The Merton Problem using model free Reinforcement Learning

John Goodacre
Supervisor: Professor Jun Wang

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Master of Research

in

Financial Computing.

Department of Computer Science

University College London

July 19, 2018

I, John Goodacre, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the work.

Code for this project, can be found in the following public repository:

https://github.com/jsg71/MRes-Thesis

Abstract

An important problem within Finance is the issue of how to allocate one's wealth across assets. The portfolio allocation problem remains on-going within the investment arena, with major contributions dating back to Nobel prize winners such as Markowitz [19], Sharpe and Merton [20]. The simpler setting for asset allocation is the single period one, where actions are myopic, that is take the best action for that particular time period and repeat. Merton's problem is more sophisticated due to its taking into account all future time periods at every step. The goal being to maximise the expected utility of terminal wealth at some distant time-step T.

This dissertation lies at the crossroads of finance and machine learning. In particular I use techniques from Reinforcement Learning (RL) and apply them to the Merton problem, specifically I attempt to train model free reinforcement learners in various settings to maximise the expected utility of wealth over some future time period.

Merton's famed work [20] provides an analytical solution to the multi-period investment problem under certain restrictive assumptions and given prior knowledge of the risky process and its parameters. Given this prior knowledge Merton used mathematical techniques from stochastic optimal control to derive a solution.

The RL agent being model free will have no such prior knowledge. It knows nothing of the stochastic process or its parameters, it simply takes actions (random at first) and by receiving rewards attempts to learn how to trade, with performance approximating Merton optimality. Merton's solution not only requires prior knowl-

edge of the process, and its parameters, but also different utility curves require different mathematical derivations, indeed extensions such as adding in transactions costs would be different again and more sophisticated stochastic processes than say Geometric Brownian motion may prove to be intractable in terms of an analytic solution. In this work I first start with log-utility where I derive a very simple exact reward for the RL agent to learn from, I then generalise to other utility curves using a parameterised mean variance approximation used by Ritter [25] for trading an Ornstein Uhlenbeck process. Finally I introduce greater realism by increasing the volatility of the stochastic process to market levels and adding fat tails, where the RL agent begins to come into his own and outperform the Merton agent.

Key issues in this thesis concerned shaping rewards suitable for the RL agent (we are not attemping to maximise expected wealth, but its utility, thus risk aversion is also embedded) and also the fact that within finance rewards are noisy. The noise means that even in comparing a merton optimal agent to a random one one needed many episodes in order to differentiate skill from luck and thus informed my test statistic where I examined the distributions of test values (not just expected performance). Recent successes in RL tend to have come via using function approximators such as neural nets, to address high dimensional state spaces, however usually with a stable environment and little noise. This problem setting is the opposite, dimensionality is relatively easy to handle, but overcoming the noise requires many episodes of training and indeed the agent has almost no control of his environment being a price taker and can only hope to learn to take actions that positively influence his future distribution of wealth.

The RL agents despite being relatively straightforward did indeed learn to approach the performance of the merton optimal solution in a model free way. Indeed when the environment became more realistic and exhibited high volatility and fat tails, the RL agent performed as well as Merton optimal with less variability in performance.

Acknowledgements

Many thanks to Professor Jun Wang for his guidance within this project. Not least for encouraging me to look at Merton's problem in the context of Reinforcement Learning in the first place.

Contents

1 Introduction **15**

1.1 Motivation 15

1.2 Structure of the thesis 16

1.3 Literature Review 18

1.3.1 Finance 18

1.3.2 Machine Learning 19

1.3.3 Reinforcement Learning applied to Finance 20

2 Finance Background **22**

2.1 Portfolio Theory in brief 22

2.2 Data representations 24

2.2.1 Risk-free Bond 24

2.2.2 Geometric Brownian Motion 25

2.2.3 Fat Tails 26

2.3 Risk Neutrality and Utility Curves 27

2.3.1 Utility Curve Intuitions 28

2.3.2 Mean Variance Utility 29

2.4 Merton's Problem 30

2.4.1 Merton solution - Logarithmic Utility 30

2.4.2 Merton Solution - Power Utility 33

3 Machine Learning Background **35**

3.1 Reinforcement Learning 35

3.1.1 The Reinforcement Learning setting 35
3.1.2 Markov Decision Processes 37
3.1.3 The Bellman Expectation equation 40
3.1.4 The Bellman Optimality equation 42
3.1.5 Learning algorithms - From Dynamic Programming to Dyna 42
3.2 Advanced Reinforcement Learning 49
3.2.1 Function Approximation 49
3.2.2 Deep Learning 49
3.2.3 Deep Q and Double Q Learning 49
3.2.4 Prioritised experience replay and Noisy Networks 49

4 Implementation **50**
4.1 Reward Shaping 50
4.1.1 Discretisation - terminal utilities and step-wise rewards . . . 51
4.1.2 Mean Variance Equivalent Distributions, Episodic 53
4.1.3 Mean-Variance equivalent distributions, Step-wise 54
4.1.4 Reward Shaping Conclusion 57
4.2 Design 57
4.2.1 Market Environment 58
4.2.2 Agent 59
4.2.3 Noise intuitions - Merton and Random agents episodically . 60
4.3 Test Metrics 62
4.3.1 Utilities 62
4.3.2 Rewards 63
4.3.3 Wealth 63
4.3.4 Sharpe Ratio 64

5 Experiments **65**
5.1 Experiment group 1: Learning to trade like Merton, Double-Q . . . 66
5.1.1 Log-Utility 66
5.1.2 Increasing the Brownian Motion volatility 71

5.1.3 Generalising to Mean-Variance rewards 76
5.1.4 Power Utility 79
5.1.5 Learning risk aversion κ 82
5.2 Experiment group 2: Adding Realism 87
5.2.1 Realistic (S&P) volatility and risk free levels 87
5.2.2 Fat tails 93
5.3 Experiment group 3: Adding power, alternative RL agents 98
5.3.1 Tabular agents Double Sarsa and Dyna 98
5.3.2 Deep-Q learning 103
5.3.3 Double Deep-Q learning 108
5.3.4 Noisy Net Deep-Q Networks 110

6 General Conclusions **113**
6.1 Main Conclusions 113
6.2 Further Directions 115

Appendices **118**

7 Colophon **118**

Bibliography **119**

List of Figures

2.1 Efficient Frontier 24
2.2 Student-T pdfs - Fat Tail test 27

3.1 Basic RL framework 36
3.2 Dyna conceptual model 48

4.1 Low Volatility Geometric Brownian Motion Paths 51
4.2 Discretised Log Utility versus Step-wise rewards 52
4.3 MV equivalent approx to Log-Utility, $\kappa = 0.006$ 54
4.4 Mean Variance equivalent approximation to log-utility 56
4.5 $\kappa = 0$, does this match risk neutrality? 57
4.6 Episodic distribution of final utilities and rewards, Random v Merton 61
4.7 Stepwise rewards Merton optimal versus a Random agent 62
4.8 Batched utilities and rewards, random agent versus Merton 63

5.1 Double-Q learning: Utilities verses Merton and Random agent . . . 69
5.2 Moving average utilities Double-Q v Merton v Random 70
5.3 Medium Volatility Geometric Brownian Motion Paths 72
5.4 Discretised Log Utility versus Step-wise rewards - medium vol . . . 72
5.5 Distribution of batch Utilities Merton optimal v Random - medium vol 73
5.6 Double-Q learning: Utilities verses Merton -Medium Volatility . . . 74
5.7 Moving average utilities Double-Q v Merton - medium volatility . . 75
5.8 Double-Q learning: Utilities, MV-equivalent 77
5.9 Moving average utilities Double-Q, MV-equivalent 78

5.10 Double-Q learning: Utilities verses Merton, power utility 80
5.11 Moving average utilities Double-Q v Merton, power utility 81
5.12 $\kappa = 0$, Risk neutrality 82
5.13 Double-Q learning: Wealth Distribution v Merton, Risk Neutral . . 83
5.14 Double-Q learning: Wealth Distribution v Merton, Risk Aversion . . 85
5.15 Double-Q learning: Utility curve for $\kappa = 0.13$, Extreme Risk Aversion 86
5.16 Double-Q learning: Wealth Distribution v Merton, Extreme Risk Aversion $\kappa = 0.13$ 87
5.17 GBM sample paths, SP500 volatility levels 88
5.18 $\sigma = 0.2$, Log-utility, End episode utilities v Step-wise rewards $\kappa = 0.006$ 89
5.19 Batched Utilities and Reward distributions - Random v Merton Optimal 90
5.20 Q learning: Utilities verses Merton, $\sigma = 0.2$ 91
5.21 Moving average utilities Q v Merton, $\sigma = 0.2$ 92
5.22 Sample fat tail paths/ Episodic distribution, Random agent v Merton 93
5.23 $\sigma = 0.35$ fat-tails, Log-utility, End episode utilities v Step-wise rewards $\kappa = 0.006$ 94
5.24 Q learning: Utilities verses Merton, $\sigma = 0.35$, fat tails 95
5.25 Moving average utilities Q v Merton, $\sigma = 0.35$, fat tails 96
5.26 Double Sarsa learning: Utilities verses Merton, $\sigma = 0.35$, fat tails . 99
5.27 Moving average utilities Double Sarsa v Merton, $\sigma = 0.35$, fat tails 99
5.28 Dyna with experience replay: Utilities verses Merton, $\sigma = 0.35$, fat tails 100
5.29 Moving average utilities Dyna with experience replay v Merton, $\sigma = 0.35$, fat tails 101
5.30 Sample paths and log-utilities - constant investment, $\sigma = 0.20$ . . . 103
5.31 Final training rewards and loss function training DQN 104
5.32 Deep Q learning: Utilities verses Merton, $\sigma = 0.20$ 104
5.33 Moving average utilities Deep Q v Merton, $\sigma = 0.20$ 105

5.34 DQN investment choices for a variety of states 106
5.35 Double Deep Q learning: Utilities verses Merton, $\sigma = 0.20$ 108
5.36 Moving average utilities Deep Q v Merton, $\sigma = 0.20$ 109
5.37 Final training rewards and loss function training DQN 110
5.38 Noisy Deep Q learning: Utilities verses Merton, $\sigma = 0.20$ 111
5.39 Moving average utilities Noisy Deep Q v Merton, $\sigma = 0.20$ 112

List of Tables

5.1 Double-Q, test performance v Merton - Mean 70
5.2 Double-Q, test performance v Merton - Std 70
5.3 Double-Q, test performance v Merton - Mean, Medium vol 75
5.4 Double-Q, test performance v Merton - Std, Medium vol 75
5.5 Double-Q, MV equivalent, test performance - Mean 78
5.6 Double-Q, MV equivalent, test performance - Std 78
5.7 Double-Q, test performance v Merton - Mean, power utility 81
5.8 Double-Q, test performance v Merton - Std, power utility 81
5.9 Double-Q, Risk Neutral agent, test performance - Mean 83
5.10 Double-Q, Risk Neutral agent, test performance - Std 84
5.11 Double-Q, Risk averse agent, test performance - Mean 85
5.12 Double-Q, Risk averse agent, test performance - Std 85
5.13 Q, test performance v Merton - Mean 92
5.14 Q, test performance v Merton - Std 92
5.15 Q, test performance v Merton - Mean, fat tails 96
5.16 Q, test performance v Merton - Std, fat tails 96
5.17 Dyna-experience replay, test performance v Merton - Mean, fat tails 101
5.18 Dyna-experience replay, test performance v Merton - Std, fat tails . 101
5.19 Deep Q model specification 103
5.20 Deep Q, test performance v Merton - Mean 105
5.21 DeepQ, test performance v Merton 105
5.22 Double Deep Q, test performance v Merton - Mean 109
5.23 DeepQ, test performance v Merton 109

5.24 Noisy Deep Q, test performance v Merton - Mean 112
5.25 Noisy DeepQ, test performance v Merton 112

Chapter 1

Introduction

In this chapter I give a brief introduction to the problem being discussed, and motivation behind it. Then I lay out the specific questions being asked and finally, we outline the structure of the thesis.

1.1 Motivation

The problem of how to allocate between risky assets through time is highly non-trivial. Merton's famed work [20] derives a solution for maximising the expected utility of future wealth $\mathbf{E}[U(w_T)]$ by allocating between a stochastic process (a geometric brownian motion with a known volatility and drift) and a risk-free bond, given a known utility function U. The Merton problem has analytical solutions for only a few stochastic processes, and requires both considerable mathematical machinery and prior knowledge of the processes and parameters. It should be mentioned that his work goes further by also introducing optimal consumption, however within this thesis our viewpoint is that of a portfolio manager who wishes solely to maximise the expected utility of future wealth.

The motivation behind this thesis is an exploration of the Merton problem using techniques from reinforcement learning, which if possible provide certain advantages. In this setting we have an agent who knows nothing of the stochastic process, the risk free rate or any parameters. The agent simply finds itself in a state of the market and has a choice, 'What proportion of my wealth should I

invest in the risky asset and how much should I place in the risk-free bond?', this may involve borrowing or even going short. After making this decision, the agent submits its order and prices move in an unknown manner - thus giving the agent a new state of the market and a new decision. The agent learns to perform better actions through the rewards it receives. These rewards are stochastic but are shaped to enable the agent to learn to maximise its expected future utility of wealth over multiple future time periods (in contrast to single period investment problems).

The advantage of being able to achieve this is that the agent can learn to approximate merton optimal behaviour for a variety of utility curves and random processes simply by receiving and learning from rewards and interacting with the market in a step-wise manner. The further advantage is that when one does not know the underlying stochastic process - Merton is either silent or ill-specified, whereas the RL agent will remain capable of learning given enough data.

In terms of outcomes, a trained agent would be capable of learning solely from the data by interacting with the market, but also by taking into account multiple time-periods and automatically incorporating risk (because he is maximising the expected utility of future wealth not just the expected wealth itself). It would also be capable of adjusting to a variety of stochastic processes, utility curves with easy extensions such as incorporating transaction costs.

1.2 Structure of the thesis

The thesis comprises both a finance side and a machine learning side.

Chapter 2, first gives a brief background to portfolio theory motivating Merton's problem. From there I introduce the stochastic processes used together with an explanation of utilities and motivation for why one would wish to maximise expected utility and not expected wealth itself. Within the finance section I then

introduce merton's problem mathematically for a geometric brownian motion with log-utility and power utility, the log-utility solution I derived and the surprising (but very well known) result given that the investor in this setting should always invest a constant proportion of one's wealth at every point in time.

Chapter 3, introduces the basics of reinforcement learning necessary to understand the RL agents used in my future experiments.

Chapter 4, covers the bulk of my design, implementation and the test methodology I chose. Implementation involves coding a market environment and an RL agent capable of interacting with one another. For the RL agent to learn it is necessary for me to provide rewards from the market. In the first section I derive the step-wise rewards specific for log-utility and from there illustrate an extension based on mean variance equivalent distributions used by Ritter [25] in his work on Ornstein Uhlenbeck processes.

Due to the noise within this problem, the test methodology is not entirely trivial. There is a genuine difficulty in this problem setting (and in finance in general) in differentiating between skill and luck. Indeed if I simply examined the performance in utilities between a merton optimal agent and a random agent over time, I might learn little (I explain why later). Thus despite it not being necessary for the RL agent to learn (because there is nothing special about an end of episode for the agent, rewards are purely step-wise), I choose an episodic setting.

I ran several million episodes (3 million) for each test and used this to build distributions of episodic utilities, rewards, wealth and sharpe ratios which by batching together I was able to clearly differentiate between the performance of a merton optimal and random agent. I then used this same test methodology for the trained RL agent. I find this approach appealing and perhaps more appropriate when dealing with stochastic processes as we now see the whole distribution of outcomes (not

just some expected value) and there is no chance of being fooled by some excellent out of sample test performance on an inherently random process.

Chapter 5, covers my experiments, which first range from the simplest case. A very low volatility geometric brownian motion with exactly derived log-utility rewards, to more realistic cases such as a high volatility, fat tailed process with a risk aversion parameter suitable for general utility functions. I also demonstrate the effects of tailoring the risk aversion of the RL agent in training, ranging from extremely risk averse (it only wished to invest in the risk free bond) to risk neutral (where although its wealth distribution had a higher expectation than even the merton optimal agent, this came at a huge cost in terms of risk).

Chapter 6, covers my conclusions.

1.3 Literature Review

In this section I shall give a brief over-view of literature I consider necessary for this thesis. This is by no means comprehensive on each of the following subjects.

1.3.1 Finance

Our problem setting is a multi-period asset allocation problem, where we wish to maximise the expected utility of terminal wealth. The theorems of expected utility theory actually go all the way back to the classic work by Von Neumann and Morgenstern [38] and utility theory itself is held up as a standard of rational behaviour in both statistical analysis and economics.

Markowitz [19] motivated and developed the mean-variance approach to portfolio construction, which is probably more recognised and utilised by the actual finance community. However, it is also not without its controversies, both practical and theoretical. The theoretical being 'Borch's paradox' where it is actually impossible to draw indifference curves in the mean-variance plane [5], and the practical being that the Markowitz paradigm (and extensions such as from Sharpe

who developed the CAPM [29]) comprises a single period optimisation and thus does not cover multiple future time periods. From a practitioners viewpoint there are many further problems in reality, the scope of which is outside this thesis but good discussions may be found in Grinold and Kahn [14], or in Ang's excellent book on factor investing [1], with a variety of proposed solutions such as risk-parity proposed by Roncalli [26].

From the point of view of this thesis, the financial angle follows a multi-period problem introduced and solved within certain restrictive contexts by Merton [20]. A good overview of both single and multi-period portfolio choice problems may be found in Chapados [9].

The analytical solution for the Merton problem relies upon financial mathematics (in particular stochastic calculus), of which there are many works such as Bjork [4] or Shreve [31]. For a comprehensive introduction to stochastic optimal control applied to not just the Merton problem, but also more sophisticated applications such statistical arbitrage and optimal liquidation problems Cartea et al, [6] covers much of the material necessary for the financial part of this thesis.

1.3.2 Machine Learning

Machine learning covers a very wide church indeed, and there are various excellent introductions to the broader area such as Hastie et al. [17], Bishop [3] and Murphy [22] however for the Merton problem I shall be using Reinforcement Learning, of which the best current exposition I have found is in Sutton and Barto [34], and also their new addition not yet printed but available online. Many of my experiments will be using an off-policy reinforcement learning setting Q-learning developed by Watkins [39], although I shall be using various extensions such as Double-Q and others. The advantage of Q-learning from the point of view of the Merton problem and a proof of concept is that it is a simple model-free setting. In particular the agent requires no knowledge of the environment and simply learns in a step-wise manner receiving rewards. It is off-policy in that its actions comprise sometimes

exploring randomly as well as greedily, but its target always updates towards the optimal action.

