorithms will both learn on identical environments. Since the theoretical framework is clearly defined, this environment is fairly simple. The environment keeps track of the following quantities; the underlying asset price, S_t, the agent's position, Q_t, and the agent's wealth W_t. The environment has an initial time $t = 0$, and terminal time $T = 1$, with 200 timesteps, so that $\Delta t = 0.005$. At each step, the environment updates its asset price according to the Euler scheme,

$$S_{t+\Delta t} = S_t + \sigma\sqrt{\Delta t} Z_t, \tag{2.1.1}$$

where $Z_t \sim N(0,1)$ for all t. Trades are decided by the Poisson processes N_t^a, and N_t^b. To simulate these, we use the following lemma.

Lemma 2.1.1. *Let $\{N_t\}$ be a Poisson process with rate λ. Then*

$$\mathbb{P}(N_{t+dt} - N_t = j) = \begin{cases} 1 - \lambda dt + \mathcal{O}(dt) & j = 0 \\ \lambda dt + \mathcal{O}(dt) & j = 1 \\ \mathcal{O}(dt) & j > 1 \end{cases}$$

as $dt \to 0$.

Combining this lemma with (1.5.2), we can calculate the probability a trade is made after a small timestep Δt, i.e.

$$\mathbb{P}(N_{t+\Delta t} = N_t + 1) = Ae^{-k\delta_t^b}\Delta t$$

$$\mathbb{P}(N_{t+\Delta t} = N_t) = 1 - Ae^{-k\delta_t^b}\Delta t$$

These update rules give us a complete picture of the stochastic dynamics of our environment. Once the agent provides an action (i.e. a bid/ask price) to the environment, the environment determines whether a trade is made using the update rule above, and then updates the position and cash balances. The agent then steps the price forward according to the update rule (2.1.1), and calculates the new wealth of the agent.

2.1.1 State Space

The states of our market making environment contain three pieces of information:

- The current asset price, S_t.
- The current inventory, Q_t.
- The number of remaining timesteps, $\frac{T-t}{\Delta t}$.

Since we are also using tabular Q-learning, we must discretise our environment states. Fortunately, the number of timesteps and the inventory are discrete, (and also bounded if we include a position limit $\bar{Q}$). However the asset price is a continuous variable, so we must make it discrete. To do so, we have some minimum price $S_{\min}$, and some number of buckets, M, of size B, and we define

$$\tilde{S}_t := \max\{s = S_{\min} + iB : i \in \{0, M-1\}, s < S_t\}. \tag{2.1.2}$$

This gives us the states of the environment as the tuple $(\tilde{S}_t, Q_t, \frac{T-t}{\Delta t})$. The number of states is hence $B \times (2\bar{Q}+1) \times \frac{T}{\Delta t}$.

2.1.2 Action Space

The actions that the agent can take represent what market they are quoting at time t. To do this, we represent the market as two quantities: the spread, $\Sigma_t > 0$ and the offset, $\omega_t \in \mathbb{R}$. More concretely, suppose the asset price is S_t. Then,

$$b_t = S_t + \omega_t - \frac{\Sigma_t}{2}, \tag{2.1.3}$$

$$a_t = S_t + \omega_t + \frac{\Sigma_t}{2}, \tag{2.1.4}$$

where b_t and a_t are the bid/ask prices set by the agent. These prices are hence fully captured by Σ_t and ω_t. If $\omega_t > 0$, then the market is positively skewed, i.e. the bid price is closer to the mid-price than the ask. If $\omega_t = 0$, then the market is symmetrical about the mid-price. If $\omega_t < 0$, then the market is negatively skewed, i.e. the ask price is closer to the mid-price than the bid. Ideally, Σ_t and ω_t would be continuous, but since one of the agents we use will use tabular Q-learning, we must discretise the action space. To do this, we introduce some maximum offset $\tilde{\omega} > 0$, and some maximum spread $\tilde{\Sigma} > 0$. We then introduce two integers, N_ω and N_Σ, representing the number of levels that the offset and spread can take. The agent then has possible actions

$$\mathcal{A} = \{(\omega, \Sigma) : \omega \in \{1, 2, ...N_\omega - 1, N_\omega\},\ \Sigma \in \{1, 2, ..., N_\Sigma - 1, N_\Sigma\}\}.$$

Suppose the agent chooses the action $(A_\omega, A_\sigma) \in \mathcal{A}$. Then

$$\omega_t = -\tilde{\omega} + \frac{2\tilde{\omega}}{N_\omega} A_\omega,$$

and

$$\Sigma_t = \frac{\tilde{\Sigma}}{N_\Sigma} A_\Sigma.$$

We can then simply use (2.1.3) and (2.1.4) to recover the agent's market. Overall, there are $N_\Sigma N_\omega$ actions, and in our experiments, we use $N_\Sigma = 30$ and $N_\omega = 16$.

2.1.3 Reward Signal

Recall from (1.5.5) that the goal of the agent is to maximise the following quantity

$$\mathbb{E}\left[-\exp(-\gamma W_T)\right],$$

i.e. the expected value of some concave function of the terminal wealth W_T. Ritter, 2017 [28] shows that a policy which maximises a quantity of the form $\mathbb{E}[u(W_T)]$ for some concave u is also optimal for maximising the quantity

$$\mathbb{E}[W_T] - \frac{\kappa}{2}\mathbb{V}[W_T], \tag{2.1.5}$$

for some $\kappa > 0$. This is known as *mean-variance equivalence*. For our specific problem, it can be shown that $\kappa = 2\gamma$. We can use this equivalence to formulate our reward signal. Notice that by the linearity of expectation,

$$\mathbb{E}[W_T] = W_0 + \sum_{t=1}^{T} \mathbb{E}[W_t - W_{t-1}], \tag{2.1.6}$$

and supposing that $\{W_t\}$ has independent increments, which in our setting is the case due to the independent increments of $\{S_t\}$, we also have that

$$\mathbb{V}[W_T] = \sum_{t=1}^{T} \mathbb{V}[W_t - W_{t-1}]. \tag{2.1.7}$$

If we let $\delta W_t := W_t - W_{t-1}$, then combining (2.1.6) and (2.1.7) with (2.1.5), our agent is simply trying to maximise

$$\sum_{t=1}^{T} \mathbb{E}[\delta W_t] - \frac{\kappa}{2}\mathbb{V}[\delta W_t].$$

This motivates a reward signal at every step of the environment. We let

$$R_{t+1} = \delta W_t - \frac{\kappa}{2}(\delta W_t - \hat{\mu})^2,$$

where $\hat{\mu}$ is the sample mean of the previously observed δW_ts. The intuition behind this reward signal is that the MM wishes to increase its wealth (represented by the first term), but also reduce the variance of its changes in wealth, i.e. not be exposed to large price swings (represented by the second term). Hence by keeping track of this quantity, we hope to reward the market maker for increasing its wealth whilst maintaining a low exposure to price changes, i.e. low position.

2.2 Results

After training the Q-agent and the A2C agent for 200,000 environment steps, we performed 1,000 new simulations and recorded the terminal wealth of each reinforcement learning agent, and the optimal strategy described in Section 1.5.2. Figure 2.1 shows the distribution of terminal wealths of each agent.

Figure 2.1: Terminal wealth distributions of the optimal strategy, Q-learning agent, and A2C agent.