Reinforcement learning is an appropriate setting for this problem due to the step-wise nature of the uncertain rewards and the necessarily online nature of the agent's learning. However it is also a difficult setting due to the noise embedded within the rewards and the fact that the agent has little control over much of the environment bar its own investment choices. However it is more suitable than other machine learning methodologies such as Supervised learning which would require known training targets or Unsupervised learning which is more usually used for classification type problems where data needs compressing or separation.

1.3.3 Reinforcement Learning applied to Finance

Attempts to use reinforcement learning for trading problems go all the way back to the 1990's with notable papers such as Moody and Saffell [21] who derived a differential sharpe ratio type reward using recurrent reinforcement learning (and subsequently created a hedge fund), and also Neuneier [23] who applied Q-learning to a single instrument with no risk aversion.

Some of the techniques applied in this thesis come from Ritter [25] who used a mean-variance approximation to trade an Ornstein-Uhlenbeck process (when I move from my exact log-utility reward formulation, this is the exact reward formulation I use). Another recent and related contribution comes from Halperin, 2017 [15] who used Q-learning to derive a model free agent capable of discrete time option pricing (whichout needing the Black-Scholes Merton formula), in philosophy very similar to this thesis, where the potential benefits of a model-free approach are highlighted.

Finally I would point the reader to a recent paper by Spooner, 2018, [33], where instead of examining the Merton problem he examines reinforcement learning in a market making context, which I believe is an excellent direction. There are a vari-

ety of related problems that may be appropriate for the RL paradigm here (indeed perhaps more so than the Merton problem), in problems such as market making, inventory management, optimal liquidation and many statistical arbitrage problems where the whole limit order book is taken into account we now have a situation where the agent's actions not only receive a reward but where there is also some influence on the next state of the environment (for example placing a limit order of a certain volume immediately changes the limit order book), so he is now longer a pure 'price-taker' as he is for the problem in this thesis, but actually influences the environment also.

Chapter 2

Finance Background

In this section I will give an introduction to the financial ideas needed for this thesis, as well as the data, intuitions and mathematical machinery. Given the potentially very wide scope - I can potentially cover ideas from finance, utility theory, single and multi-period optimisation as well as the later machine learning machinery, I shall brush over some areas (such as utility theory) and only do a deeper dive into areas where I think extra intuition is helpful in order to understand the later experiments.

2.1 Portfolio Theory in brief

I will assume the reader is aware of the works of Markowitz [19], Sharpe [29] and Merton [20]. I will briefly motivate the problem being solved in this section.

Portfolio theory addresses the problem of how best to allocate wealth between competing assets. The seminal work in this area by Markowitz [19] solved this problem with a one-period decision model using a mean-variance approach. Indeed Markowitz's original work showed mathematically the importance of diversification and quantified the risk of a portfolio via its covariance matrix of underlying stock returns. If one can estimate the returns of the underlying stocks as well as the covariance matrix (i.e. the predicted covariances between underlying assets), then Markowitz showed how to find the optimal solution for the portfolio weights.

There are of course a great many problems in practice with naively applying this approach, For example given a large number of stocks one needs a very large time series history to prevent the covariance matrix from becoming singular and in any rate the covariance matrix may have a poor conditioning number and of course stock returns may behave in a heteroscedastic manner and thus blow prior mathematical assumptions away. Various approaches have been tried within industry and academia to overcome these problems such as reducing dimensionality by using factor models (Barr Rosenberg and associates), adding a regulariser or simply applying 'risk parity', of whom the most famour practitioner is the worlds largest hedge fund manager Ray Dalio.

Figure 2.1, illustrates some key points behind Markowitz and Sharpe's work. First that given individual assets whose correlations are less than one, then the potential set of portfolios in a risk-return space will be hyperbolic. If one adds a risk-free asset then there is a 'best' set of portfolios, along the line between the risk-free rate and the tangency portfolio (that with the best risk-return trade off). (This is Sharpe's capital market line). If one thinks of the tangency portfolio as being able to be represented by one asset, then in my future work we are simply choosing asset proportions along this line (I allow borrowing and going short also), but covering multiple time-periods not just one.

The Markowitz paradigm has a problem in that one is locked into a fixed single investment period and is unable to rebalance in the meantime. A re-reading of Markowitz however shows that even in the 1950's he was not naive to these facts. Indeed he addresses multi-period investment by examining how long term geometric returns are damaged by increased variance and thus justifies approximating the geometric mean through the mean of the logarithm, and from here justifies the mean variance approximation. In spirit this is actually incredibly similar to the techniques I use later to build the reward signal for the reinforcement learner on a

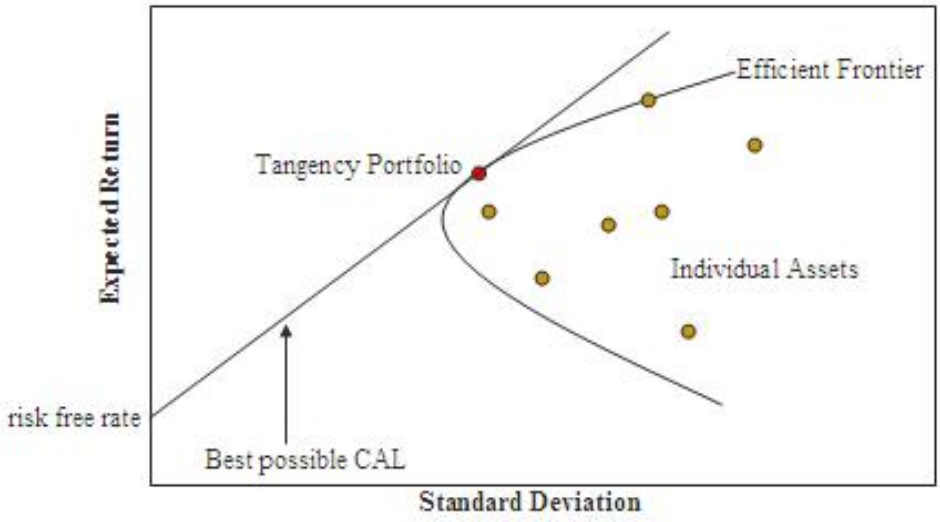


Figure 2.1: Efficient Frontier

step-wise basis, picked up from Ritter [25] applied to a different problem.

From here I shall illustrate the data representations mathematically, introduce utilities and the Merton problem.

2.2 Data representations

In this section I will introduce the data that will be used in the thesis. In all the problems in this project our agent will have at each point in time a choice between investing in a risk free bond with a positive rate of interest r, or an unknown stochastic process. The agent will have no knowledge of the kind of risky process it is trading, the drift rates μ and r of the risky process and bond, and nor will the agent have any awareness of the volatility σ. The agent will also not have any explicit knowledge of the utility function, however rewards will be received shaped specifically to a particular utility function.

2.2.1 Risk-free Bond

The bond is very simple. Given a riskless rate of return r, and a bond of price B_t at time t (Without loss of generality I set $B_0 = 1$). We have that:

$$dB_t = B_t r dt \text{ , therefore } B_t = e^{rt}$$

I will now formalise the stochastic processes used for the risky asset. The theory of stochastic processes is wide and well developed and results below are standard and may be found in textbooks such as [31].

2.2.2 Geometric Brownian Motion

Geometric brownian motion is possibly the simplest stochastic process used to model for a trending risky asset within finance. It is also somewhat naive as it is known that the real markets exhibit both skew, kurtosis, jumps and heteroscedacity. However it is a good starting point and for our purposes, solving Merton's problem, also gives an analytic solution. Of course the hope it that an RL agent can go somewhat beyond these simplifying assumptions and learn to trade more sophisticated prices processes, for which analytical solutions would be unavailable and Merton would be lost.

Given a complete probability space $(\Omega, \mathscr{F}, P)$, with $\mathscr{F}$ being a filtration (i.e. contains measure 0 sets and is right continuous) and if $T > 0$ is our terminal time period. Then $(Z_t)_{t\in[0,T]}$ is a brownian motion if it satisfies the following conditions (note that notationally I would normally use (W_t) or (B_t), however I use W_t for wealth at time t, and B_t for the bond price at time t):

Brownian Motion conditions

- Z is continuous
- $Z_t - Z_s$ is independent of $\mathscr{F}_s, 0 \leq s < t$
- $\forall t > 0, Z_t \sim N(0,t)$ under $P, 0 \leq s < t$

Therefore if the risky stock S_t evolves as a geometric brownian motion (it certainly does not in the real world...), then we have:

$$dS_t = S_t(\mu dt + \sigma dZ_t) \text{ , with } S_0 = s_0 > 0$$

Without loss of generality I use $s_0 = 1$. Note also the drift term μ and volatility $\sigma > 0$. Therefore using Ito calculus, it is well known that we can derive the price evolution of our risky asset as follows:

$$S_t = S_0 e^{(\mu - \frac{\sigma^2}{2})t + \sigma Z_t}$$

In my later work, I will create a market environment by discretising the above equations. In general for any of my stochastic processes the wealth W_t of the agent will also evolve stochastically according to:

$$dW_t = N_t^B dB_t + N_t^S dS_t$$

Where N_t^B and N_t^S, are the numbers of bonds and stocks held at time t.

2.2.3 Fat Tails

It is commonly known that financial time series rarely seem to exhibit Gaussian noise, but have fatter tails where the probability of extreme loss is much lower than a Gaussian might suggest.

For my later tests I chose to add random increments from the Student-t distribution with 10 degrees of freedom in order to simulate adding fatter tailed noise. With a large number of degrees of freedom the Student-t approaches a Gaussian, with a low number then extreme moves become more likely. The equation for the pdf is given below.

$$\Pr(x, df) = \frac{\Gamma\left(\frac{df+1}{2}\right)}{\sqrt{\pi df}\,\Gamma\left(\frac{df}{2}\right)} \left(1 + \frac{x^2}{df}\right)^{-\left(\frac{df+1}{2}\right)}$$

The Student-t pdf for varying degrees of freedom is given in figure 2.2.

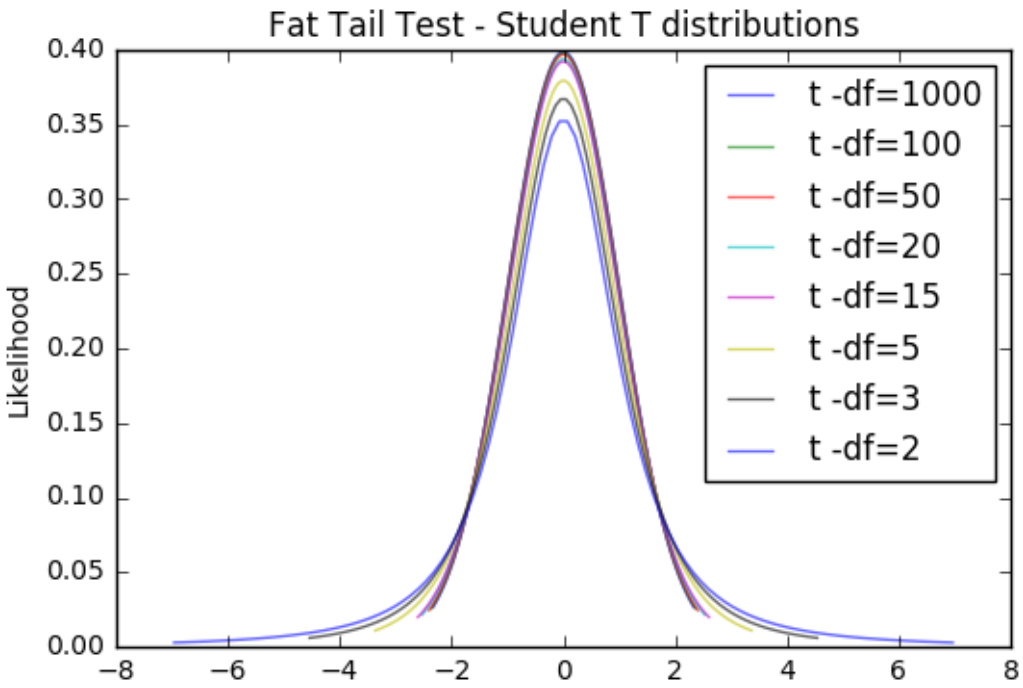


Figure 2.2: Student-T pdfs - Fat Tail test

2.3 Risk Neutrality and Utility Curves

To give intuition, I will borrow from Professor Gordon Ritter's simple thought experiment from his lectures at NYU. Imagine you own two gold bars located abroad. You are only able to realise the value of these gold bars if they are shipped home. The problem being the location is rife with pirates and at present shipments have a 50% chance of being lost.

There are two ships available, should you put one gold bar on each ship and sent home separately? or should you put both gold bars on one ship and take your chances? A risk neutral investor might say, 'Well there is no difference, the expected value of either option is one gold bar'. This is of course true.

Expected terminal wealth

- One ship: $E(w_T) = \frac{1}{2} * 2 = 1$
- Two ships: $E(w_T) = \frac{1}{2} * 1 + \frac{1}{2} * 1 = 1$

Thus the wealth at terminal time T is one gold bar in either case. However for most people this does't quite sit right and they might prefer to spread their risk or diversify. These are risk averse investors. If one now examines the variance.

Expected Volatility of terminal wealth

- One ship: $V(w_T) = \frac{1}{2} * (0-1)^2 + \frac{1}{2} * (2-1)^2 = 1$
- Two ships: $V(w_T) = \frac{1}{4} * (0-1)^2 + \frac{1}{4} * (2-1)^2 + \frac{1}{2} * (1-1)^2 = 0.5$

We see that those who choose to diversify with two ships experience half the variance of terminal wealth. These differences can be represented by 'utility'.

In this work I will be attempting to maximise the expected utility of terminal wealth over multiple time periods. That is given a utility curve U, in all problems I am attempting to solve:

$$\max \mathbf{E}[U(w_T)] \text{ for } t \in [0, \ldots, T]$$

2.3.1 Utility Curve Intuitions

There is a whole body of literature and theory on utility curves, which I am not going to go into. However I will give a few key takeaways.

- Utility Curves are upward sloping (we have a higher utility of greater wealth).
- A risk neutral investor has a linear utility curve.
- Risk Aversion introduces concavity in the utility curve.
- By introducing a utility function, the path to our wealth becomes important, extreme ups and downs gives us less utility than a smooth path. Thus risk is intertwined within the concept of utility.

- Over multiple time steps it is most often NOT the case that the optimal set of actions to maximise the expected utility of wealth long term is identical to taking a series of optimal myopic wealth maximising actions.

The intuition as to why it may not be the cleverest thing to maximise wealth itself becomes more obvious when one realises that over repeated time periods such an investor would always accept ANY positive expectation bet, and be willing to risk all of his wealth to do so. Indeed he would indeed be happy to gamble his entire wealth as long as the outcome had the same expectation as the certainty of not gambling it at all.

2.3.2 Mean Variance Utility

Later on I will apply reinforcement learning to the Merton problem, but in order to do so I will need to reformulate the expected terminal utility into a series of rewards. I will be taking a mean-variance approximation to the terminal utility and then creating step by step rewards which in expectation should match the mean-variance approximation. Theoretically this is actually very similar to mean-variance utility which I now introduce.

Markowitz [19] groundbreaking work which contributed to his Nobel prize, used mean variance utility which was only fully formalised in terms of expected utility many years later in a joint paper with Levy, [18]. They approximated the expected utility function as follows:

$$E(U(1+r_T) \approx U(1+E(r_T)) + \frac{1}{2}U^{''}(1+E(r_T))var(r_T)$$

As mentioned above utility functions are concave, therefore the second derivative is negative and thus any investor maximising expected utility is similar to an investor maximising the expected mean return given a certain variance.

Within the finance industry mean variance optimisation is used, but not as of-

ten as one might think and is often distrusted by practitioners. This is for a variety of reasons (singular or poor conditioning number for covariance matrix of returns), the difficult in predicting the covariance matrix (although a whole industry using factor models developed attempting this).

It is a common misconception that the weakness of Markowitz work is that we require returns to be normally distributed (which is unrealistic). However one can see from the above equation that this is simply not the case, the lack of realism comes of course when means and variances are poor statistics to describe the overall returns distribution.

I will now move from Markowitz's single period optimisation above to introduce Merton's multi-period solution.

2.4 Merton's Problem

The problem of portfolio optimisation again comprises a large body of literature dating from Markowitz [19] on portfolio construction and choice, however I shall skip over this as much of the work concentrates on how best to construct a portfolio over a single period. In this thesis we are more interested in the question of maximising the utility of our final wealth over multiple time periods. This is the classical portfolio optimisation from Merton [20]. As mentioned earlier the problem is to find:

$$\max_{u} E[U(W_T)|W_0 = w_o] \text{ for } t \in [0,\ldots,T]$$

2.4.1 Merton solution - Logarithmic Utility

My initial reinforcement learning experiments begin where the investor has logarithmic utility. The main reason being that first the solution to the Merton problem with logarithmic utility and geometric brownian motion has a very nice clean analytic formula and is thus easy to test against. The second reason is that this is also easy to reformulate in terms of myopic rewards on a step by step basic for a

reinforcement learning agent, which I will demonstrate later.

The solution to Merton's problem requires mathematical machinery from stochastic optimal control which I won't go into, detailed introductions to this can be found in Alvaro Cartea's book on Algorithmic and high frequency trading [6].

I give a clean sketch of the proof for Merton using logarithmic utility below, this basically follows an MSc thesis by Kinga Tikosi on Merton's problem barring one small change I made at the end.

For $U(W) = log(W)$, the optimal policy is

$$u_t^* = \frac{\mu - r}{\sigma^2}, \text{ for all } t \in [0, T]$$

In other words the surprising result for geometric brownian motion with a logarthmic utility is that regardless of ones wealth level at every time period one should invest a fixed proportion of one's wealth based upon the drift of the risky asset μ, the risk-free rate of return r, and the volatility of the risky asset σ.

Proof

The wealth W_t of the agent will evolve according to:

$$dW_t = N_t^B dB_t + N_t^S dS_t$$

Where N_t^B and N_t^S, are the numbers of bonds and stocks held at time t. If we reformulate the number of stocks and bonds bought in terms of the investment proportion of our wealth invested in the risky asset at time t, u_t, then we have that

$$N_t^B = \frac{(1-u_t)W_t}{B_t} \text{ and } N_t^S = \frac{u_t W_t}{S_t}$$

Given the above we can now write the evolution of our wealth process as fol-

lows:

$$\begin{aligned} dW_t &= (1-u_t)W_t r dt + u_t X_t(\mu dt + \sigma dZ_t) \\ &= W_t((r+u_t(\mu-r))dt + \mu\sigma dZ_t) \end{aligned}$$

If we imagine a solution of the form,

$$W_t = w_0 exp(\int_0^t g_s ds + \int_0^t h_s dZ_s)$$

Then applying Ito we have,

$$\begin{aligned} dW_t &= W_t g_t dt + X_t h_t dZ_t + \frac{1}{2} W_t g_t^2 dt \\ &= W_t((g_t + \frac{1}{2}h_t^2)dt + h_t dZ_t) \end{aligned}$$

We can now read off our values for g_t and h_t, so:

$$W_t = w_0 exp(\int_0^t (r+(\mu-r)u_s - \frac{1}{2}\sigma^2 u_s^2)ds + \int_0^t \sigma u_s dZ_s$$

We wish to maximise $E(log(W_T^u)|W_0 = w_0)$, therefore using the fact that $E(\int_0^T \sigma u_t dZ_t) = 0$. We have that

$$J(w_o, u) = log w_0 + E\left(\int_0^T (r+(\mu-r)u_s - \frac{1}{2}\sigma^2 u_s^2)ds\right)$$

If we now take derivatives

$$\frac{\partial}{\partial u}(r+(\mu-r)u_s - \frac{1}{2}\sigma^2 u_s^2) = \mu - r - \sigma^2 u$$

$$\frac{\partial}{\partial^2 u}(r+(\mu-r)u_s - \frac{1}{2}\sigma^2 u_s^2) = -\sigma^2$$

The negativity of the second derivative proves concavity and we can solve for

u by setting the first derivative equal to zero. Therefore

$$u_t^* = \frac{\mu - r}{\sigma^2}$$

with a value of

$$J(w_o, u^*) = log(w_0) + \left(r + \frac{(\mu - r)^2}{2\sigma^2} \right) T$$

(My final equation differs slightly from that derived by Tikosi, where I believe he may have made a small slip at the end).

In my future experiments, I shall discretise the above for simulation and test purposes. I will compare a trained reinforcement learning agent trading over multiple periods, to the Merton optimal analytical solution given above.

2.4.2 Merton Solution - Power Utility

The second utility I shall use in my experiments is the power utility. This is more general than log-utility and is formulated as follows:

$$U(x) = x^\gamma \, , 0 < \gamma < 1$$

If the stochastic process is a geometric brownian motion then the Merton optimal investment is again a fixed proportion of one's wealth regardless of the level of the wealth or indeed the level of the GBM. It can be shown by a similar derivation to the above, that the Merton optimal investment is:

$$u^* = \frac{\mu - r}{\sigma^2(1 - \gamma)}$$

The parameter γ, reflects the degree of risk aversion. I shall show later that regardless of the type of utility curve, as long as the curve is upward sloping and the underlying stochastic process comes from a mean-variance equivalent distribution (which includes elliptical distributions) then we have a simple reformulation of the utility curve into an equivalent mean-variance trade-off. This fact is used later to

enable my shaping rewards for the RL agent.

Chapter 3

Machine Learning Background

3.1 Reinforcement Learning

I believe the best current exposition of Reinforcement Learning comes from 'Reinforcement Learning - An Introduction', by Sutton and Barto [34], as well as their new edition (currently on-line). My mathematical explanation of the parts of Reinforcement Learning needed for this thesis closely follows [34]. For the most part the RL models used were relatively basic tabular models, there are of course many extensions possible particularly those involving function approximation such as DeepQ-learning and methods using Policy gradients. In this section I will work form the basics and justify the algorithms used.

3.1.1 The Reinforcement Learning setting

Unlike Supervised Learning, where we have a training set with training labels and attempt to predict labels on an unseen test set. And unlike Unsupervised Learning, where one has no labels and attempts to group similar data items together in some fashion. Reinforcement learning has a quite different problem setting.

In Reinforcement Learning we have an agent, who interacts with an environment and receives a reward in an online fashion. These rewards may be delayed, and noisy. The RL agent is able to interact with its environment by choosing an action, which results in a reward and a new state (not necessarily the one the agent hoped for as this also depends upon the environment). Learning in an RL sense

comprises the agent learning a set of actions (a policy) appropriate to whatever state it finds itself in, in order to maximise its expected future rewards. I will formalise this with some mathematics, however the key intuition is that this is actually a very broad and ambitious goal for a machine learning algorithm and that it appears to capture many real life situations in principle.

Figure 3.1, from Sutton and Barto, shows the basic setup.

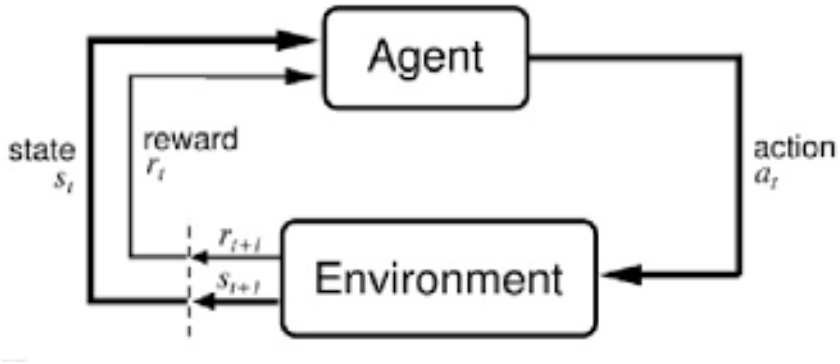


Figure 3.1: Basic RL framework

In a finance setting we may for example imagine the market itself is an environment our agent interacts with. Perhaps price levels, and the agent's wealth level are possible states (as well as any features that we believe may influence the agent's actions). The agent action at each time step may be to invest in a stock or set of stocks with certain weightings, and the reward perhaps a risk adjusted change in wealth. In a completely different setting it could be to safely drive a self-driving car despite unknown terrain.

To contrast with supervised learning, this is far broader than for example regressing states against rewards, as rewards may be delayed and the states the agent may find itself within dependent on its previous actions. And it is also far more sophisticated than taking the best myopic action at each step as this may prove to be suboptimal over multiple steps. If the last sentence reminds one of dynamic programming, then that is exactly the right way to think.

Reinforcement Learning thus involves elements of optimisation, online learning and has to manage additional problems such as when to explore and when to exploit its existing knowledge. Many problems also require a function approximator such as a neural network to simplify a potentially intractable state, action space. It may also involve aspects of transfer learning, in moving from a simulated environment to a real world setting. It is thus both one of the oldest forms of machine learning, but also one that I believe shows a great deal of promise.

I introduce the standard basic results for reinforcement learning below. Again this is book work and thus borrows heavily from Sutton and Barto [34] and from the Deep Mind lectures at University College London.

3.1.2 Markov Decision Processes

3.1.2.1 Markov Property

A state s has the Markov property if $\forall s' \in S$ and $\forall$ rewards $r \in R$ and all histories $S_1, \ldots, S_{t-1}$

$$\mathscr{P}(R_{t+1} = r, S_{t+1} = s' | S_t = s) = \mathscr{P}(R_{t+1} = r, S_{t+1} = s' | S_1, \ldots, S_{t-1}, S_t = s)$$

A market practitioner may look at this and say well this does not apply in reality, in some ways true. However, the fact that all prior information is subsumed in the current state is not as restrictive as first appears. Because of course one can add whatever prior information one wishes to the current state. The state-action space becomes more complex of course, however the point is that RL algorithms can use information prior to the current time step as long as this is included in the current state.

3.1.2.2 State Transition Matrix

If we have random states with the markov property, then we can define the state transition probability as:

$$P_{ss'} = \mathscr{P}(S_{t+1} = s' | S_t = s)$$

Given a finite set of states this is gives a state transition probability matrix and is part of the model of the environment. Bar the simplest settings such as dynamic programming, this model of the environment is likely to be hidden from the agent (and certainly is in this thesis, so Dynamic Programming will not be a solution). There are various ways an RL agent can deal with this, from learning its own model of the environment online, to simply learning optimal actions from rewards without caring, to various combinations in between (such as Dyna).

3.1.2.3 Markov Process

A Markov process is now easy to define as a set of states with a state transition matrix.

3.1.2.4 Markov Reward Process

A Markov reward process is simply a markov process with rewards. i.e we add a reward function. For both purposes of convengence (for non episodic environments) and also to alter the myopia of the agent, we may also discount rewards with a variable $\gamma \in (0,1)$.

$$\mathscr{R}_s = E(R_{t+1} | S_t = s) \tag{3.1}$$

3.1.2.5 Return

The return is the total discounted reward from a given time step t.

$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{3.2}$$

3.1.2.6 Markov Decision Process

In my view the core of RL lies around Markov Decision Processes and using variations on the Bellman equation to find optimal control values. There is more to it, but a lot is built upon these foundations. A Markov Decision Process (MDP) is a

Markov Reward Process with decisions.

Thus we adjust our state transition probability matrix and reward function as follows:

$$P^a_{ss'} = \mathscr{P}(S_{t+1} = s' | S_t = s, A_t = a)$$

$$\mathscr{R}^a_s = E(R_{t+1} | S_t = s, A_t = a)$$

or more concretely,

$$p(s', r | s, a) = \mathscr{P}(S_{t+1} = s', R_{t+1} = r | S_t = s, A_t = a)$$

therefore,

$$P^a_{ss'} = \sum_{r \in R} p(s', r | s, a)$$

and

$$\mathscr{R}^a_s = \sum_{r \in R} r \sum_{s' \in S} p(s', s | s, a)$$

3.1.2.7 Policy

A policy is a distribution of action over states. Notation hasn't quite settled down here yet, so following the Deep Mind lecturers...

$$\pi(s, a) = \pi(s|a) = \mathscr{P}(A_t = a | S_t = s)$$

3.1.2.8 State Value Function

The state value function $v^\pi(s)$ of a markov decision process is the expected return from beginning in a state s and following a policy π.

$$v^\pi(s) = E_\pi(G_t | S_t = s)$$

3.1.2.9 Action Value Function - Q

The action-value function $q^{\pi}(s,a)$ is the expected return from starting in a state s, taking an action a and then following the policy π.

$$q^{\pi}(s,a) = E_{\pi}(G_t|S_t = s, A_t = a)$$

3.1.3 The Bellman Expectation equation

For state value functions we can see that:

$$\begin{aligned} v^{\pi}(s) &= E_{\pi}(G_t|S_t = s) \\ &= E_{\pi}(R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots|S_t = s) \\ &= E_{\pi}(R_{t+1} + \gamma(R_{t+2} + \gamma^2 R_{t+3} + \ldots|S_t = s) \\ &= E_{\pi}(R_{t+1} + \gamma G_{t+1}|S_t = s) \\ &= E_{\pi}(R_{t+1} + \gamma v^{\pi}(S_{t+1})|S_t = s) \end{aligned}$$

using the law of iteration expectations.

Similarly for action value functions we can see that:

$$q^{\pi}(s,a) = E_{\pi}(R_{t+1} + \gamma v^{\pi}(S_{t+1}, A_{t+1})|S_t = s, A_t = a)$$

It is straightforward to see the connection between state value functions and action-value functions as follows:

$$v^{\pi}(s) = \sum_{a \in A} \pi(s,a) q^{\pi}(s,a)$$

$$q^{\pi}(s,a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v^{\pi}(s')$$

Therefore

$$v^{\pi}(s) = \sum_{a \in A} \pi(s,a)(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v^{\pi}(s'))$$

and

$$q^{\pi}(s,a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \sum_{a' \in A} \pi(s',a') q^{\pi}(s',a')$$

The bellman expectation equation thus gives us a method of finding the value of being in a particular state given a policy, or the value of being in a state, taking an action and then following the policy. This is nice, however we are more interested in finding the best policy.

3.1.3.1 Optimal State Value Function

The optimal state-value function $v^*(s)$ is the maximum value function over all possible policies.

$$v^*(s) = \max_{\pi} v^{\pi}(s)$$

3.1.3.2 Optimal Action Value Function

Similarly the optimal state-value function $q^*(s,a)$ is the maximum value function over all possible policies.

$$q^*(s,a) = \max_{\pi} q^{\pi}(s,a)$$

3.1.3.3 Optimal Policies

We can define a partial ordering over policies $\pi \geq \pi'$ if $v^{\pi} \geq v^{\pi'}(s), \forall s$. It is known that for any markov decision process $\exists \pi^* \geq \pi \; \forall \pi$ and also that $v^{\pi^*}(s) = v^*(s)$ and $q^{\pi^*}(s,a) = q^*(s,a)$. In other words there is an optimal policy and all optimal policies achieve the same optimal state and action value function.

Therefore if we know $q^*(s,a)$, then we know our optimal policy immediately.

$\pi^*(s,a) = 1$ if $a = \arg\max_{a \in A} q^*(s,a)$ and 0 otherwise

3.1.4 The Bellman Optimality equation

Again we can use bellman this time on our optimal value and action value functions.

$$v^*(s) = \max_a q^*(s,a)$$

$$q^*(s,a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v^*(s')$$

Therefore,

$$v^*(s) = \max_a R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v^*(s')$$

and,

$$q^*(s,a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \max_{a'} q^*(s',a')$$

However because of the maximum term there is in general no closed form solution for this equation. There are various iterative solution methods. Policy and Value iteration as well as Q-learning and Sarsa. In my first RL experiments I tested Sarsa, Q-learning, Double-Q learning, and Dyna. Unlike dynamic programming or earlier solution methods these methods do not require that the model of the environment is given, they either learn from experience or learn from experience whilst hypothesising the agent's own model.

3.1.5 Learning algorithms - From Dynamic Programming to Dyna

We now have the Bellman optimality equation for both $q^*(s,a)$ and $v^*(s)$, the key question is what algorithms can we use to find this optimum? It turns out that there is in general no closed form solution for this bellman optimality, however there are various iterative solution methods, some common methods include dynamic programming methods, monte carlo methods and TD learning.

3.1.5.1 Dynamic programming methods

If we already have a policy π that we wish to evaluate for every state then we know from Bellman that:

$$\begin{aligned} v_\pi(s) &= \mathbf{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s) \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')] \end{aligned}$$

Using the above it is simple to use an iterative method to derive successive approximations to the bellman equation where the final equation itself is a fixed point. Thus if we guess some value for all states bar the terminal V_0 then successive approximations are derived for $k = 0, \ldots,$ and $\forall s \in$ by performing the following iteration until convergence:

$$v_{k+1}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r'|s, a)[r + \gamma v_k(s')]$$

This is called policy evaluation, indeed after convergence we will have a new set of values $v(s)$ and for example by acting greedily with respect to these values we can derive a new policy. By performing policy evaluation on this policy we again have a new set of values $v(s)$ and so on. It can be shown that this will converge to an optimal policy and is called policy iteration.

There are various adjustments to this which I won't go into (for example we don't necessarily need to go all the way to convergence when we do policy evaluation and can instead do value iteration), but for the purposes of this thesis I demonstrate this solely to highlight a somewhat obvious flaw for our financial setting. To successfully perform this iteration we must already know the transition probabilities $p(s', r'|s, a)$, which implies that we already need to have a perfect model of the market. Thus, if we knew the model of the market we could perform this bootstrapping operation in a step-wise manner and solve the Merton problem.

Indeed, these are precisely the properties that Merton used in solving his problem (albeit in a continuous time setting using a reformulation of Bellman called the HJB equation), which is why Merton must know the formulation of the stochastic risky process in advance, as well as its parameters in order to solve his equation.

However our goal is somewhat different, we wish to learn the optimal investment strategy in a model free fashion, with minimal assumptions and purely through and agent's interactions with the market.

3.1.5.2 ε-Greedy exploration

Before taking a very brief look at Monte Carlo methods and why I chose not to use them I will briefly explain ε-greedy exploration.
If one imagines that an agent always takes the best action, then learning is compromised. The problem is that if for example the agent takes a random initial action, after the first positive update of the Q values then the agent will only ever take this specific action in this state. The agent will never try other potential actions that might be better and will therefore never fully explore the state-action. To counter this and ensure convergence I use an ε-greedy policy. The agent takes the best action in any given state with probability $1-\varepsilon$ and takes a random action with probability ε. This ensures the complete exploration of the state-action space.

3.1.5.3 Monte Carlo methods

Monte Carlo methods are capable of learning without a model of the environment, however they rely upon sampling experience in an episodic fashion. I did create an episodic environment for this particular problem (partly for test purposes), however I also did not wish to be tied to it, the methods used in this thesis can be just as easily used in a continuing environment due to my use of step-wise rewards and the fact there is nothing special about reaching the end of an episode.

Monte carlo methods rely upon being able to reach an end state repeatedly and then update the q-values based upon their respective reward experience. Thus although the prior issue of needing an exact model of the environment is solved,

we create a new issue in that the environment itself needs to be episodic in nature (for example millions of games of chess). Therefore I chose not to use monte carlo methods. For completeness I include the every visit Monte Carlo control algorithm below (algorithm 1), notice that this is an on policy (we update the q-values of the policy we are actually following) control algorithm using ε-greedy exploration and also that it is usually more straighforward although memory intensive to work directly with q-values for control problems rather than v-values:

Algorithm 1 On Policy MC Control

procedure MC CONTROL
  Initialise:
  $\pi \leftarrow$ some ε-soft policy
  $Q(s,a) \in \mathscr{R}\ \forall s \in S, a \in A$
  $Returns(s,a) \leftarrow$ an empty list $\forall s \in S, a \in A$
  while True **do** ▷ Loop forever for each episode
    Generate an episode following π:$S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$
    $G \leftarrow 0$
    while t $\leq T$ **do** ▷ Loop for each step in episode $t = T-1, T-2, \ldots, 0$
      $G \leftarrow G + R_{t+1}$
      Append G to $Returns(S_t, A_t)$
      $Q(S_t, A_t) \leftarrow average(Returns(S_t, A_t))$
      $A^* \leftarrow \arg\max_a Q(S_t, a)$
      $\forall a \in A$:
      $\pi(a|S_t) \leftarrow 1 - \varepsilon + \frac{\varepsilon}{|A|}$,if $a = A^*$
      $\pi(a|S_t) \leftarrow \frac{\varepsilon}{|A|}$,if $a \neq A^*$

3.1.5.4 TD, and Q learning

Temporal difference methods are capable of learning without having to reach the end of an episode unlike monte carlo methods, but also without needing a model of the environment unlike dynamic programming methods. For most of my experiments I used the Q-learning algorithm (and sometimes a variant Double-Q learning) created by Watkins [39]. This is an off-policy TD variant in that the learned q values approximate the optimal q values independent of the policy being followed (I used ε-greedy policy exploration. The key update is:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_t + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$$

Thus instead of our target being the return from the whole episode as in monte carlo methods the target is bootstrapped from the next state as in dynamic programming. by choosing the $\max_a$ in our target update we are performing an off-policy update of maximal actions in each state. Q-learning is effectively learning from a guess, but one where there is slightly greater information available. For the Merton problem I considered Q-learning to be the simplest suitable RL algorithm.