We observe that the Q-learning agent has a similar variance in terminal wealth to the optimal strategy, but a slightly lower average wealth. However the A2C agent outperforms the optimal strategy in terms of average terminal wealth, indicating it is possible for reinforcement learning to outperform the theoretical optimal in terms of average wealth. The distribution is more spread out, indicating a trade off between high average terminal wealth and low variance in terminal wealth. This is still an impressive result however.

Table 2.1 shows that the A2C agent has the highest average wealth, and also the highest number of trades. If we consider the wealth per trade, the optimal strategy is clearly extracting the most wealth per trade, however this is probably due to the observed tighter spread of the A2C agent. The A2C agent is providing the most liquidity, which is advantageous if the exchange offers rebates for example. The tabular Q-learning agent performs worst on most metrics, however does have a lower standard deviation of wealth than the A2C agent. In conclusion, the A2C agent clearly outperforms the tabular Q-learning agent, and also outperforms the optimal agent in terminal wealth.

Statistic	Optimal AS	Tabular Q	Actor-Critic
Mean Wealth	47.78	37.42	58.19
Median Wealth	47.78	37.42	58.19
Wealth Standard Deviation	5.90	8.25	14.45
Mean Trades	76.78	116.48	143.14
Wealth per Trade	0.62	0.32	0.40

Table 2.1: Statistics for different algorithms on Avellaneda-Stoikov environment

To get a sense of what each agent is actually doing, we will inspect some realisations of a single episode for each agent, and see we can observe any interesting behaviour. Firstly, we look at the theoretically optimal agent.

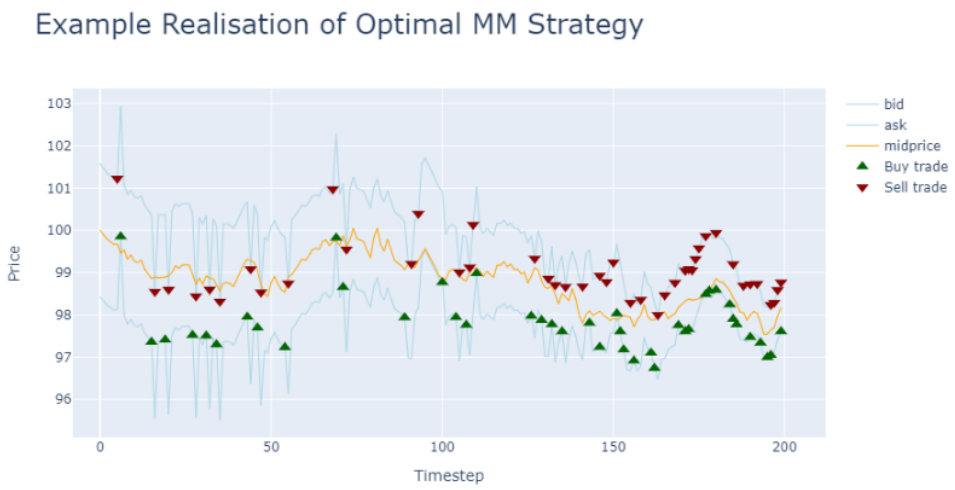

Figure 2.2: A single realisation of the optimal Avellaneda-Stoikov strategy.

Figure 2.2 shows that towards the start of the period, the optimal market maker is very passive, quoting quite a wide spread. When they receive a trade, they attempt to immediately exit their position by quoting very close to the mi-price (sometimes even crossing it) on the opposite side. This behaviour is observed often in the first half of the realisation. However towards the end, the market maker tightens its spread and performs more active market making.

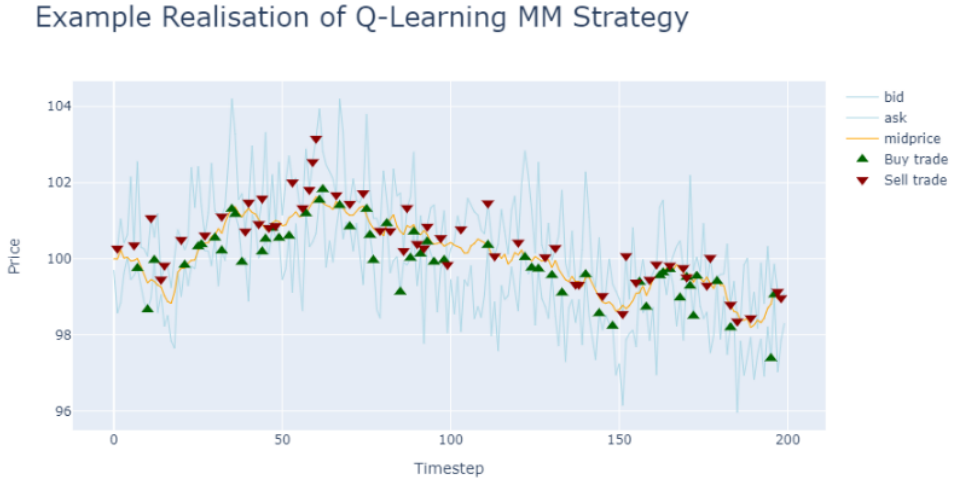

Figure 2.3: A single realisation of the tabular Q-learning strategy.

Figure 2.3 shows a single realisation of the Q-learning agent. The quotes here are more erratic, possibly verging on random. This could indicate that more training is needed, as the state/action space is very large, so it is unlikely that the Q-values converged fully.

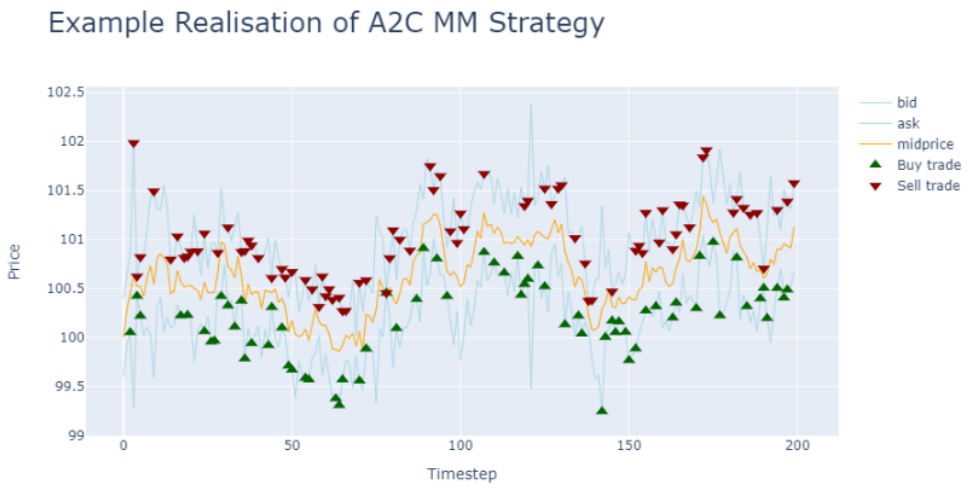

Figure 2.4: A single realisation of the A2C strategy.

Figure 2.4 shows the A2C agent quotes a fairly consistent spread, with a larger trading volume than the optimal strategy. This lines up with our empirical observation that the optimal agent has a lower variance in its wealth (i.e. it tends to make less risky decisions) while receiving a lower wealth on average. We now consider some typical wealth paths of each strategy.