Algorithm 2 Q-learning

procedure Q LEARNING
  Initialise:
  Step size: $\alpha \in (0,1]$
  $Q(s,a) \in \mathscr{R}\ \forall s \in S, a \in A$
  while True **do** ▷ Loop for each episode
    Initialise S
    while t $\leq T$ **do** ▷ Loop for each step in episode $t = 0, \ldots, T$
      Choose A from S, ε-greedy policy
      Take action A and receive from environment R,S'
      $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S',a) - Q(S,A)]$
      $S \leftarrow S'$

Here α is our step-size and controls our learning rate and γ is our discount rate. On convergence the optimal policy π^* is found by acting greedily with respect to Q^*.

3.1.5.5 Double-Q learning and Double Sarsa

In Q-learning there is a known bias called the maximisation bias. This is because we are maximising when creating our target policies but taking this maximum over estimated values. Sutton and Barto [34] give a simple demonstration of this in their book. To avoid maximisation bias I tended to use Double-Q learning for the majority of my experiments. This is a simple adjustment where we maintain two separate sets of q-values and we randomly update one or the other at each step. When choosing my optimal policy I then acted greedily with respect to the average of the q-values. In the case of Sarsa the difference lies in the fact that the policy we follow is also our target policy. From an algorithmic viewpoint the code is almost

identical, but the effects in terms of agent behaviour can be quite different, one example being windy gridworld in Sutton and Barto [34].

Algorithm 3 Double Q-learning

procedure DOUBLE Q LEARNING
  Initialise:
  Step size: $\alpha \in (0,1]$
  $Q_1(s,a), Q_2(s,a) \in \mathscr{R}\ \forall s \in S, a \in A$
  while True **do** ▷ Loop for each episode
    Initialise S
    while t $\leq T$ **do** ▷ Loop for each step in episode $t = 0, \ldots, T$
      Choose A from S, ε-greedy policy in $Q_1 + Q_2$
      Take action A and receive from environment R,S'
      if $p < 0.5$ **then** ▷ If p probability is 0.5 or less
        $Q_1(S,A) \leftarrow Q_1(S,A) + \alpha[R + \gamma Q_2(S', \arg\max_a Q_1(S',a)) - Q_1(S,A)]$
        $S \leftarrow S'$
      else
        $Q_2(S,A) \leftarrow Q_2(S,A) + \alpha[R + \gamma Q_1(S', \arg\max_a Q_2(S',a)) - Q_2(S,A)]$
        $S \leftarrow S'$

3.1.5.6 Dyna

The final tabular RL agent I tested in my experiments is Dyna. Although the aim of this thesis is to apply model free reinforcement learning methods, Dyna could be classified as being model based. However it is model free in terms of its requirements from the environment, the difference in philosophy is that it builds its own model as it gains experience, in order to predict how the environment will respond. Given this model it can also simulate its own experience (called experience replay) thus it learns by both interacting with the environment and gaining real experience and its own simulated experience. Sutton and Barto [34] give a conceptual diagram showing the idea behind Dyna which is shown in figure 3.2 and I also show the algorithm below, notice the use the model, as well as two sets of q-value updates, one from real experience and one from experience replay.

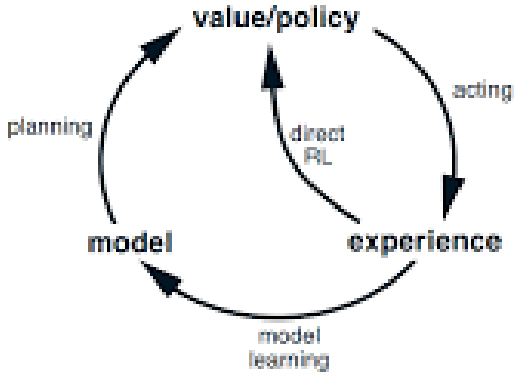


Figure 8.1: Relationships among learning, planning, and acting.

Figure 3.2: Dyna conceptual model

Algorithm 4 Dyna

procedure DYNA
  Initialise:
  Step size: $\alpha \in (0,1]$
  $Q(s,a) \in \mathscr{R}$ and $Model(s,a)\ \forall s \in S, a \in A$
  while True **do**
    $S \leftarrow$ Current non-terminal state
    $A \leftarrow \varepsilon$ -greedy(S,Q
    Choose action A and receive reward R and state S'
    $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \max_a Q_(S',a)) - Q(S,A)]$
    $Model(S,A) \leftarrow R,S'$
    while i in range(n) **do** ▷ Loop for n, number of experience replays
      Choose A from S, ε-greedy policy in Q
      Take action A and receive from environment R,S'
      S $\leftarrow$ random previous observed state
      A $\leftarrow$ random action previously taken in S
      $R,S' \leftarrow Model(S,A)$
      $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \max_a Q_(S',a)) - Q(S,A)]$

3.2 Advanced Reinforcement Learning

3.2.1 Function Approximation

3.2.2 Deep Learning

3.2.3 Deep Q and Double Q Learning

3.2.4 Prioritised experience replay and Noisy Networks

Chapter 4

Implementation

In this section I will lay out the implementation and test metrics for future experiments. A major aspect in creating the environment between the market and agent is of course to give rewards to the RL agent that it can learn from in order to hope to approach the performance of a Merton optimal agent.

4.1 Reward Shaping

We need to reformulate the terminal utility of wealth at time T, into a series of step by step rewards g_t, for $t \in [1,\dots,T]$. With a logarithmic utility curve this is simple:

$$\begin{aligned} U(W_T) &= log(W_T) \\ &= log(W_0(1+r_1)(1+r_2)\dots(1+r_T)) \\ &= log(W_0)+log(1+r_1)+\cdots+log(1+r_T) \\ &= log(W_0)+log(\frac{dW_1}{W_0})+\cdots+log(\frac{dW_T}{W_{T-1}}) \end{aligned}$$

Where dW_t is the change in wealth from period t-1, to t and r_t the return in the corresponding period.

In reinforcement learning section we wish to maximise the expected sum of (discounted rewards) i.e. $E(g_1+\cdots+g_T)$. By letting each reward $g_t = log(\frac{dW_t}{W_{t-1}})$, then maximising the expected sum of these rewards is precisely equivalent to max-

imising the expected utility of terminal wealth $U(W_T)$ given above. Basically informing my initial choice of log utility and GBM, to get started. Therefore the step-wise rewards signals at least for the initial experiments using log-utility and GBM should be an exact match for end of episode utilities.

4.1.1 Discretisation - terminal utilities and step-wise rewards

Given a geometric brownian motion with drift $\mu = 0.11$ and a volatility $\sigma = 0.028$, I discretised to 20 time steps, each episode starting with a stock price $s_0 = 1$. Ten sample realisations are shown in figure 4.1.

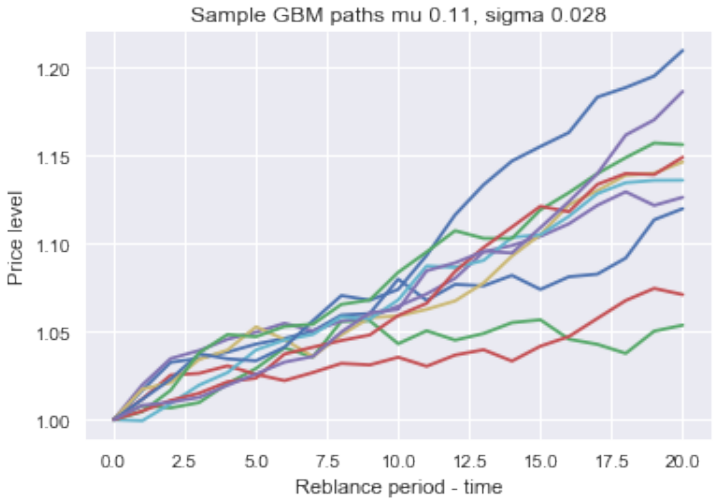


Figure 4.1: Low Volatility Geometric Brownian Motion Paths

For log-utility we already know the formula for the continuous-time optimal Merton ratio derived in the finance background section. That is $u^* = \frac{\mu - r}{\sigma^2}$. In this particular case this would imply that the Merton optimal solution would involve levering up his wealth and investing 12.75x his wealth in the stock at every time step. By discretising we are of course no longer rebalancing in continuous time.

In figure 4.2, the left chart shows the result of mean end utilities for various fixed ratio investment strategies. I illustrate a range of investment proportions to invested the risky stock, both long and short from -10x to 25x investment in the

stock.

Despite the discretisation, the simulation matches the analytic solution given by Merton well, with an optimum at 12.5x (the investment ratio available closest to the analytic solution).

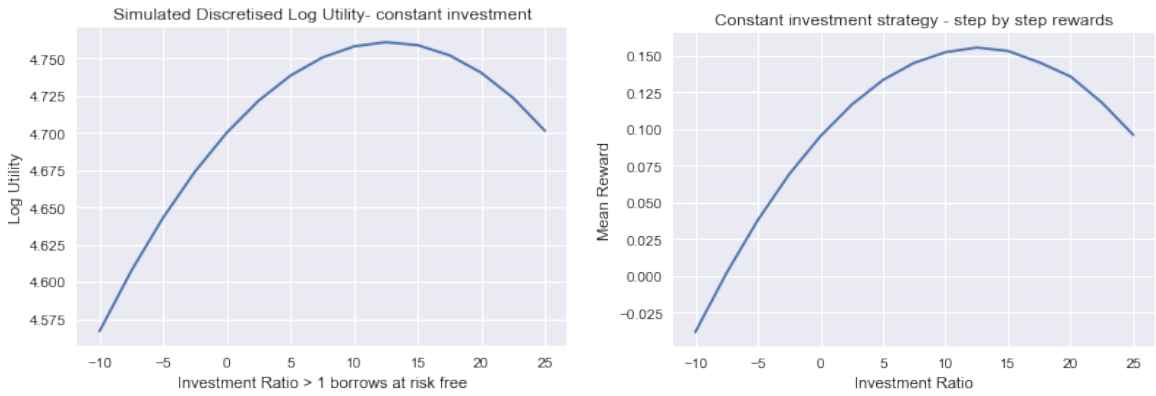


Figure 4.2: Discretised Log Utility versus Step-wise rewards

The right chart in figure 4.2 shows the same investment strategies now tested in the step-wise market environment. Recall that log-utility is myopic so as derived earlier I model the rewards on a step-wise basis for log-utility as $r_t = log(\frac{dW_t}{W_{t-1}})$.

The simulation again involves multiple episodes where a range of fixed investment strategies were used, however the chart now shows the mean rewards. The two charts should match very closely. In particular we would hope that the maximum mean reward corresponds to the maximum utility and also closely matches the Merton optimal solution of 12.75x leveraged.

The charts do indeed match closely match. Indeed the best strategy for fixed investment with step-wise rewards would also have an optimal fixed investment of 12.5x. Which is the closest action available to the analytical solution provided by Merton and so we have moved from end of episode utilities to step-wise rewards, albeit specifically shaped for log-utility and GBM.

4.1.2 Mean Variance Equivalent Distributions, Episodic

We will not be fortunate enough to move so smoothly from end of episode utilities to step-wise rewards for most processes and utility curves. In Ritter's [25] work last year, he trained a Q-learning agent to learn to trade an Ornstein Uhlenbeck process with transaction costs, the technique he used being closely related to Chapter 2 where I describe mean-variance utilities. In order to generalise beyond log-utilities, I decided to use Ritter's approximation and apply this to the Merton problem.

Specifically, a random variable **r** follows a **mean-variance equivalent** distribution if it has a density $p(\mathbf{r})$, which has first and second moments and where for any increasing utility function u, there exists a constant $\kappa > 0$, such that the policy maximised by $\mathbf{E}(u(w_T))$ is also optimal for the simpler problem:

$$\max_{\pi}[\mathbf{E}(w_T) - \frac{\kappa}{2}\mathbf{V}(w_T)]$$

Given this equation one can clearly see the connection with Markowitz's work, [19], and also mean-variance utility which I laid out in the finance section. The difference here being an extra parameter κ, our risk aversion parameter, which reshapes our distribution to approximate the utility curve. Note that although the optimal solution matches exactly, the whole curve need not necessarily match as this is now a second order approximation.

In figure 4.3, I again ran the simulation for fixed levels of investment. However this time I use the above mentioned, mean-variance utility of terminal wealth. In this case my risk aversion paramter $\kappa = 0.006$, one can clearly see that this matches the curves above for both utilities and rewards, however we are now no longer tied to log-utility.

This mean-variance equivalent approximation however takes us back into an episodic world with figure 4.3 illustrating end of episode results, so in my final adjustment to enable step-wise rewards I follow Ritter [25], as illustrated below.

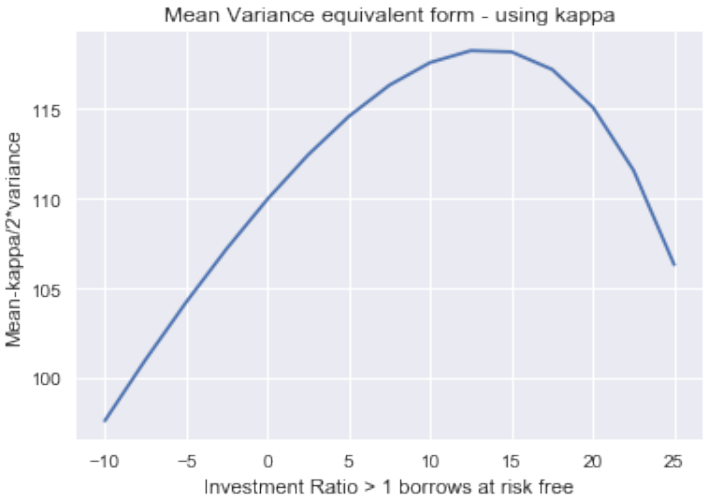


Figure 4.3: MV equivalent approx to Log-Utility, $\kappa = 0.006$

4.1.3 Mean-Variance equivalent distributions, Step-wise

To recap we have moved from Utility curves, to end of episode rewards, to step-wise rewards suitable for a myopic utility such as log-utility, to mean-variance equivalent distributions. If one examines what has been done so far, there needs to be a final step in the implementation to move onto step-wise rewards for mean-variance equivalent distributions, in which case we have completed the pre-requisites of an environment suitable for a reinforcement learning agent.

Here I again follow Ritter's [25] example and indeed it was on reading his paper and implementation that I felt that his technique may be suitable for applying reinforcement learning to the Merton problem.

If we look at the mean-variance equivalent form Ritter used, we have that:

$$\max_{\pi}[\mathbf{E}(w_T) - \frac{\kappa}{2}\mathbf{V}(w_T)]$$

Thus we need to calculate $\mathbf{E}(w_T)$ and $\mathbf{V}(w_T)$, in a step-wise reward fashion. The expected value is easy:

$$\mathbf{E}(w_T) = w_0 + \sum_t \mathbf{E}(\delta w_t)$$

Where δw_t is the change in the agent's wealth in one time-step at time t. For the variance if δw_t is independent of δw_s when $t \neq s$, then we have that:

$$\mathbf{V}(w_T) = \sum_t \mathbf{V}(\delta w_t)$$

Thus the problem we need to solve has been reformulated as something far closer to something that can provide step-wise rewards, that is:

$$\max_\pi \sum_t [\mathbf{E}(\delta w_t) - \frac{\kappa}{2}\mathbf{V}(\delta w_t)]$$

If we define our reward as follows:

$$r_t = \delta w_t - \frac{\kappa}{2}(\delta w_t - \hat{\mu})^2$$

where $\hat{\mu}$ is the estimate of the change in our wealth over one time-step. Then:

$$\mathbf{E}(r_t) = \mathbf{E}(\delta w_t) - \frac{\kappa}{2}\mathbf{V}(\delta w_t)$$

which is exactly what is required. However as Ritter [25] points out, there is a circularity here in that we do not know $\hat{\mu}$. But clearly unless the Sharpe ratio of the strategy is very high and given small enough steps then we have a very close approximation:

$$\mathbf{E}(\delta w_t - \hat{\mu})^2 \approx \mathbf{E}((\delta w_t)^2)$$

The added benefit is that this is conservative as regards risk (because we are slightly over-estimating variance based upon a zero expected change in wealth). Again if one does wish to estimate $\hat{\mu}$, then we can train our reinforcement learning agent using the above formulation and after convergence, use the samples to estimate $\hat{\mu}$ and then simply retrain again including this estimate.

We have now solved the problem of creating step-wise rewards without need-

ing myopia.Specifically if our reward at time t:

$$r_t = \delta w_t - \frac{\kappa}{2}(\delta w_t)^2$$

Then in a reinforcement learning setting where we wish to maximise the sum of rewards, we find that:

$$\begin{aligned}\mathbf{E}(G_t) &= \mathbf{E}(r_{t+1} + r_{t+2} +, \ldots, r_T) \\ &= \sum_{s=t}^{T} \mathbf{E}(\delta w_s) - \frac{\kappa}{2}\mathbf{V}(\delta w_s)\end{aligned}$$

This is exactly what is needed for a step-wise reward formulation equivalent to maximising the terminal utility of wealth.

I tested this implementation for a variety of fixed investment strategies and examined the mean rewards in order to check that this does indeed give a similar optimal investment to the analytical Merton solution, figure 4.4 is an example showing that this is indeed the case.

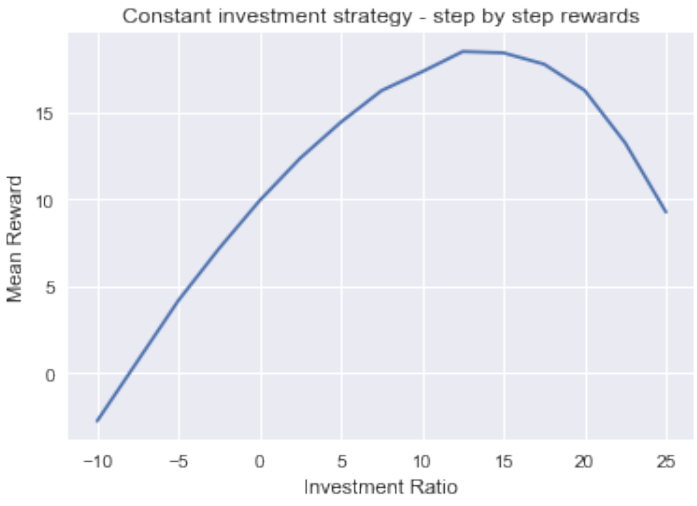


Figure 4.4: Mean Variance equivalent approximation to log-utility

We now have a step-wise formulation of an environment with a parameter κ to match the risk aversion to the utility of the investor. Given that this formulation

works for log-utility, we can adjust our κ for different utility curves. One extreme would be a risk neutral wealth maximiser, that is $\kappa = 0$. We should hope to see a linear utility (approximated by mean rewards) as the investor levers up. Figure 4.5, shows that this is indeed the case.

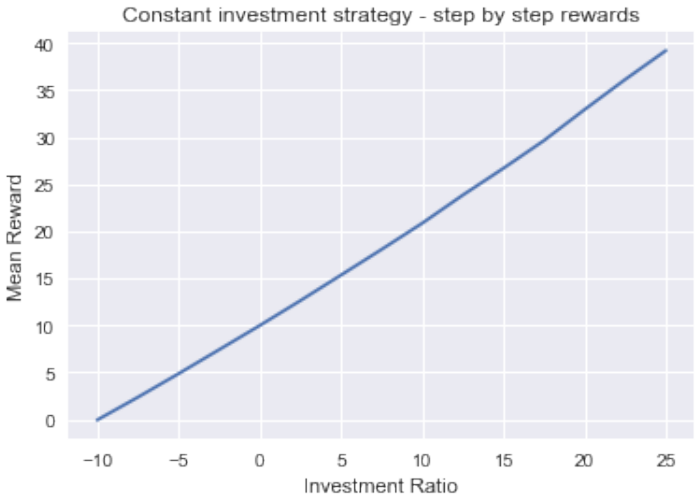


Figure 4.5: $\kappa = 0$, does this match risk neutrality?

4.1.4 Reward Shaping Conclusion

To conclude, we can now generate rewards from a market environment in a step-wise manner suitable for an RL agent. We first looked at end of episode utilities, then derived a simple step-wise reward for log utility, then generalised to a mean-variance formulation on an end of episode basis and finally a mean variance approximation on a step-wise basis. Given a correct parameterisation of the risk aversion parameter κ, this should enable an RL agent to learn the optimal policy for a variety of stochastic processes and utility curves.

4.2 Design

The goal is to test if a reinforcement learning agent can learn to trade at or close to Merton optimality. In terms of design I created an episodic environment for training and test reporting purposes. Note that for the RL agent this is not actually necessary as the agent learns online step by step (it does not require reaching the end of an episode to learn, unlike for example Monte Carlo learning) and indeed

there is nothing special about the end of an episode in terms of rewards, thus the set-up can easily be reformulated into an on-going rather than episodic environment.

By creating an episodic environment I can produce distributions of test results for my agent. I will test a trained RL agent, against the Merton optimal agent and a random agent. This will of course be on an unseen test set. Because we are dealing with stochastic processes, examining just one realisation of that time series will not give definitive answers. For example in any given episode a random agent can easily out-perform a Merton optimal agent, however if we rerun multiple times then this will not be the case and we can examine the test distribution of results.

From an RL viewpoint the design comprises states which comprise 'buckets' of wealth. The agent is a price taker and so does not influence the future market price at all. The agent's action at each step is to choose a proportion of his wealth in the risky asset and similarly the risk-free bond (he also has the choice to lever up and borrow money and also to go 'short'). The agent will receieve a 'reward' from the environment at each step and find himself in a new state provided by the market environment.

4.2.1 Market Environment

The market environment can be interacted with on a step-wise basis by an RL agent. It comprises two financial instruments. A risk-free bond and a risky asset (for example geometric brownian motion, ornstein-uhlenbeck etc.). The market environment is episodic, thus comprises multiple time periods T (a parameter we can of course vary). After each episode concludes we begin again with a new realisation of the stochastic process, bond and of we course reset the agent's wealth level.

4.2.2 Agent

The agent is a reinforcement learning algorithm. The agent at any given time is in a certain wealth state, depending on the information available (prices, and other features) we can of course add whatever information we wish to the current state. The agent knows nothing about the underlying price process of the market, nor the type of stochastic process or any parameters of its parameters, similarly it knows no details about the risk-free bond.

The agent can only interact with the environment by providing an order, this order based on the agent's current wealth with be received by the market and executed, the order comprises an instruction to invest a certain amount of the agent's wealth in the risky asset and a certain amount in the risk-free bond. After receiving the order, the market will take a step, returning to the agent a new wealth state (by first moving to a new price state for the risky asset and the bond), it also returns a reward. The agents goal is to use these rewards to attempt to learn to trade at or close to Merton optimality.

The reinforcement learning agents were quite plain vanilla: Q-Learning, Sarsa, Double-Q Learning, and Dyna. The initial goal was not to over-complicate the RL side of the implementation, where obvious extensions are easily added. As mentioned earlier the agents had states which comprised wealth buckets and the actions were to invest a certain amount of their stock at each time step into a risky asset of a risk-free asset. The interaction with the environment was through these actions.

My state space was 100-150 wealth buckets, with 15 actions covering a wide variety of investment possibilities at each time step from -10x short to 25x long the risky asset in the most extreme example. Thus, despite the reduced size, the state-action space still covered over 1500 possibilities, with multiple time steps. The number of time-steps for each episode was 20 steps, however 50, 100 or more time steps was also viable, but just slower to train. Even over 20 time steps there are 15^{20}

possible policies for the agent to decide between, each episode.

4.2.3 Noise intuitions - Merton and Random agents episodically

I will first illustrate how I modelled rewards approximating terminal utilities with a specific example. All simulations performed in this section (to test the correct implementation of the market environment and the associated mathematics in going from an episodic end utility to a step-wise reward) were performed with 200,000 simulations.

Before moving on to the test methodology, I will describe an initial issue and how it informed my test metrics.

Perhaps naively, I first thought that the Merton problem would be solved quite cleanly given Ritter's [25] formulation of the mean reverting trading problem for Q-Learning.

I had slight concerns that the set up was part-way between a bandit setting and a multi-state RL setting. The concern was that the agent has little control of much of his environment, he is simply a price taker. However, the agent can potentially control the future distribution of his utility of wealth.

After the initial implementation the trained agent performed poorly. Hence the reason for reducing the state space dimensionality to only 100 buckets of wealth and only 15 investment actions. I began with 1000 time steps for each episode (to minimise the impact of discretisation), and eventually reduced to 20 (due to training time on a 3Ghz laptop). Even after these simplifications there were learning problems, there are hyper-parameters to adjust and some time was wasted there - but the issue was more basic: noise.

I reduced the volatility of the geometric brownian motion. This has the disadvantage that we move further again from a real market problem and instead shift to

the question of how much should I borrow to invest in this now not very risky stock. With a far lower volatility than one would find in an actual market the Q-learner began to learn.

The crux of the problem is shown in Figure 4.6. The left chart shows the distribution of terminal utilities, the right the distribution of terminal rewards. This is on an episodic basic (in the low volatility setting described earlier). What is difficult to see is that I am actually showing rewards and utilities for two extremely different strategies, Merton optimal and a random agent, i.e. the distributions for the best performer and the worst are actually very similar on an episodic base for this particular example. Given this fact it is unsurprising that an RL agent will take many training episodes to learn anything better than random.

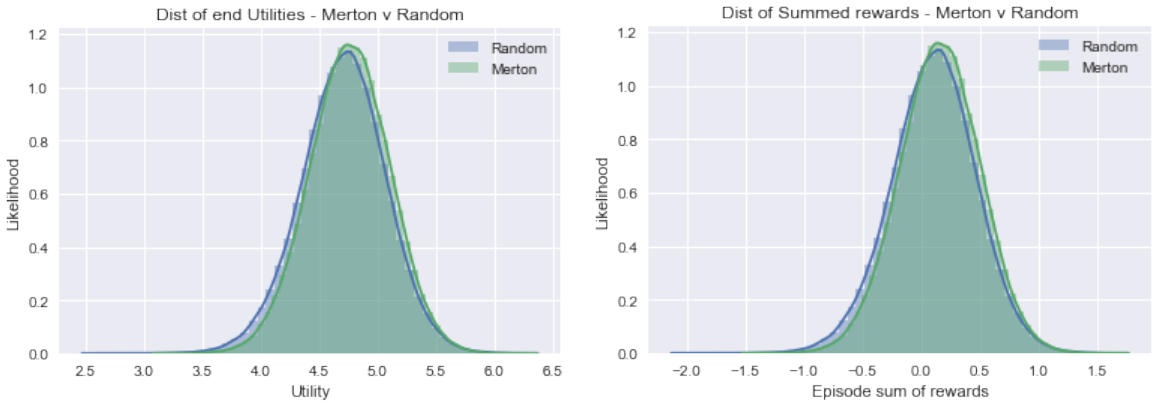


Figure 4.6: Episodic distribution of final utilities and rewards, Random v Merton

Recall that the agent does not even receive the benefit of episodic rewards, the rewards are actually received on a step-wise basis. Figure 4.7 illustrates the step-wise rewards for a random agent compared to a Merton optimal one, again the differences are very minimal.

It became clear that I needed to train the RL agent over several million episodes rather than several thousand and also that simply examining the average of mean

Figure 4.7: Stepwise rewards Merton optimal versus a Random agent

utilities over episodes on a test set would not clearly differentiate between optimal skill and randomness.

4.3 Test Metrics

4.3.1 Utilities

As mentioned above, my first thought of comparing the difference in episodic utility between a trained agent and Merton optimal looked difficult for this specific problem due to the very slight differences on an episodic basis even between a random agent and a Merton optimal one.

I therefore decided to examine the performance not on an episodic basis, but grouped over 1000 episodes. I would then batch another 1000 episodes (so they are independent) and repeated this process 3000 times. Thus 3,000,000 episodes in 3,000 batches. Figure 4.8 shows the difference in performance between a Merton optimal agent and a random agent using this revised test metric.

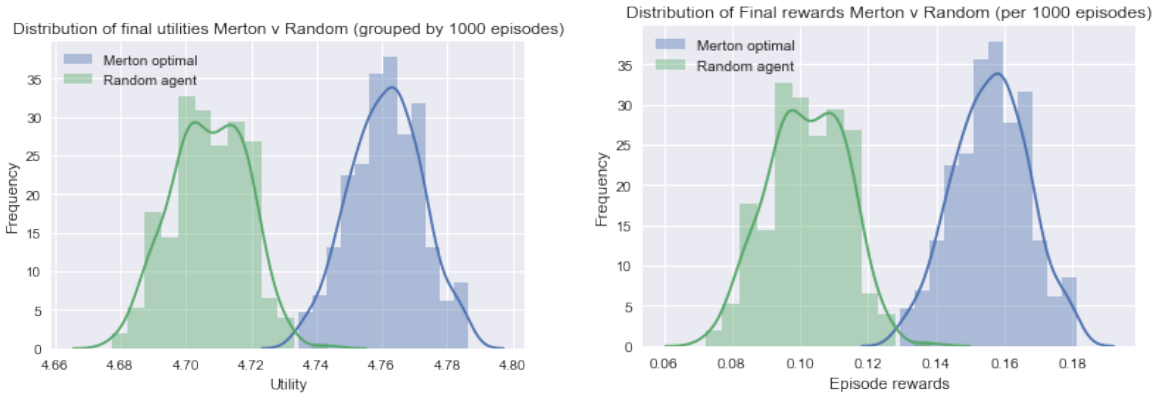


Figure 4.8: Batched utilities and rewards, random agent versus Merton

We can now see a clear differentiation in performance over both rewards and utilities between a random agent and a Merton optimal one. For test purposes the Random, Merton optimal and Q or Double-Q learner were tested using the above methodology with the Q-learner acting greedily with respect to the trained Q-values.

4.3.2 Rewards

The right hand chart of figure 4.8, shows the differences on the same batched basis between rewards for a random agent and a Merton optimal. One would hope that this looks very similar to the utility chart if the rewards are matching utilities well, which in this case it does. However it may be the case that the mean-variance approximation is a poor approximator of utilities, in which case the agent is being trained towards in inappropriate policy. Therefore I also included rewards as another test metric to shed further light on the performance of the RL agent.

4.3.3 Wealth

The question, 'Why not just show who made the most money?', is absolutely the wrong question. As an ex-fund manager, it is a slip I frequently make. The object of this game is to learn to trade close to a Merton optimal manner, which involves learning a policy that maximises the expected utility of terminal wealth. Not terminal wealth itself. They are only the same if we are risk neutral (and thus have a linear utility curve). However examining the distribution of end episode wealth remains interesting, particularly when we wish to examine a risk neutral verses a

risk-averse agent.

4.3.4 Sharpe Ratio

The Sharpe ratio $\frac{\mathbf{E}(r-r_f)}{\sqrt{\mathbf{V}(r)}}$, where r is the return of our strategy, r_f and risk free rate is a well known metric in finance named after the Nobel Laureatte William Sharpe. I chose a simpler variation $\frac{\mathbf{E}(r)}{\sqrt{\mathbf{V}(r)}}$, where for each episode I calculated the standard deviation of return and episode return, in order to build a distribution of episodic 'Sharpe' ratios.

I used the same batching method as above with the same 3,000,000 test set episodes for each of the three agents (random, Merton and trained RL). The reason for reporting the distribution of Sharpe ratios was twofold. First this is a common and accepted metric in finance to examine the performance of strategies on a risk adjusted basis (admittedly this is usually on an annualised rather than per-episode basis), and second because my approximation using mean-variance equivalence is a step-wise risk return trade-off and I wished to examine any cases where the agent actually did very well on a Sharpe ratio basis, but less so as regards final utilities. One case might be learning extreme risk aversion and the agent hiding out in the risk-free bond.

Chapter 5

Experiments

In this section I will describe the experiments performed. Starting from the simplest formulation using exact log-utility rewards and a a very low volatility geometric brownian motion, to more realistic higher volatility examples with 'fat tails'.

In the following experiments I trained Q and Double-Q learners, using 3,000,000 training episodes with a step-size of 0.1 and ε- greedy exploration of 0.1. γ was set to 0.95. The test methodology was as described earlier using 3,000 batches of 1,000 episodes, thus comprising an unseen testset of 3,000,000 episodes. T was set to 20 periods, purely for convenience as regards training time, (50, 100 or even more time periods had similar results but of course longer training times).

In each case test distributions will be produced showing where appropriate, utilities, rewards, wealth and sharpe ratios. The 'straw man' random agent will be compared to Merton optimal, as well as the trained RL agent.

A key reminder for each of these tests is that to derive the Merton ratio analytically, we first need to know that the underlying process is in this case a geometric brownian motion, we also need to know its parameters μ, σ and the risk-free rate r as well as the fact that the utility is log-utility. In contrast the Double-Q learner, knows none of the above and only has raw experiential data to learn from.

5.1 Experiment group 1: Learning to trade like Merton, Double-Q

A double-Q agent was trained to optimise the terminal log-utility of wealth. For the first test I used the exact myopic formula for step-wise rewards that I derived at the beginning of Chapter 3 on reward shaping (not the mean variance equivalent approximation used by Ritter [25]).

5.1.1 Log-Utility

5.1.1.1 Log-utility: Description

The agent had 15 choices over a wide range for each of 20 time-steps, from investing -10x in the risky asset, to 25x leveraged. The agent could also choose how much of its current level of wealth to invest in a risk-free bond at each time step. The risky asset was a geometric brownian motion with a drift $\mu = 0.11$, and a low volatility $\sigma = 0.028$, the risk free bond had a rate of interest $r = 0.1$.

In this particular experiment we have a very small difference between the drift of the risk-free rate and the GBM. The GBM is also very low volatility, the solution from a Merton optimal viewpoint would be to lever up to 12.5x one's wealth at every time-step and invest all of this in the risky asset. The differences in the rewards between Merton optimal and random are also very small as mentioned above.

5.1.1.2 Log-utility: Results

Results are given in figure 5.1, the top row of charts shows the distributions of test utilities as well as a violin plot of the same. The Merton optimal agent as one would expect has a higher expected utility than the random agent, with the distribution further to the right. The double-Q learner is far better than random but not at the level of the Merton optimal agent. It has certainly learned to do way better than random, but is not perfect.

The second row shows the rewards, in this case the results are identical as I am using my exact formula for log-utility step-wise rewards and there is no approximation. The third row gives an interesting aside, the wealth distribution of the double-Q learner is better even than the Merton optimal agent. If one examined results only from this naive viewpoint then one would argue that the Double-Q learner has outperformed even the Merton agent. However, this comes at a cost.The final row gives the distribution of Sharpe ratios for the three agents. The Merton optimal agent has the highest Sharpe ratios, thus showing that the strong final wealth results of the Double-Q learner came at the cost of taking on greater volatility, damaging both Sharpe ratios and final log-utilities.

5.1.1.3 Log-utility: Conclusion

Figure 5.2 shows the 10,000 episode moving average of the utilities on the test set. With table 5.1, showing the means of standard deviations of the various test statistics.

In conclusion I would argue that the double-Q learner has not quite matched the performance of a Merton optimal agent in this particular case, but still has a very good performance, certainly way better than a random agent. The distribution of utilities and rewards is slightly worse than Merton optimal, with a slight surprise that the distribution of wealth is actually higher. Thus the agent has learned to generate more wealth even than the Merton optimal agent, but at the cost of higher volatility as evidenced by a lower sharpe ratio and log-utility.

Due to the stochastic nature of the risky asset as well as double-Q learning itself having stochasticity embedded within its exploration, it is important to note that experimental results can also vary slightly. Notice also in table 5.1 that the utility distribution and rewards distribution are basically identical, indeed if we added the utility of the initial wealth, $log(w_0 = 100)$ to the sum of the rewards then they

would be exactly the same. This is of course due to my being able to use my exactly derived step-wise reward in the case of log-utility, rather than the mean-variance equivalent approximation.