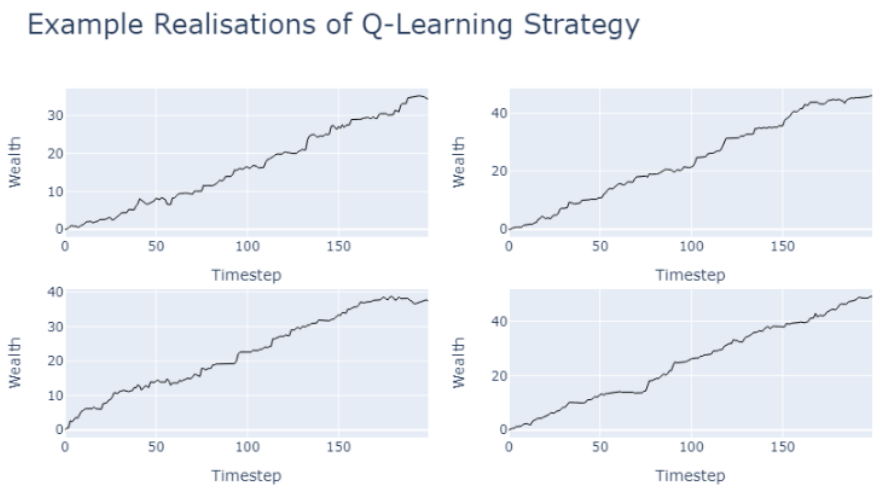

Figure 2.5: Wealth paths for Q-Learning Agent.

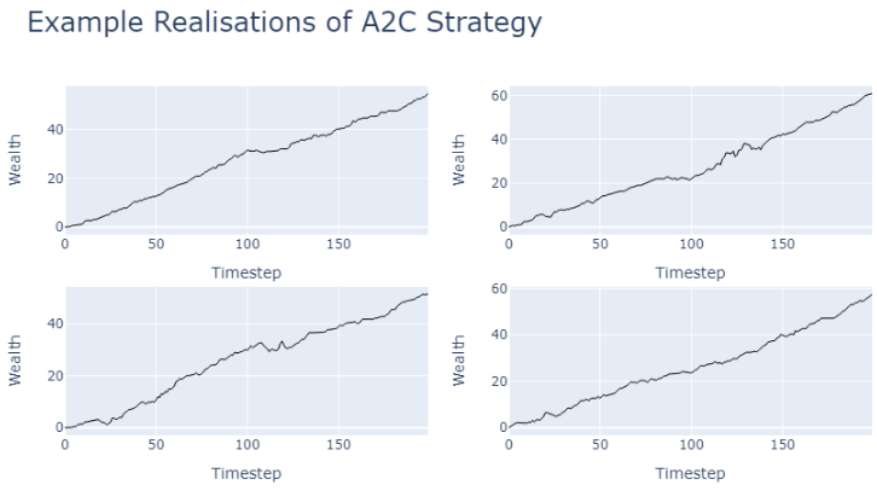

Figure 2.6: Wealth paths for Advantage Actor-Critic Agent.

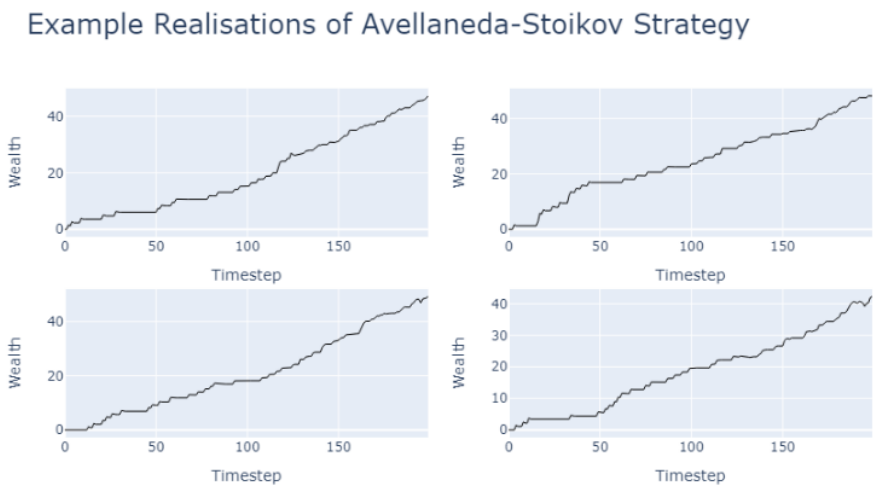

Figure 2.7: Wealth paths for Avellaneda-Stoikov Agent.

Figures 2.5, 2.6, and 2.7 all show that each agent manages to create consistent wealth, but as expected from previous results, the A2C strategy creates more wealth than the other two.

2.2.1 Results Summary

We have observed that the actor-critic method, Advantage Actor-Critic is able to outperform the semi-optimal strategy of Avellaneda-Stoikov in terms of wealth, but does have a slightly higher variance in terminal wealth. However considering the empirical wealth paths in Figure 2.6, the growth in wealth across each path is steady. Overall, we have shown that actor-critic approaches can reproduce and outperform Avellaneda-Stoikov, and we will now attempt to see if this also holds on real market data in the next chapter.

Chapter 3

Actor-Critic Approach Using Market Data

After observing the success of deep reinforcement learning algorithms in a theoretical environment, and their improvement over traditional reinforcement learning techniques, we now wish to see if that performance can translate to real market data. We will look at three state of the art algorithms, namely SAC, PPO and DDPG, and compare their performance in this real environment.

3.1 Data Source

For the testing of these algorithms, we will use limit order book data of a small tick stock. For our purposes, we consider TSLA. The data were sourced from LOBSTER, a data provider for high quality granular limit order book data. The data contain the following relevant fields.

- Time, recorded as seconds after midnight, down to the nanosecond.
- Events, which can be submissions of new orders, cancellations, executions etc. These are associated with a volume and a side (buy/sell).
- 10 levels of bid/ask prices and volumes.

The data were cleaned and transformed, so that each event was associated with its previous limit order book state. The reinforcement learning algorithms will have access to the limit order book state, and create a market using that state. Then, the event will be compared to the market, and we determine whether a trade was made.

3.2 Environment

One of the most important pieces of the reinforcement learning process is the environment. Market making is a particularly difficult environment to create using historical data, as it is difficult to capture the impact that the market maker has on the environment through its quotes. For example, a market making agent who quotes a very tight spread will not generate more trades if historical data is solely used than if they simply quoted a single tick inwards from the best bid/ask. For this reason a simulated market environment, such as one described by Lehalle *et al.* [29] was considered, however it proved difficult to capture the true order book dynamics. Instead, a compromise solution was implemented. We assume that the market maker is quoting small volumes, so as not to have a large impact on the bid/ask volumes, and we also assume that the market maker receives additional trades not observed in the data if it is quoting between the actual best bid/ask. These *artificial trades* will arrive at some constant rate λ, multiplied by the proportional distance between the mid-price and observed the best bid/ask. Specifically, let A_t^{BUY} be the event that an artificial buy trade is made

$$\mathbb{P}(A_t^{\text{BUY}}) = \lambda dt \frac{b_t^{\text{MM}} - b_t^{\text{Obs}}}{m_t - b_t^{\text{Obs}}} \mathbb{1}_{\{b_t^{\text{MM}} > b_t^{\text{Obs}}\}},$$

where b_t^{MM} is the bid price that the agent is quoting, and b_t^{Obs} is the bid price observed in the historical data. Analogously,

$$\mathbb{P}(A_t^{\text{SELL}}) = \lambda dt \frac{a_t^{\text{Obs}} - a_t^{\text{MM}}}{a_t^{\text{Obs}} - m_t} \mathbf{1}_{\{a_t^{\text{MM}} < a_t^{\text{Obs}}\}}.$$

This solution gives a good compromise between representing the impact of the market maker's quotes, and also giving a true reflection of the market dynamics.

The dynamics of the environment are fairly simple. The environment is reset, and a random time index of the limit order book is chosen (in the training data, there were over one million rows to select from). The agent will then set its bid ask price, and the environment determines whether a trade was made against the agent's market. If a trade is made, the agent updates its position, cash, and wealth, and is also provided with a small rebate, similar to real exchanges providing rebates to liquidity providers. The environment then steps to its next time index (the next time the state of the order book changed), and the agent selects a new market. This process continues for a fixed number of steps, initially set to 3000, and then the environment/agent reset. The agent also has a position limit, i.e a $\bar{Q} > 0$ such that if $Q_t > \bar{Q}$, then the agent does not quote a bid price, and if $Q_t < -\bar{Q}$, then the agent does not quote an ask. This ensures that the agent does not accumulate an excessive position. For our purposes, $\bar{Q}$ was set to 100.

3.2.1 State Space

Since we are using real world price data, the state space is already discrete. The state of the system is described by the three-tuple $(\delta_t^b, \delta_t^a, Q_t)$, where $\delta^b = m_t - b_t$, $\delta_t^a = a_t - m_t$, and Q_t is the agent's position.

3.2.2 Action Space

In this environment, our actions will be quite simple. The action an agent can take will consist of the tuple (δ^b, δ^a), where δ^b is the difference between the mid-price and the agent's desired bid, and δ^a is the difference between the agent's desired ask and the mid-price.

3.2.3 Rewards

For a complex problem such as optimal market making, the reward function plays a crucial role in ensuring the agent performs the way in which we would expect. In this situation, the reward is a delicate balance between ensuring the agent keeps a fairly small position (i.e. does not go excessively long/short and makes a lot of money from changes in the asset price, as this is not the goal of a market maker), and actually makes trades and profit. This is done between a balance of position penalty, and a reward for increasing wealth. If the position penalty is too large, the agent will barely trade, and if it is too small, it may overfit and always go too long/short. The reward function that was used for training all of the agents is as follows,

$$R_t = W_t - W_{t-1} - \phi Q_t^2,$$

where W_t is the total wealth of the agent, Q_t is the inventory, and ϕ is the inventory penalty. The reason that the square of the position was used, is that we wish for the punishment for larger positions to be proportionally higher than for small positions. A practical reason for this is that if the agent was forced to liquidate its inventory at trading close for example, the cost of doing so would be proportionally higher for larger inventories, due to lower liquidity. Hence we punish the agent proportionally more for larger positions. We include the change in wealth in the reward signal, as an increase in wealth should be rewarded, but a decrease in wealth should be punished. If we sum up all rewards with $\gamma = 1$, then we retrieve the agent's total PnL (assuming $W_0 = 0$) minus some cumulative position penalty. In reality, we choose γ to be close to, but not equal to 1 for convergence reasons. In training, we set $\gamma = 0.999$. Hence for small enough timesteps, and large enough γ,

$$G_T \approx \text{PnL}_T - \phi \int_0^T Q_t^2 dt,$$

so the terminal discounted cumulative reward is approximately the agent's PnL, minus some penalty multiplied by the integral of the square of the agent's position over time.

3.3 Fitting Avellaneda-Stoikov to Market Data

In order to compare our reinforcement learning agents against some known strategy, we will fit an Avellaneda-Stoikov model to the data, and compare their performance to this model. In order to use this model, we must fit the model to the TSLA order book data. This involves finding estimates for the parameters σ, A, and k. We will call these estimates $\hat{\sigma}$, $\hat{A}$, and $\hat{k}$.

3.3.1 Estimating Volatility

We begin by finding $\hat{\sigma}$, an estimate for the volatility parameter in the Avellaneda-Stoikov market making model. Recall the proposed diffusion of the asset value from the model,

$$dS_t = \sigma dB_t.$$

From this, we have that

$$\begin{aligned} & S_t \sim S_0 + N(0, \sigma^2 t) \\ \implies & S_t - S_0 \sim \sqrt{t} N(0, \sigma^2) \\ \implies & \frac{S_t - S_0}{\sqrt{t}} \sim N(0, \sigma^2) \\ \implies & \mathbb{V}\left(\frac{S_t - S_0}{\sqrt{t}}\right) = \sigma^2. \end{aligned}$$

We can hence use the square root of the sample variance of $\frac{S_t - S_0}{\sqrt{t}}$ as an estimator for σ, i.e,

$$\hat{\sigma} = \sqrt{\frac{1}{N-1} \sum_{i=1}^{N} \left(\frac{S_{t_i} - S_0}{\sqrt{t_i}} - \overline{\frac{S_t - S_0}{\sqrt{t}}} \right)^2},$$

where $\overline{\frac{S_t - S_0}{\sqrt{t}}}$ is the mean over observations at times $t = t_1, ..., t_N$. Applying this to the data, we obtain that $\hat{\sigma} = 0.0612$.

3.3.2 Estimating Order Fill Probabilities

We now turn our attention to estimating A and k. Recall that in the Avellaneda-Stoikov market making model, the market maker's orders are filled with a Poisson rate $\lambda(\delta)$, where δ is the distance from the mid-price of the bid/ask quote. The functional form of λ was assumed to be

$$\lambda(\delta) = Ae^{-k\delta}. \tag{3.3.1}$$

To estimate these two parameters from the data, we firstly define some new notation. Firstly, suppose that the distance from the mid-price of each order can take values in $\delta_1, ..., \delta_n$. We let $x_j^{(i)}$ denote the fill time of the jth order which has distance δ_i from the mid-price. To estimate the parameters, for each spread δ_i, we calculate $\lambda_i := \lambda(\delta_i)$. Since the order fill times, $x_j^{(i)}$ are exponentially distributed, we use the maximum likelihood, $\widehat{\lambda_i}$ of λ_i. The maximum likelihood estimator for the parameter of an exponential distribution is given by

$$\widehat{\lambda_i} = \frac{N_i}{\sum_{j=1}^{N_i} x_j^{(i)}},$$

where in this setting, the $x_j^{(i)}$s are the fill times for the orders which have spread δ_i. We hence obtain values $\widehat{\lambda_1}, ..., \widehat{\lambda_n}$, an estimate of the rate of filling for each different spread level, δ_i. We must then fit these values to (3.3.1) to obtain approximate values for $\hat{k}$ and $\hat{A}$. To do this, we

will use a non-linear least squares algorithm, the Levenberg–Marquardt algorithm, as described by Gavin in [30, Page 3]. After performing the parameter fitting, we obtain estimates $\hat{A} = 15.32$ and $\hat{k} = 23.43$. Figure 3.1 shows how the fitted parameter values compare to the observed fitted rates, $\widehat{\lambda_i}$s.

Figure 3.1: Fitted order fill rates, versus observed rates, $\widehat{\lambda_i}$.

The fit shown is not perfect, but the general exponentially decaying shape can clearly be shown, and the fitted curve will be a good enough approximation of the fill probabilities. We will use the estimates $\hat{A}$, $\hat{k}$, and $\hat{\sigma}$ to create an optimal Avellaneda-Stoikov agent to compare against our three actor-critic agents, with the aim of observing superior performance from the actor-critic agents.

3.4 Results

The three agents were trained on 1.4 million environment interactions, and the best performing model was saved. The episodes of testing were 4,000 steps, each randomly chosen from a week's worth of order book data. Each of the three models were then tested on approximately five minutes of unseen data from a different week, and this test was performed 1,000 times to generate a distribution of quantities of interest. Five minute periods were chosen as this allows for testing on a variety of different market conditions; if the strategies are able to perform well on many different short periods of time, this translates well to long term performance.