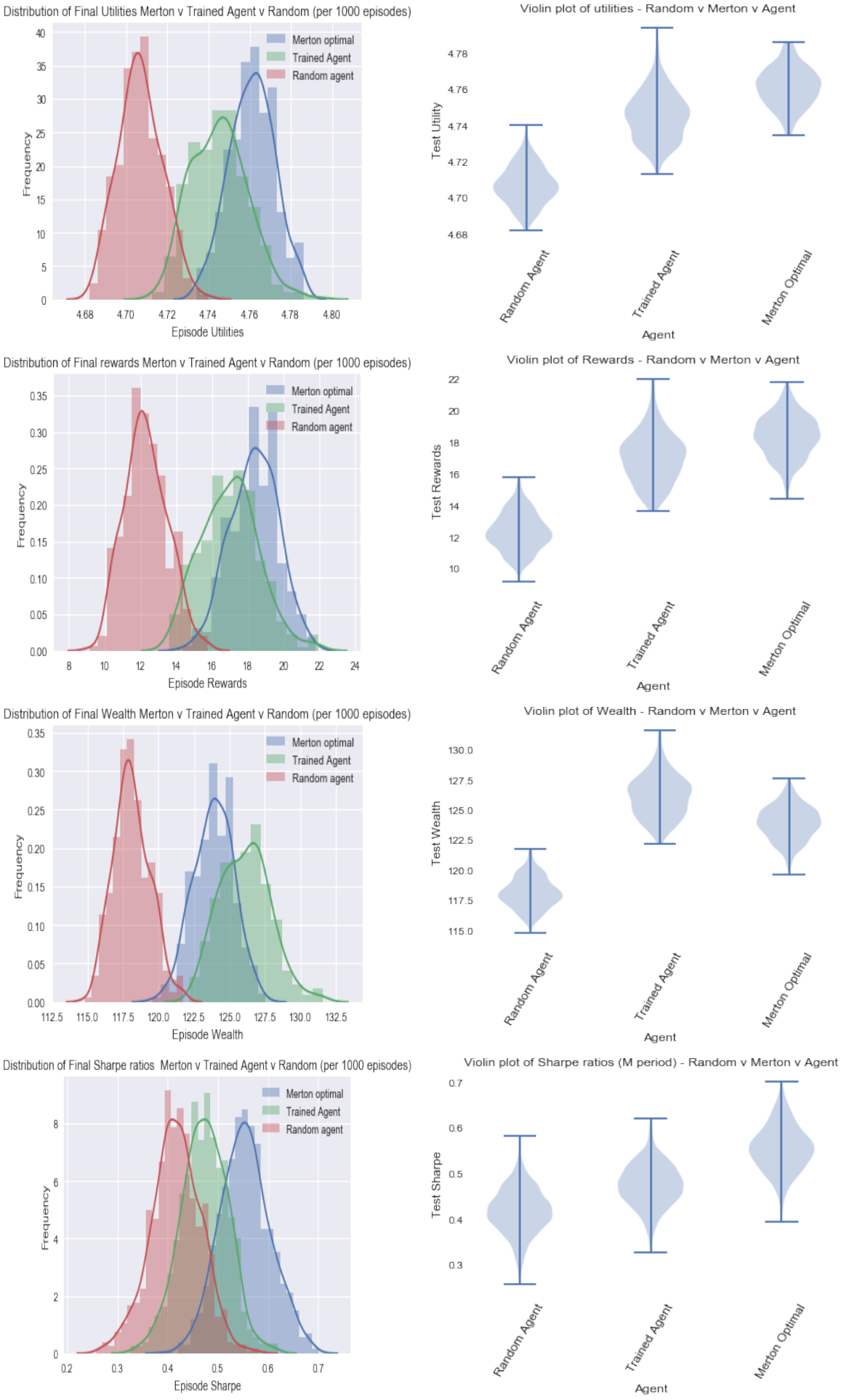


Figure 5.1: Double-Q learning: Utilities verses Merton and Random agent

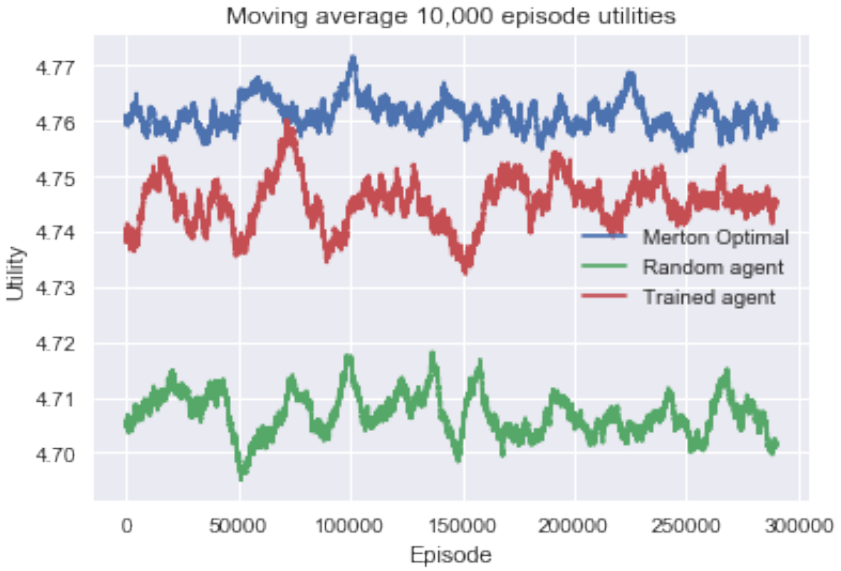


Figure 5.2: Moving average utilities Double-Q v Merton v Random

	Random Agent	Double-Q	Merton Optimal
Utility Distribution	4.7062	4.7442	4.7611
Rewards Distribution	0.1010	0.1390	0.1559
Wealth Distribution	118.07	125.99	123.87
Sharpe Ratio Distribution	0.4145	0.4708	0.5544

Table 5.1: Double-Q, test performance v Merton - Mean

	Random Agent	Double-Q	Merton Optimal
Utility Distribution	0.0118	0.0138	0.0108
Reward Distribution	0.0118	0.0138	0.0108
Wealth Distribution	1.3946	1.7390	1.4035
Sharpe Ratio Distribution	0.0493	0.0459	0.0501

Table 5.2: Double-Q, test performance v Merton - Std

My interpretation is that these results are good, but by no means perfect. The agent still hasn't performed quite at the level of a Merton optimal agent utility-wise. It is true that it has outperformed Merton in terms of the terminal wealth distribution, but this is only at the cost of taking on greater risk, as evidenced by the lower Sharpe ratio. Before generalising to mean-variance equivalence I shall now test the double-Q learner on a higher volatility problem, both to add greater realism

and to add comfort that the performance on this particular problem was not some lucky coincidence.

5.1.2 Increasing the Brownian Motion volatility

5.1.2.1 Brownian Motion volatility: Description

Each parameter in this new problem formulation is exactly the same bar one - volatility, for our risky asset we remain with a geometric brownian motion with a drift $\mu = 0.11$ and a risk-free asset with $r = 0.1$, the volatility of the GBM σ has now increased nearly fourfold however, from 0.028, to 0.1. This will of course reduce the merton optimal solution.$\frac{\mu - r}{\sigma^2}$. The solution in Merton optimal terms is to remain fully invested with a constant 100% investment in the risky asset, to not lever up further, not go short and not reduce risk by owning the risk-free bond.

Figure 5.3, illustrates some sample paths for the new stochastic process. Because the process is more volatile we are likely to experience a wider range of price states in the market and thus wealth states for our RL agent, however the agent remains with the action space given above, that is 15 possible investment actions at each time step. However because the merton optimal solution for the investment ratio has now moved considerably (from 12.75x to 1x), I have changed the actions to now range from being short 100%, to long 2.5x. This is far closer to more realistic leverage ranges that many hedge funds might use and feels less contrived than the previous experiment.

5.1.2.2 Simulated utilities

Figure 5.4, shows the results achieved by simulating the environment 200,000 times with a range of constant investment strategies. The left chart shows that the peak episodic log-utility simulated matches the theoretical merton optimal ratio of investment (i.e. 1x). The right chart shows that the conversion of this into step-wise rewards run under the same simulation gives a very close match. Thus giving confidence that a reinforcement learning agent will be able to learn from the step-wise rewards.

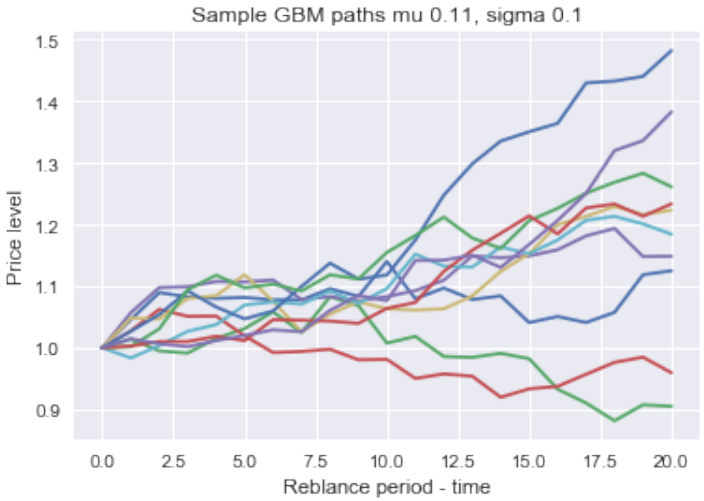


Figure 5.3: Medium Volatility Geometric Brownian Motion Paths

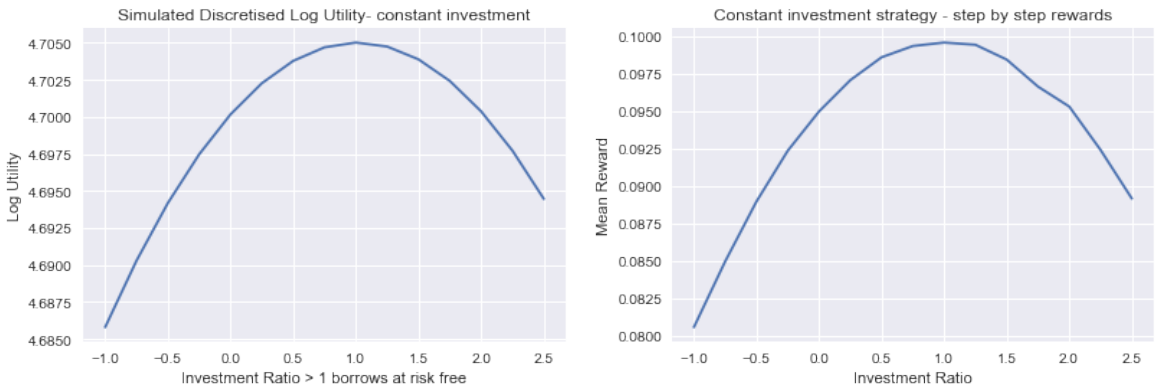


Figure 5.4: Discretised Log Utility versus Step-wise rewards - medium vol

5.1.2.3 Merton optimal v Random - rewards

In a similar fashion to the earlier low volatility problem, figure 5.5 shows the comparison of the actual rewards for a Merton optimal versus a random agent. Compared to the previous example the higher volatility environment has now resulted in the two distributions overlapping somewhat, thus possibly making this a tougher environment for the RL agent to learn.

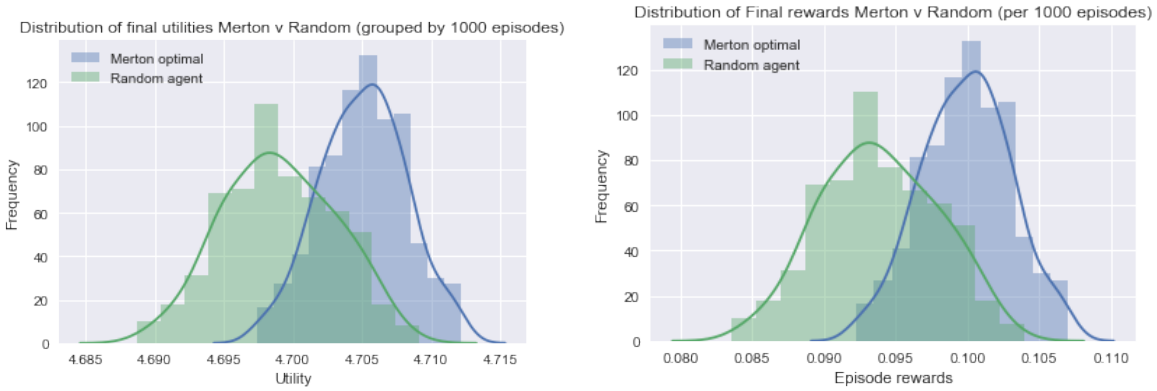


Figure 5.5: Distribution of batch Utilities Merton optimal v Random - medium vol

5.1.2.4 Brownian Motion volatility: Conclusion

Figures 5.6, 5.7 and table 5.3 illustrate the results of the higher volatility experiment. Because I am again using the exactly derived log-utility rewards (not the mean-variance approximation) the rewards results are left out (due to being superfluous).

I consider these particular results to be excellent. On the test set the distribution of utilities and wealth is very close indeed to the merton optimal agent. Similarly for the moving average of utilities. The double-Q agent is close to indistinguisable from the merton optimal agent bar the fact that it experienced a greater volatility of wealth on the test set resulting in a slightly lower sharpe ratio.

The agent appears to have successfully approximated the performance of a merton optimal agent in a reasonably difficult case where volatility is reasonably high and the relative drifts of the risk free and risky asset are fairly similar, resulting in distributions of rewards from actions that are very close. I will now move away from the exact log-utility reward formula towards to mean-variance approximations in order to generalise to a wider range of utility functions.

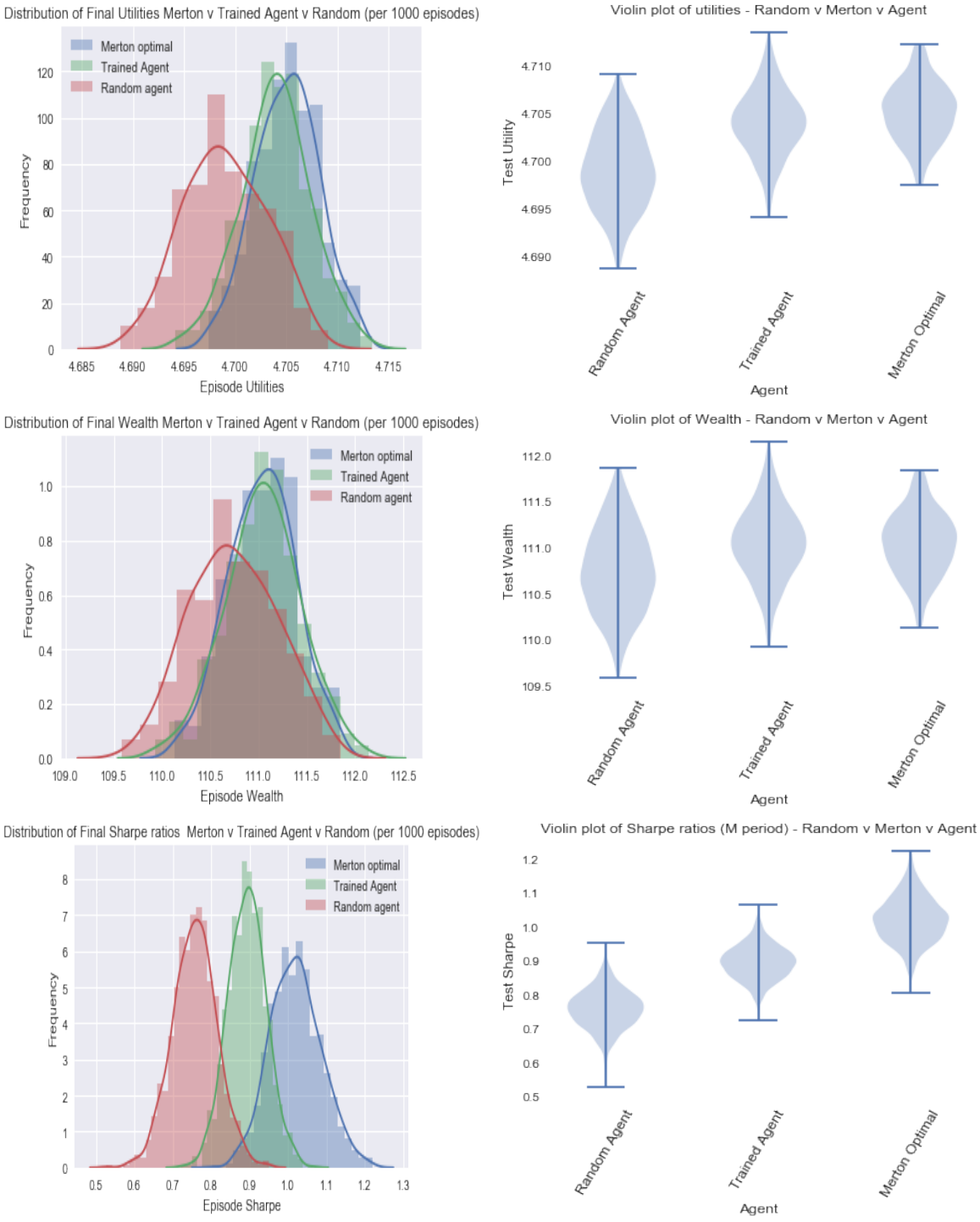


Figure 5.6: Double-Q learning: Utilities verses Merton -Medium Volatility

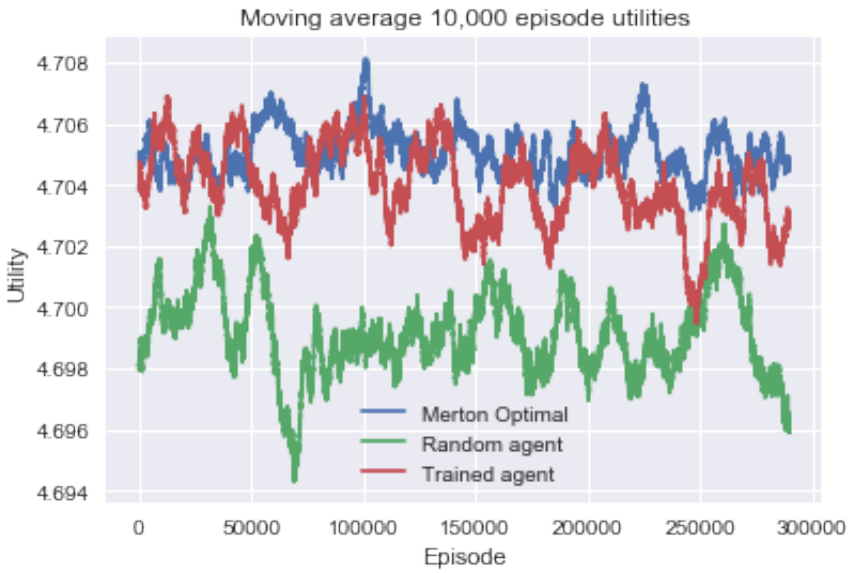


Figure 5.7: Moving average utilities Double-Q v Merton - medium volatility

	Random Agent	Double-Q	Merton Optimal
Utility Distribution	4.6990	4.7043	4.7050
Wealth Distribution	110.74	111.09	111.03
Sharpe Ratio Distribution	0.7574	0.8961	1.0202

Table 5.3: Double-Q, test performance v Merton - Mean, Medium vol

	Random Agent	Double-Q	Merton Optimal
Utility Distribution	0.0040	0.0034	0.0030
Wealth Distribution	0.4545	0.3963	0.3467
Sharpe Ratio Distribution	0.0581	0.0509	0.0665

Table 5.4: Double-Q, test performance v Merton - Std, Medium vol

5.1.3 Generalising to Mean-Variance rewards

5.1.3.1 Mean-Variance rewards: Description

For log-utilities I derived an exact step-wise reward function for the reinforcement learning agent, however in general this will not be possible. In this experiment I will still use log-utility and the higher volatility environment, but now introduces the mean-variance approximation described earlier and used by [25]. I will continue to use the double-Q learning agent.