3.4.1 Comparison to Avellaneda-Stoikov

We consider the agents' performance against the Avellaneda-Stoikov strategy, a well known, and well performing market making strategy. We begin by looking at a table of useful statistics to see how each algorithm performed on the five minute testing periods.

Statistic	Avellaneda	SAC	DDPG	PPO
Mean Wealth	\$136.63	\$107.95	\$143.84	\$120.49
Median Wealth	\$127.02	\$126.64	\$160.68	\$140.66
Wealth Standard Deviation	80.3	182.7	159.4	172.0

Table 3.1: Statistics for different algorithms on 5 minute intervals on TSLA data

Table 3.1 shows that all three reinforcement learning algorithms achieved a mean wealth of greater than zero, but had a fairly significant variance, with DDPG being the lowest. DDPG managed to achieve a higher mean and median terminal wealth than the Avellaneda-Stoikov strategy, which in itself is a promising result; this shows that actor-critic approaches may have potential to outperform well known market making strategies. We now perform Mood's median test, a test to compare whether the medians of two samples are different, and we compare each reinforcement learning algorithm against the Avellaneda-Stoikov strategy. Table 3.2 shows the statistics and p-values of each of the tests.

	SAC	PPO	DDPG
χ-squared statistic	0.02	5.62	27.4
p-value	0.964	0.018*	1.67e-7*

Table 3.2: Mood's median test for different median than Avellaneda-Stoikov

Table 3.2 shows that both PPO and DDPG have a statistically significant higher median wealth than Avellaneda-Stoikov at the 5% level, however SAC does not produce a higher median. From these statistics, combined with Table 3.1, we conclude that DDPG outclasses the other two algorithms in this environment (in terms of both accumulated wealth, and variance of wealth), and also performs better than standard market making strategies in some metrics. From this point on, we consider only DDPG.

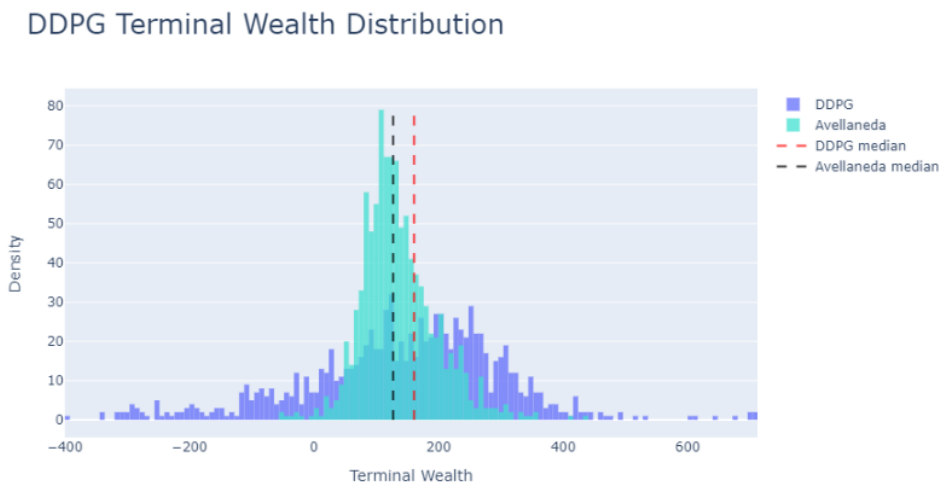

Figure 3.2: Terminal wealth distributions of DDPG vs Avellaneda-Stoikov on TSLA data.

Figure 3.2 backs up the results shown in Table 3.1, i.e. that the Avellaneda-Stoikov strategy has a lower mean/median than DDPG, but also a lower variance. It seems that Avellaneda-Stoikov is less risky than DDPG, however has less potential to make large profits. There are two possible ways the variance of the DDPG algorithm could be reduced,

- Additional interaction with the environment (training steps).

- Variance reduction for policy gradient methods, such as those described by Kaledin *et al.* [31].

3.4.2 DDPG Training Metrics

In order to see if the DDPG agent has actually learned generalised information about the environment, we can consider different metrics that were recorded during the training process. Firstly, we can look at the mean reward generated during the training. If the agent learns attributes of its environment, we would expect the mean reward signal to increase over time. Figure 3.3 shows the mean reward over the course of the training.

Figure 3.3: DDPG mean reward during training.

From Figure 3.3, we see that the agent does in general increase its mean reward over the course of the training. The speed of increase does seem to plateau towards the end, however a longer training period may be needed to see if this is a true trend. We can also consider the value of the critic loss during training. Critic loss gives an indication of the convergence of the critic, i.e. the estimates of the value function. Figure 3.4 shows the critic loss over the course of the training.

Figure 3.4: DDPG critic loss during training.

Figure 3.4 shows that initially, as the agent is learning and exploring the environment, we see that the critic loss increases, but then after around 100,000 interactions with the environment, the critic begins to learn the values of states, and the loss decreases over the rest of the training period. This is a good indicator that the agent is learning information about its environment through successive interactions.

3.4.3 DDPG Statistics

First, we will the qualitative properties of the strategy. Figure 3.5 shows the wealth progression of the DDPG agent over some sample realisations of the test data.

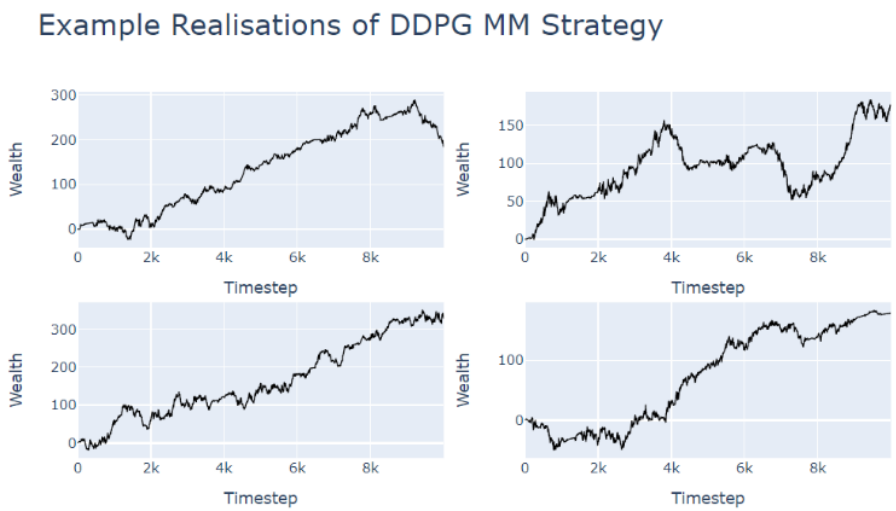

Figure 3.5: Four realisations of DDPG strategy on TSLA data

Figure 3.5 shows that the DDPG strategy seems to make consistent profit, with a general upwards trend in wealth, however there are periods where the strategy does have drops in wealth in the short term.