This also introduces an additional risk aversion parameter $\kappa = 0.006$, parameterised earlier in this thesis. The key idea of κ being that as described by [25], that is the optimal policy of any upward sloping utility curve will match the optimal policy for some κ. From an investor viewpoint κ will also enable us to tailor our risk aversion and train the RL agent for differing risk preferences, as I shall test in a future experiment.

5.1.3.2 Mean-Variance rewards: Conclusion

Figures 5.8, 5.9 and table 5.5 shows the results using a mean-variance reward approximation for log-utility. Again I consider these results to be excellent. On the test set the agent has learned to closely match the performance of the merton optimal agent for utilities, and the final distribution of wealth. By introducing κ I am now approximating rewards and the agent is directly targeting variance adjusted changes in wealth.

Thus in this particular experiment the double-Q agent has closely matched the test utility performance of a merton optimal agent, but has a better sharpe ratio than the merton optimal due to its exhibiting a lower volatility of final wealth on the test set. Given this formulation we are now able to adjust our risk preference κ and indeed use this to train the RL agent to approach merton's problem for any sensible utility curve. I shall test changes to κ in the next experiment.

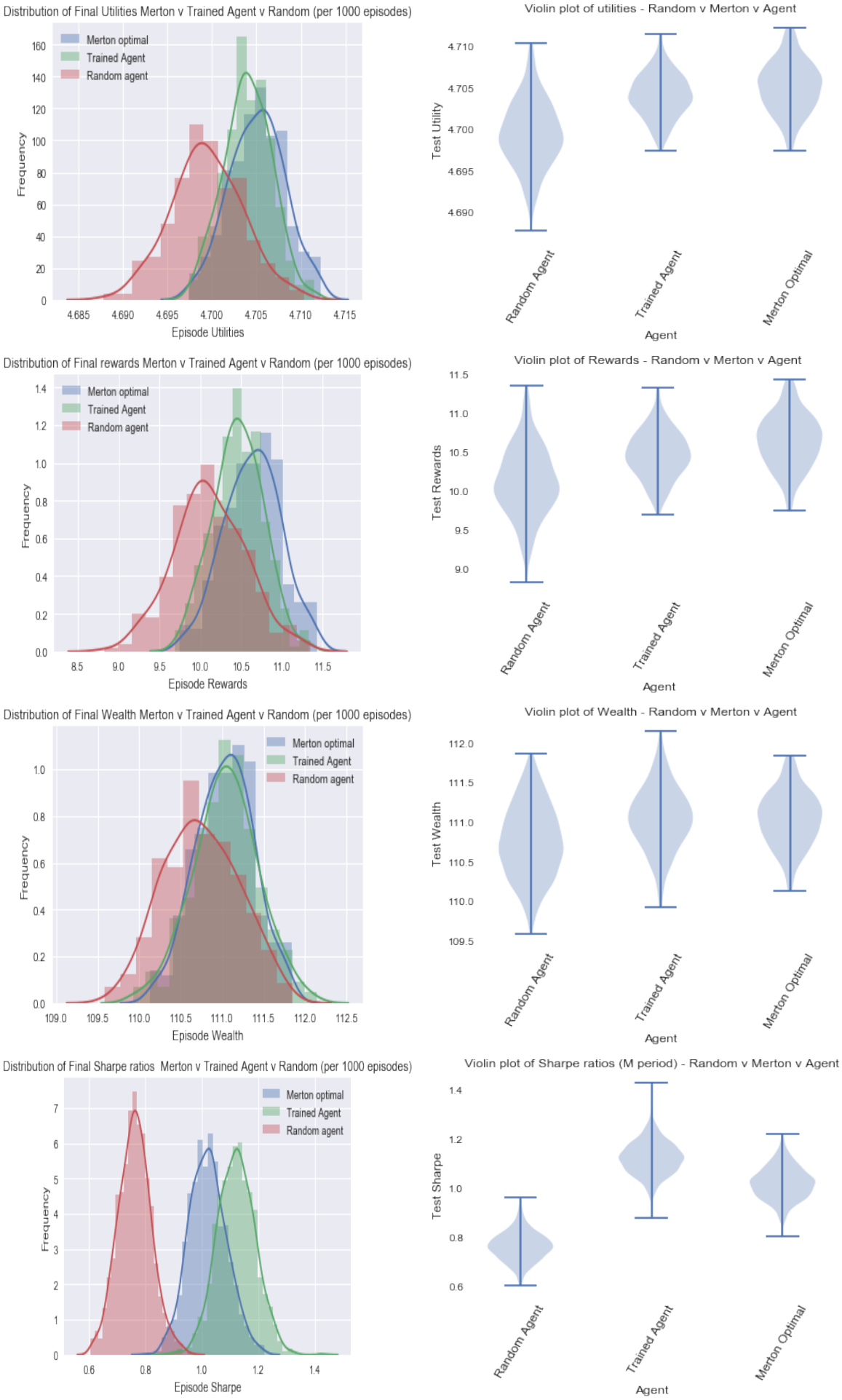


Figure 5.8: Double-Q learning: Utilities, MV-equivalent

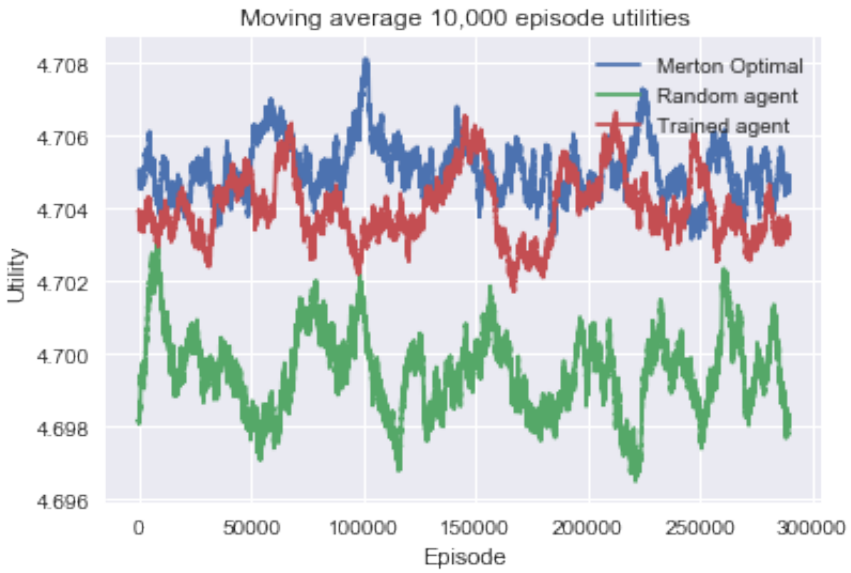


Figure 5.9: Moving average utilities Double-Q, MV-equivalent

	Random Agent	Double-Q	Merton Optimal
Utility Distribution	4.6994	4.7040	4.7050
Rewards Distribution	10.11	10.46	10.63
Wealth Distribution	110.78	111.01	111.03
Sharpe Ratio Distribution	0.76	1.12	1.02

Table 5.5: Double-Q, MV equivalent, test performance - Mean

	Random Agent	Double-Q	Merton Optimal
Utility Distribution	0.0040	0.0026	0.0030
Reward Distribution	0.4389	0.3070	0.3439
Wealth Distribution	0.4441	0.3147	0.3467
Sharpe Ratio Distribution	0.0586	0.0673	0.0665

Table 5.6: Double-Q, MV equivalent, test performance - Std

5.1.4 Power Utility

In Chapter 2, I introduced power utility and the merton optimal solution for a geometric brownian motion. The is more general than log-utility. The key point from the point of view of this thesis is that this could be one of any number of utility functions we can approximate through the mean-variance approach to merton's problem.

The equation and solution for power utility is given below:

$$U(x) = x^{\gamma}\,, 0 < \gamma < 1$$

$$u^* = \frac{\mu - r}{\sigma^2(1-\gamma)}$$

In experiment 5, the parameters of the experiment are the same as for the higher volatility environment. I set $\gamma = 0.1$ and approximate my rewards using mean-variance.

5.1.4.1 Power Utility conclusion

I feel the results for this experiment are certainly way better than for example a random agent, but also less conclusive than those given previously. For this test run however even the merton optimal agent had a very wide distribution of outcomes on the test set. The double-Q agent tests exhibited a distribution of utilities close to the merton optimal agent, but with far lower variability. Thus, the RL agent compared to merton had differing behaviour as regards the final wealth distribution and so the sharpe ratio distributions were also different. It appeared to learn behaviours that gave a lower expected wealth, but with much lower volatility, hence the high sharpe ratio. In this case from the purist viewpoint of learning optimal merton for power utility, then in this experiment that did not happen, however the agent certainly learned behaviours that showed good performance on both utilities and sharpe ratios.

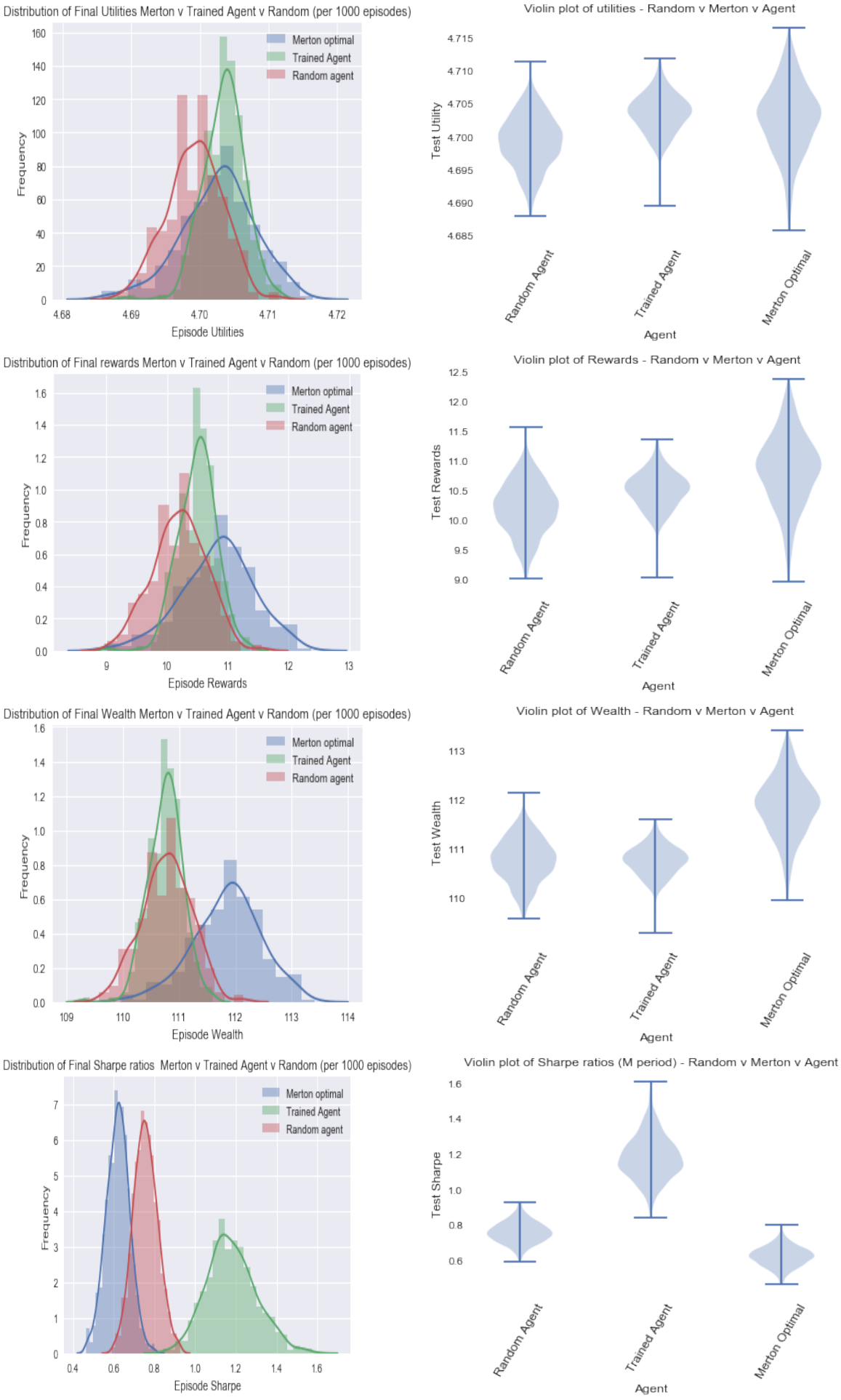


Figure 5.10: Double-Q learning: Utilities verses Merton, power utility

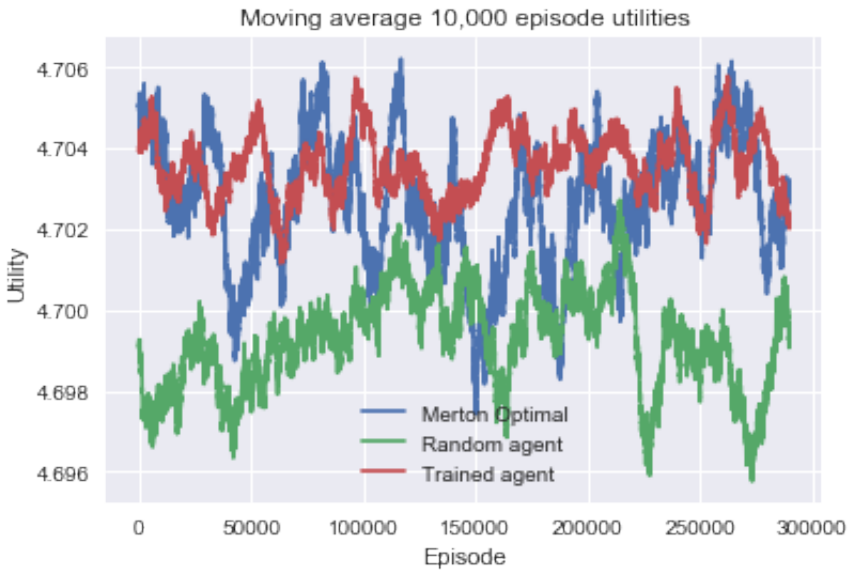


Figure 5.11: Moving average utilities Double-Q v Merton, power utility

	Random Agent	Double-Q	Merton Optimal
Utility Distribution	4.6995	4.7038	4.7050
Rewards Distribution	10.22	10.52	10.69
Wealth Distribution	110.79	110.78	111.03
Sharpe Ratio Distribution	0.76	1.18	1.02

Table 5.7: Double-Q, test performance v Merton - Mean, power utility

	Random Agent	Double-Q	Merton Optimal
Utility Distribution	0.0118	0.0138	0.0108
Reward Distribution	0.0118	0.0138	0.0108
Wealth Distribution	1.3946	1.7390	1.4035
Sharpe Ratio Distribution	0.0493	0.0459	0.0501

Table 5.8: Double-Q, test performance v Merton - Std, power utility

5.1.5 Learning risk aversion κ

In experiment 4 I shall test the effect of varying the risk aversion parameter κ, one would expect that for a particular κ we can train differing risk behaviours into the RL agent. I shall test the behaviour of the agent including two extremes - risk neutrality and extreme risk aversion. One would hope that in the case of risk neutrality that the agent might become a pure wealth maximiser and indeed even exceed the wealth of a merton optimal agent, albeit at the cost of risk.

In the case of extreme risk aversion, one might expect that the agent shies away from any leverage and any investment in the risky asset. Instead preferring to be 100% invested in the risk-free bond.

5.1.5.1 Learning risk aversion: Risk Neutrality

Figure 5.12 illustrates the utility gained by simulating various fixed investment ratios. In the case where $\kappa = 0$ as one might expect, the curve is linear and the maximum reward is gained by taking on as much leverage as possible and putting this into the risky asset.

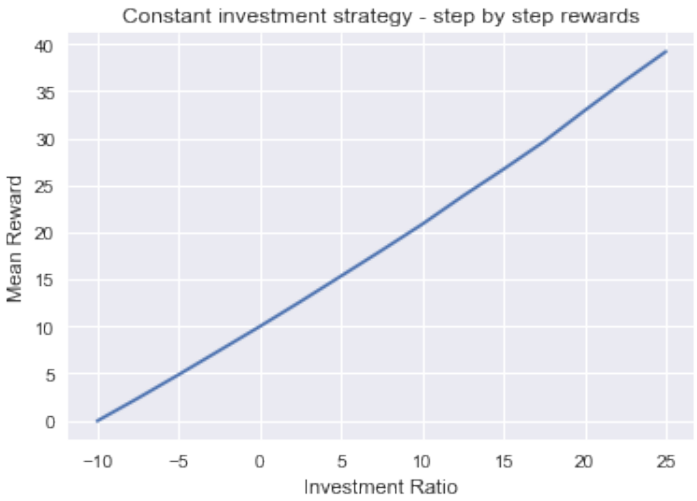


Figure 5.12: $\kappa = 0$, Risk neutrality

5.1.5.2 Risk Neutrality results

Figure 5.13 and tables 5.9, 5.10 illustrate the distributions of weath and sharpe ratios for the agent trained towards risk neutrality.

We can see the the trained double-Q agent indeed outperforms even the merton optimal agent as regards the test set wealth distribution, however this comes at a considerable cost. The sharpe ratio is drastically reduced due to the higher variability of final wealth. Thus, the agent behaves exactly as intuition might suggest when trained towards risk neutrality.