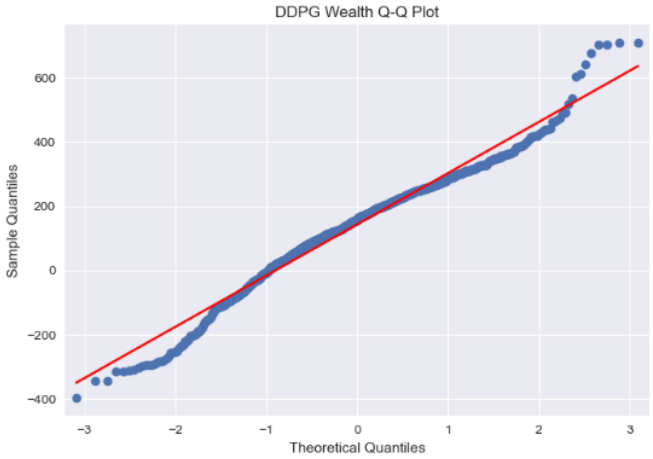

Figure 3.6: Q-Q Plot of DDPG terminal wealth distribution

Figure 3.6 shows that the DDPG wealth distribution does not strictly follow a normal distri-

bution, and may be heavy tailed, meaning that extreme profits/losses have a higher probability of occurring than if the wealth was normally distributed.

3.4.4 Summary of Results

Overall, we have found that one algorithm in particular, DDPG was able to learn information about its environment through interaction (Figures 3.3 and 3.4), leading to a higher mean and median wealth than the Avellaneda-Stoikov strategy. The variance of the DDPG strategy was indeed higher than the Avellaneda-Stoikov strategy, but this variance is a problem that will require further investigation through variance reduction methods, and hyperparameter tuning. However the results achieved so far are a promising starting point for future investigation into the effectiveness of actor-critic algorithms for optimal market making.

A Note on Assumptions

During the testing of these reinforcement learning agents, it is important to note that there have been some assumptions and simplifications that have been made out of both necessity, and ease of result collection. The notable assumptions of this environment are as follows.

- Zero transaction costs on trading. In reality, when trading in small lot sizes as outlined here, transaction costs should be negligible. Also as mentioned in Section 3.2, some exchanges may even provide some rebate for providing liquidity.
- Priority of limit orders when quoting on best bid/ask. Due to the fact that we are not able to determine which orders have been traded against in the limit order book, we assume that if our agent is quoting on the best bid/ask level, their order is matched if a market order is placed. This assumption essentially says that our agent has faster execution speed than other market participants, and may at times lead to small errors in results.
- Zero slippage. We assume that our agent will always trade at the price that is observed in the limit order book at a given time. Since most of the agent's trades are its own passive limit orders being hit, slippage should not significantly affect the results.

Conclusion

Reinforcement learning is clearly of growing important in financial problems. We have demonstrated in Chapter 2 that reinforcement learning, (more specifically actor-critic methods) can learn and outperform well known market making models on a simulated theoretical environment. The Advantage Actor-Critic algorithm outperformed traditional Q-learning techniques, and achieved a higher mean wealth than the mathematical model, Avellaneda-Stoikov.

In Chapter 3, we then moved on to testing actor-critic approaches to real limit order book data, specifically TSLA. We set up a reinforcement learning market making environment by looking at historical data, and also adding in artificial trades if the agent posts a bid/ask price that is more competitive than the historical best bid/ask price. We also provide the agent a small rebate for receiving trades, similar to how real exchanges will often provide rebates to large market making firms for providing liquidity to the market. It is hard to fully use historical data to test market making strategies, as it is hard to capture the market impact of an agent. We hence assumed that the market maker quoted low volumes to minimise this.

The three actor-critic algorithms that were trained on this environment were Soft Actor-Critic, Deep Deterministic Policy Gradient, and Proximal Policy Optimisation. From the testing results, we observed that Soft Actor-Critic performed the worst on the unseen data, with a higher variance and lower mean wealth than all of the other agents. Proximal Policy Optimisation was the next best, and the best performing algorithm on this environment was Deep Deterministic Policy Gradient. We find that this algorithm was also able to outperform the Avellaneda-Stoikov strategy, achieving a mean wealth of \$143.84, vs the Avellaneda-Stoikov strategy's wealth of \$136.63. The variance of the wealth of the actor-critic strategies were all higher than that of the Avellaneda-Stoikov strategy. In future research, further investigation could be done into variance reduction methods, or with more computational power, we could train the agent on significantly more environment steps to observe if underfitting was the cause of high variance. There is also room for investigating hyperparameter tuning, as there are many hyperparameters in the actor-critic agents that could be tuned to further improve the agent's performance. Finally, there may be room for improvement in the construction of the reward signal; there are many other options for incorporating the position penalty into the signal, for example using the absolute value of the inventory, rather than its square. With more time and computational power, these improvements would be simple to implement, and could lead to an even better market making strategy.

Appendix A

Auxiliary Proofs

A.1 Proof of Dynamic Programming Principle

$$H(s, X_s) = \sup_{\pi \in \mathcal{A}_{s,t}} \mathbb{E}[H(t, X_t^\pi) | \mathcal{F}_s] \qquad \text{for all } s \leq t \leq T$$

Proof. Let π^0 be an arbitrary Markov control. Then, for all $s < t$,

$$\begin{aligned}
J^{\pi^0}(s, X_s) =& \mathbb{E}\left[\psi(X_T^{\pi^0})\right] && \text{(by definition)} \\
=& \mathbb{E}\left[\mathbb{E}\left[\psi(X_T^{\pi^0})\middle|\mathcal{F}_t\right]\middle|\mathcal{F}_s\right] && \text{(by tower property)} \\
=& \mathbb{E}\left[J^{\pi^0}(t, X_t^{\pi^0})\middle|\mathcal{F}_s\right] \\
\implies J^{\pi^0}(s, X_s) \leq& \mathbb{E}\left[\sup_{\pi \in \mathcal{A}_{t,T}} J^\pi(t, X_t^{\pi^0})\middle|\mathcal{F}_s\right] \\
=& \mathbb{E}\left[H(t, X_t^{\pi^0}) \mid \mathcal{F}_s\right] && \text{(by definition of } H\text{)} \\
\implies \sup_{\pi \in \mathcal{A}_{s,T}} J^\pi(s, X_s) \leq& \sup_{\pi \in \mathcal{A}_{s,T}} \mathbb{E}\left[H(t, X_t^{\pi^0}) \mid \mathcal{F}_s\right] \\
\implies H(s, X_s) \leq& \sup_{\pi \in \mathcal{A}_{s,T}} \mathbb{E}\left[H(t, X_t^{\pi^0}) \mid \mathcal{F}_s\right] && \text{(by definition of } H\text{)} \\
\implies H(s, X_s) \leq& \sup_{\pi \in \mathcal{A}_{s,t}} \mathbb{E}\left[H(t, X_t^{\pi^0}) \mid \mathcal{F}_s\right], && \text{(A.1.1)}
\end{aligned}$$

Where the last line to derive (A.1.1) is true as the right hand side of the previous line has only a dependence on t. We now seek to bound $H(s, X_s)$ below by $\sup_{\pi \in \mathcal{A}_{s,t}} \mathbb{E}\left[H(t, X_t^{\pi^0}) \mid \mathcal{F}_s\right]$ to prove the theorem. To do this, recall that

$$J^{\pi^0}(s, X_s) = \mathbb{E}\left[J^{\pi^0}(t, X_t^{\pi^0})\middle|\mathcal{F}_s\right].$$

Now let $\hat{\pi}$ be an ε-optimal control, where $\varepsilon > 0$ is arbitrary. i.e. we have

$$H(t, x) \geq J^{\hat{\pi}}(t, x) \geq H(t, x) - \varepsilon.$$

We now define a new Markov control, $\tilde{\pi}$ by

$$\tilde{\pi}_u := \begin{cases} \hat{\pi}_u & \text{if } u \geq t \\ \pi_u^0 & \text{if } u < t \end{cases}.$$

We then have that

$$\begin{aligned}
J^{\tilde{\pi}} =& \mathbb{E}\left[J^{\tilde{\pi}}(t, X_t^{\tilde{\pi}}) \mid \mathcal{F}_s\right] \\
\implies J^{\tilde{\pi}} =& \mathbb{E}\left[J^{\hat{\pi}}(t, X_t^{\tilde{\pi}}) \mid \mathcal{F}_s\right] && \text{(as } J^{\tilde{\pi}}(t,\cdot) \text{ only depends on } \tilde{\pi} \text{ after } t\text{)} \\
\implies J^{\tilde{\pi}} \geq& \mathbb{E}\left[H(t, X_t^{\tilde{\pi}}) \mid \mathcal{F}_s\right] - \varepsilon && \text{(as } \hat{\pi} \text{ is } \varepsilon\text{-optimal)} \\
\implies \sup_{\pi \in \mathcal{A}_{s,T}} J^{\tilde{\pi}} \geq& \mathbb{E}\left[H(t, X_t^{\tilde{\pi}}) \mid \mathcal{F}_s\right] - \varepsilon \\
\implies H(s, X_s) \geq& \mathbb{E}\left[H(t, X_t^{\tilde{\pi}}) \mid \mathcal{F}_s\right] - \varepsilon \\
\implies H(s, X_s) \geq& \mathbb{E}\left[H(t, X_t^{\pi^0}) \mid \mathcal{F}_s\right] - \varepsilon && \text{(as } X_t^{\tilde{\pi}} \text{ only depends on } \tilde{\pi} \text{ from time } s \text{ to } t\text{)} \\
\implies H(s, X_s) \geq& \sup_{\pi \in \mathcal{A}_{s,t}} \mathbb{E}\left[H(t, X_t^{\pi}) \mid \mathcal{F}_s\right] - \varepsilon \\
\implies H(s, X_s) \geq& \sup_{\pi \in \mathcal{A}_{s,t}} \mathbb{E}\left[H(t, X_t^{\pi}) \mid \mathcal{F}_s\right]. && \text{(A.1.2)}
\end{aligned}$$

If we combine (A.1.2) and (A.1.1), we have that

$$\sup_{\pi \in \mathcal{A}_{s,t}} \mathbb{E}\left[H(t, X_t^{\pi}) \mid \mathcal{F}_s\right] \geq H(s, X_s) \geq \sup_{\pi \in \mathcal{A}_{s,t}} \mathbb{E}\left[H(t, X_t^{\pi}) \mid \mathcal{F}_s\right],$$

so that

$$H(s, X_s) = \sup_{\pi \in \mathcal{A}_{s,t}} \mathbb{E}\left[H(t, X_t^{\pi}) \mid \mathcal{F}_s\right],$$

as required. □

A.2 Proof of Hamilton-Jacobi-Bellman Equation

Theorem A.2.1 (Hamilton-Jacobi-Bellman Equation)**.** *Let X^π be a controlled diffusion satisfying*

$$dX_t^\pi = \mu(t, X_t^\pi, \pi_t)dt + \sigma(t, X_t^\pi, \pi_t)dB_t$$

Let $H : [0,T] \times \mathbb{R} \to \mathbb{R}$ be a function satisfying

$$H(t, X_t) = \sup_{\pi \in \mathcal{A}_{t,T}} \mathbb{E}(\varphi(X_T^\pi) | \mathcal{F}_t)$$

Assume $H \in \mathcal{C}^{1,2}$, then:

$$\sup_{\pi \in \mathbb{R}} \left\{ \frac{\partial H}{\partial t} + \mu(t,x,\pi)\frac{\partial H}{\partial x} + \frac{1}{2}\sigma^2(t,x,\pi)\frac{\partial^2 H}{\partial x^2} \right\} = 0$$

$$H(T, x) = \varphi(x)$$

Additionally, suppose $\pi^* : [0,T] \times \mathbb{R} \to \mathbb{R}$ attains the supremum in (1.4.6) *for all t, x. Then π^* is an optimal Markov control.*

Proof. We first introduce the differential operator $\mathcal{L}^\pi = \mu^\pi \partial_x + \frac{1}{2}(\sigma^\pi)^2 \partial_{xx}$. Let π be an arbitrary Markov control, and let $Y_t = H(t, X_t^\pi)$. Then

$$dY_t = [\partial_t H(t, X_t^\pi) + \mathcal{L}^\pi H(t, X_t^\pi)]dt + \sigma(t, X_t^\pi, \pi(t, X_t^\pi))\partial_x H(t, X_t^\pi)dB_t.$$

If we now consider some small timestep h, we have that the increment $Y_{t+h} - Y_t$ is given by

$$Y_{t+h} - Y_t = \int_t^{t+h} [\partial_t H(s, X_s^\pi) + \mathcal{L}^\pi H(s, X_s^\pi)]ds + \int_t^{t+h} \sigma(s, X_s^\pi, \pi(s, X_s^\pi))\partial_x H(s, X_s^\pi)dB_s$$

$$\implies \mathbb{E}[Y_{t+h} \mid \mathcal{F}_t] - Y_t = \mathbb{E}\left[\int_t^{t+h} [\partial_t H(s, X_s^\pi) + \mathcal{L}^\pi H(s, X_s^\pi)]ds + \int_t^{t+h} \sigma(s, X_s^\pi, \pi(s, X_s^\pi))\partial_x H(s, X_s^\pi)dB_s \,\middle|\, \mathcal{F}_t \right].$$

We know that $\mathbb{E}[\int_t^{t+h} dB_s \,|\mathcal{F}_t] = 0$, so under some technical conditions, the term on the right hand side of the expectation is zero. Hence,

$$\mathbb{E}[Y_{t+h} \mid \mathcal{F}_t] - Y_t = \mathbb{E}\left[\int_t^{t+h} [\partial_t H(s, X_s^\pi) + \mathcal{L}^\pi H(s, X_s^\pi)]ds \;\middle|\; \mathcal{F}_t\right].$$

$$\implies \mathbb{E}[H(t+h, X_{t+h}^\pi) \mid \mathcal{F}_t] - H(t, X_t) = \mathbb{E}\left[\int_t^{t+h} [\partial_t H(s, X_s^\pi) + \mathcal{L}^\pi H(s, X_s^\pi)]ds \;\middle|\; \mathcal{F}_t\right].$$

Recall that

$$H(t, X_t) := \sup_{\pi \in \mathcal{A}_{t,s}} \mathbb{E}[H(s, X_s^\pi) \mid \mathcal{F}_t] \qquad \text{for all } s \geq t. \tag{A.2.1}$$

Using this, we know that $\mathbb{E}[H(t+h, X_{t+h}^\pi) \mid \mathcal{F}_t] \leq H(t, X_t)$. Combining this with (A.2.1), we obtain

$$0 \geq \mathbb{E}\left[\int_t^{t+h} [\partial_t H(s, X_s^\pi) + \mathcal{L}^\pi H(s, X_s^\pi)]ds \;\middle|\; \mathcal{F}_t\right]$$

$$\implies 0 \geq \mathbb{E}\left[\frac{1}{h}\int_t^{t+h} [\partial_t H(s, X_s^\pi) + \mathcal{L}^\pi H(s, X_s^\pi)]ds \;\middle|\; \mathcal{F}_t\right]. \tag{A.2.2}$$

We now take this limit as $h \to 0^+$. By the fundamental theorem of calculus applied to (A.2.2), we obtain

$$0 \geq \partial_t H(s, X_s^\pi) + \mathcal{L}^\pi H(s, X_s^\pi).$$

Since this partial differential inequality holds for all π, s, it must hold for any value X_s^π that can be attained by the process. We hence replace X_s^π with some state variable x, and obtain

$$0 \geq \partial_t H(t,x) + \mu(t,x,\pi)\partial_x H(t,x) + \frac{1}{2}\sigma^2(t,x,\pi)\partial_{xx} H(t,x). \tag{A.2.3}$$

We now wish to bound the right hand side of (A.2.3) from below. Firstly, Let $\hat{\pi}$ be εh optimal from time t to $t+h$, for some $\varepsilon > 0$, $h > 0$, i.e.

$$H(t, X_t) \geq \mathbb{E}[H(t+h, X_{t+h}^{\hat{\pi}}) \mid \mathcal{F}_t] \geq H(t, X_t) - \varepsilon h.$$

Essentially, the performance of $\hat{\pi}$ is very close to H. We now proceed as before, by letting $Y_t = H(t, X_t^{\hat{\pi}})$. By performing exactly the same steps as above, we obtain

$$\mathbb{E}[H(t+h, X_{t+h}^{\hat{\pi}}) \mid \mathcal{F}_t] - H(t, X_t) = \mathbb{E}\left[\int_t^{t+h} [\partial_t H(s, X_s^{\hat{\pi}}) + \mathcal{L}^{\hat{\pi}} H(s, X_s^{\hat{\pi}})]ds \;\middle|\; \mathcal{F}_t\right].$$

$$\implies -\varepsilon h \leq \mathbb{E}\left[\int_t^{t+h} [\partial_t H(s, X_s^{\hat{\pi}}) + \mathcal{L}^{\hat{\pi}} H(s, X_s^{\hat{\pi}})]ds \;\middle|\; \mathcal{F}_t\right].$$

$$\implies -\varepsilon \leq \mathbb{E}\left[\frac{1}{h}\int_t^{t+h} [\partial_t H(s, X_s^{\hat{\pi}}) + \mathcal{L}^{\hat{\pi}} H(s, X_s^{\hat{\pi}})]ds \;\middle|\; \mathcal{F}_t\right].$$

Again, taking the limit as $h \to 0^+$, we obtain

$$-\varepsilon \leq \partial_t H(t, X_t^{\hat{\pi}}) + \mathcal{L}^{\hat{\pi}} H(t, X_t^{\hat{\pi}})$$

$$\implies -\varepsilon \leq \partial_t H(t,x) + \mu(t,x,\pi)\partial_x H(t,x) + \frac{1}{2}\sigma^2(t,x,\pi)\partial_{xx} H(t,x). \tag{A.2.4}$$

Combining (A.2.3) and (A.2.4), and recalling that $\varepsilon > 0$ was arbitrary, we obtain

$$\sup_{\pi \in \mathbb{R}} \left\{ \frac{\partial H}{\partial t} + \mu(t,x,\pi)\frac{\partial H}{\partial x} + \frac{1}{2}\sigma^2(t,x,\pi)\frac{\partial^2 H}{\partial x^2} \right\} = 0$$

$$H(T,x) = \varphi(x),$$

as required. □

Bibliography

[1] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

[2] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

[3] Mikko Pakkanen. Market microstructure, March 2022.

[4] Helen Allen, John Hawkins, and Setsuya Sato. Electronic trading and its implications for financial systems. *Bank of International Settlements*, 7, 01 2001.

[5] Leo Smigel. Global algorithmic trading market to surpass $21,685.53 million by 2026. `https://www.businesswire.com/news/home/20190205005634/en/Global-Algorithmic-Trading-Market-to-Surpass-US-21685.53-Million-by-2026`, Feb 2019. Accessed: 2022-09-01.

[6] Matthew F. Dixon, Igor Halperin, and Paul Bilokon. *Machine Learning in Finance, From Theory to Practice*. Springer Cham, first edition, 2020.

[7] Bruno Gašperov, Stjepan Begušić, Petra Posedel Šimović, and Zvonko Kostanjčar. Reinforcement learning approaches to optimal market making. *Mathematics*, 9(21), 2021.

[8] Matias Selser, Javier Kreiner, and Manuel Maurette. Optimal market making by reinforcement learning, 2021.

[9] YS Lim and D Gorse. Reinforcement learning for high-frequency market making. 04 2018.

[10] Alexey Bakshaev. Market-making with reinforcement-learning (sac), 2020.

[11] Koundinya Vajjha, Avraham Shinnar, Barry Trager, Vasily Pestun, and Nathan Fulton. Certrl: Formalizing convergence proofs for value and policy iteration in coq. CPP 2021, page 18–31, New York, NY, USA, 2021. Association for Computing Machinery.

[12] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1928–1937, New York, New York, USA, 20–22 Jun 2016. PMLR.

[13] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *CoRR*, abs/1801.01290, 2018.

[14] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015.

[15] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization, 2015.

[16] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *ICLR (Poster)*, 2016.

[17] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In Eric P. Xing and Tony Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 387–395, Bejing, China, 22–24 Jun 2014. PMLR.

[18] Á. Cartea, S. Jaimungal, and J. Penalva. *Algorithmic and High-Frequency Trading*. Cambridge University Press, 2015.

[19] Robert C Merton. Optimum consumption and portfolio rules in a continuous-time model. *Journal of Economic Theory*, 3(4):373–413, 1971.

[20] US Securities and Exchange Commission. Market centers: Buying and selling stock. `https://www.sec.gov/fast-answers/answersmarket`, Oct 2012. Accessed: 2022-07-25.

[21] Steven Poser. Market makers in financial markets: Their role, how they function, why they are important, and the nyse dmm difference. 2021.

[22] Jean-Philippe Bouchaud, Julius Bonart, Jonathan Donier, and Martin Gould. *Trades, Quotes and Prices: Financial Markets Under the Microscope*. Cambridge University Press, 2018.

[23] Marco Avellaneda and Sasha Stoikov. High frequency trading in a limit order book. *Quantitative Finance*, 8:217–224, 04 2008.

[24] Philipp Weber and Bernd Rosenow. Order book approach to price impact, 2003.

[25] Sergei Maslov and Mark Mills. Price fluctuations from the order book perspective—empirical facts and a simple model. *Physica A: Statistical Mechanics and its Applications*, 299(1-2):234–246, oct 2001.

[26] Marc Potters and Jean-Philippe Bouchaud. More statistical properties of order books and price impact. *Physica A: Statistical Mechanics and its Applications*, 324(1-2):133–140, jun 2003.

[27] Olivier Guéant, Charles-Albert Lehalle, and Joaquin Fernandez-Tapia. Dealing with the inventory risk: a solution to the market making problem. *Mathematics and Financial Economics*, 7(4):477–507, sep 2012.

[28] Gordon Ritter. Machine learning for trading. *SSRN Electronic Journal*, 01 2017.

[29] Charles-Albert Lehalle, Olivier Guéant, and Julien Razafinimanana. *High-Frequency Simulations of an Order Book: a Two-scale Approach*, pages 73–92. Springer Milan, Milano, 2011.

[30] Henri P. Gavin. The levenberg-marquardt method for nonlinear least squares curve-fitting problems. 2013.

[31] Maxim Kaledin, Alexander Golubev, and Denis Belomestny. Variance reduction for policy-gradient methods via empirical variance minimization, 2022.