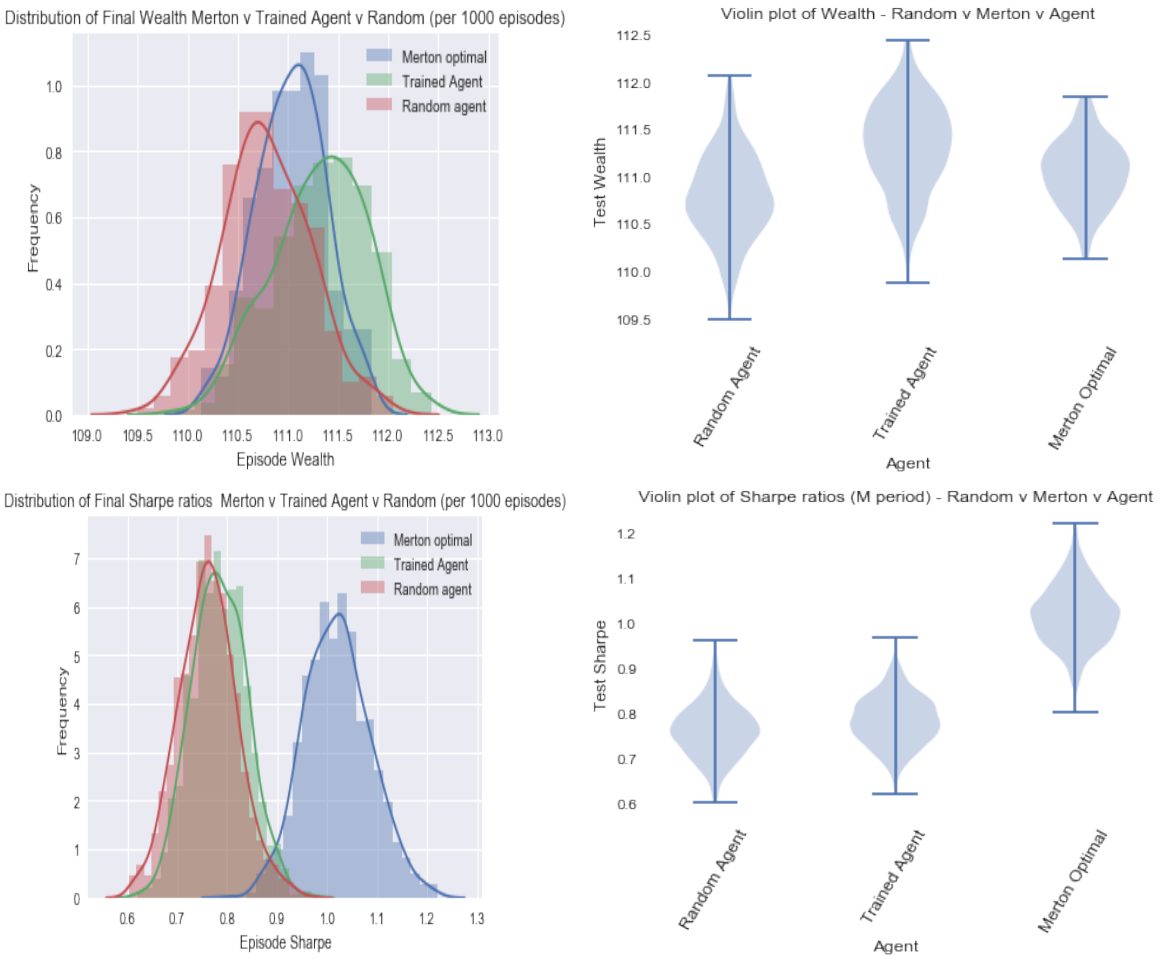


Figure 5.13: Double-Q learning: Wealth Distribution v Merton, Risk Neutral

	Random Agent	Double-Q	Merton Optimal
Wealth Distribution	110.78	111.32	111.03
Sharpe Ratio Distribution	0.7615	0.7842	1.0202

Table 5.9: Double-Q, Risk Neutral agent, test performance - Mean

	Random Agent	Double-Q	Merton Optimal
Wealth Distribution	0.4441	0.4683	0.3467
Sharpe Ratio Distribution	0.0586	0.0552	0.0665

Table 5.10: Double-Q, Risk Neutral agent, test performance - Std

5.1.5.3 Learning risk aversion: High Risk Aversion

I will now start moving towards the other extreme, that is to increase κ in order to train the agent to first exhibit greater risk aversion than the utilities we have currently been using and then finally to train extreme risk aversion.

For a double-Q agent trained with a $\kappa = 0.013$, we see the test results in figure 5.14 and tables 5.11, 5.12. The trained agent now has a far lower final distribution of wealth than both the merton optimal agent and even a random agent. However the distribution of outcomes is far narrower. This can be visualised easily when one examines the respective sharpe ratios. The risk averse agent has a far higher sharpe ratio than even the merton optimal (recall that my adjusted sharpe does not subtract the risk-free rate so we can see these effects clearly).

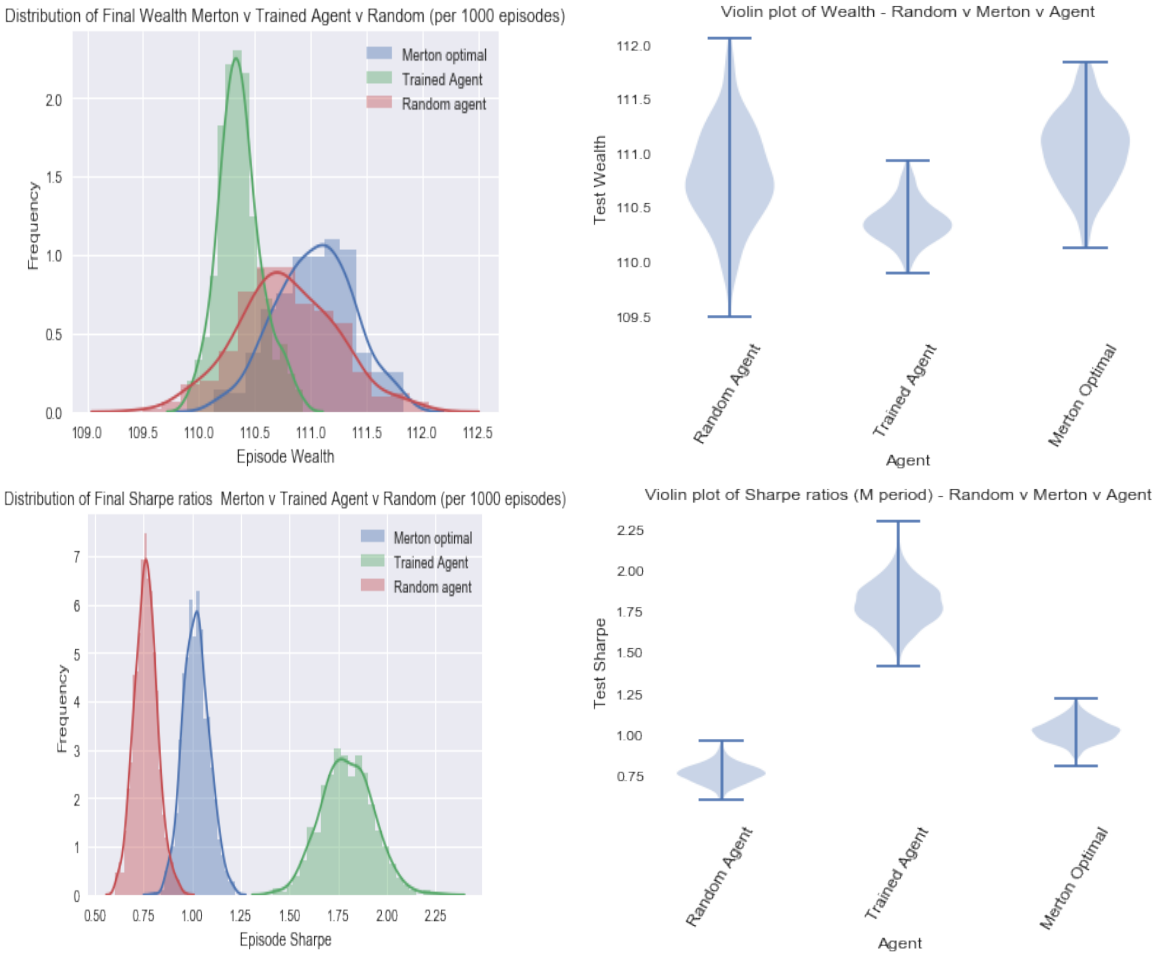


Figure 5.14: Double-Q learning: Wealth Distribution v Merton, Risk Aversion

	Random Agent	Double-Q	Merton Optimal
Wealth Distribution	110.78	110.36	111.03
Sharpe Ratio Distribution	0.7615	1.7959	1.0202

Table 5.11: Double-Q, Risk averse agent, test performance - Mean

	Random Agent	Double-Q	Merton Optimal
Wealth Distribution	0.4441	0.1923	0.3467
Sharpe Ratio Distribution	0.0586	0.1326	0.0665

Table 5.12: Double-Q, Risk averse agent, test performance - Std

5.1.5.4 Learning risk aversion: Extreme Risk Aversion

The story behind figure 5.15 hides a slightly painful anecdote. When I first implemented the code for step-wise mean-variance equivalence. I received the results in the right hand chart of figure 5.15. The flat red line showed that the double-Q agent had learned nothing. I interpreted this as a bug in my code and spent over a day testing and trying to find out where my error lay.

It was only when I simulated various constant investment strategies with $\kappa = 0.13$ and calculated the final utilities for each strategy (as shown in the left figure of 5.15) that I realised that at this level of κ the optimal utility would be gained by a 0% investment in the risky asset and 100% in the risk-free bond. A final check was to examine the greedy actions the reinforcement learning agent took (almost always action 4, which corresponded to investing 100% in the risk free bond).

The answer of course was that I had calibrated my κ inadvertently to reflect extreme risk aversion and there was simply no distribution of wealth or utility, because there was no uncertainty! Indeed I had intended $\kappa = 0.013$, and so the day attempting to locate the bug was because of a single 0! The scientific point is that the RL agent has successfully learned extreme risk aversion.

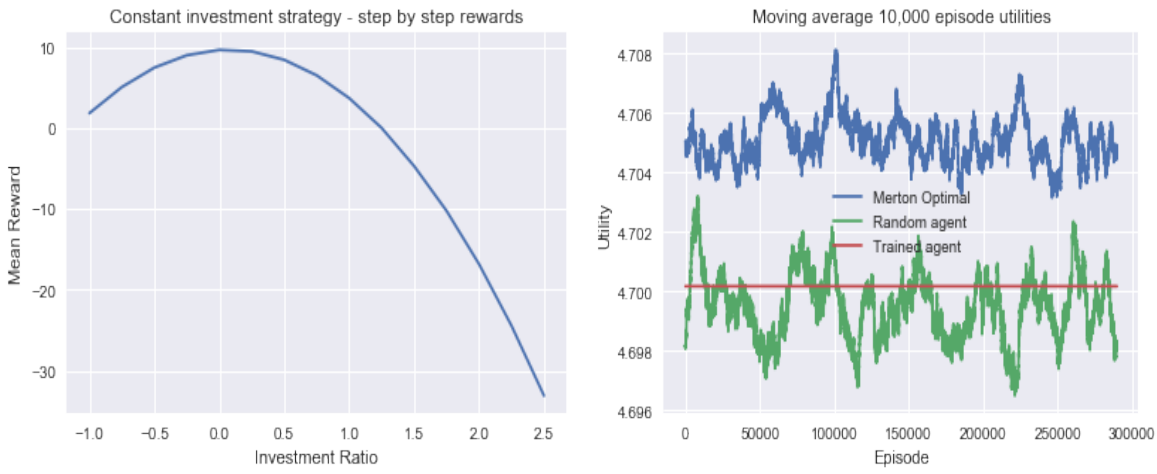


Figure 5.15: Double-Q learning: Utility curve for $\kappa = 0.13$, Extreme Risk Aversion

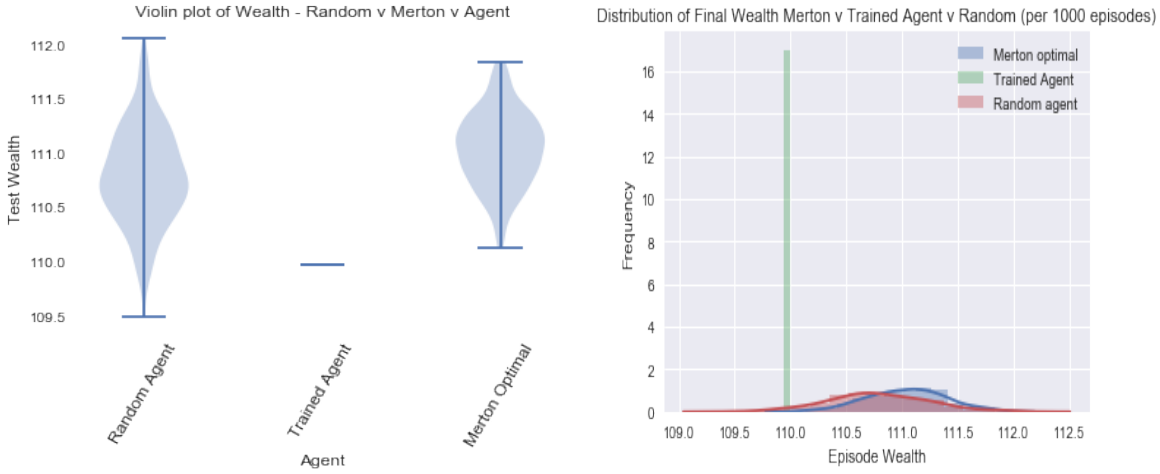


Figure 5.16: Double-Q learning: Wealth Distribution v Merton, Extreme Risk Aversion $\kappa = 0.13$

5.1.5.5 Learning risk aversion: Conclusion

The experiments thus far have tested a trained RL agent, first using an exact log-utility reward formulation, to a more general mean-variance approximation. The agent has successfully learned in a variety of volatility settings and we are able to tailor the risk aversion of the trained agent. This experiment shows the agent successfully learned a variety of behaviours, from extreme risk aversion to pure expected wealth maximisation.

From here I shall test the agent's behaviour compared to a known merton optimal solution on a power utility curve and then move towards greater realism including 'fat tails'.

5.2 Experiment group 2: Adding Realism

5.2.1 Realistic (S&P) volatility and risk free levels

In prior experiments we have shown that an RL agent (in this case a double-Q learner) can at least perform way better than a random agent and fairly close in performance to a merton optimal agent. This has so far been using geometric brownian motion and low to mid-level volatility levels. Clearly as volatility increases then the

reward signals for the agent can become more confusing. In this particular example I recallibrate the problem. The geometric brownian motion will have levels of risk approximating that of the S&P 500, and the risk free rate will be far lower (approximating currently low rates). Again the aim is to see if an RL can approximate the performance of merton optimal.

5.2.1.1 (S&P) volatility: Description

The stochastic process parameters are now changed to give a drift $\mu = 0.1$, and a volatility $\sigma = 0.2$, the risk free rate $r = 0.02$. These numbers correspond more closely to a stochastic process akin to the long run returns of the S&P 500, with far higher volatility than the prior examples. The risk-free rate has been reduced, reflecting current low rates. A lack of realism remains in that we are still using a geometric brownian motion, but this does at least enable a proper analytical comparison to the merton optimal solution once again.

Figure 5.17 illustrates some discretised sample paths in this new environment.

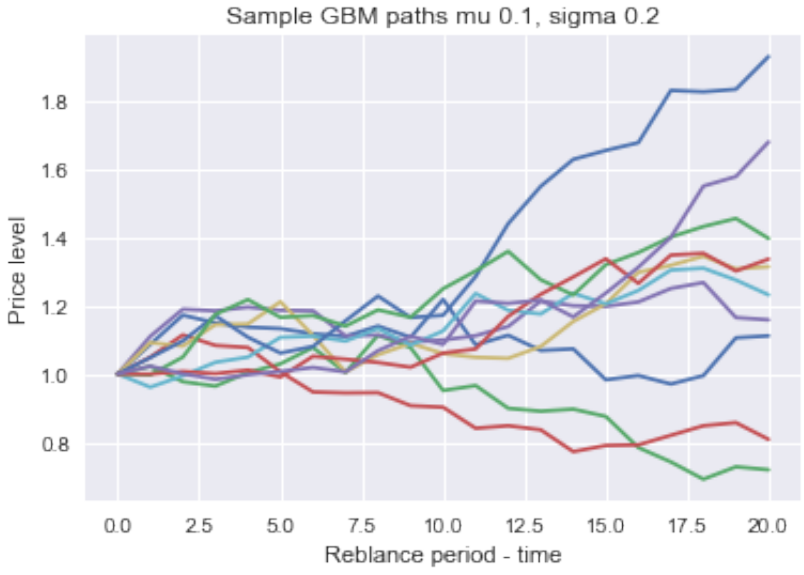


Figure 5.17: GBM sample paths, SP500 volatility levels

5.2.1.2 (S&P) volatility: Utilities and rewards

Figure 5.18 illustrates a variety of discretised constant investment simulations, where I have again shown the mean log-utility of episodes. In this case the merton optimal investment would be approximately 2x leveraged. In the right hand chart I have shown the same simulation utilising mean-variance approximated rewards. As a second order approximation there is no reason these curves should be identical, but κ should be such that the optimal solution for the step-wise rewards is similar to the end of episode log-utilities. I have used $\kappa = 0.006$.

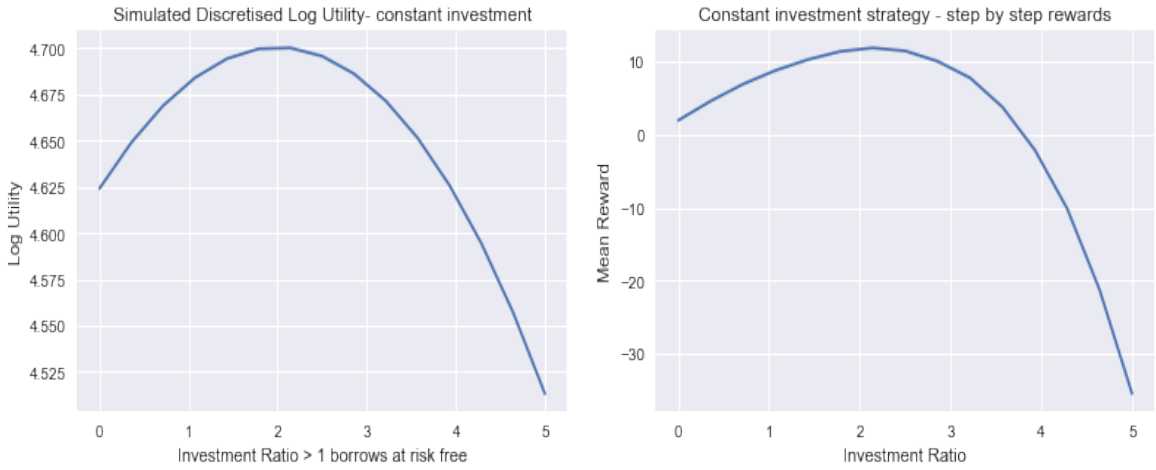


Figure 5.18: $\sigma = 0.2$, Log-utility, End episode utilities v Step-wise rewards $\kappa = 0.006$

Figure 5.19 shows the performance in this environment of the merton optimal agent compared to a random agent for both utilities and rewards. There remains an overlap in the two distributions due to the far higher level of volatilities however this is somewhat mitigated by the increased difference in returns between the risky asset and the risk free. Notice that the rewards distribution as an approximation is less clear than the final utilities, of when using mean-variance equivalence the agent will not have the benefit of seeing the utilities.

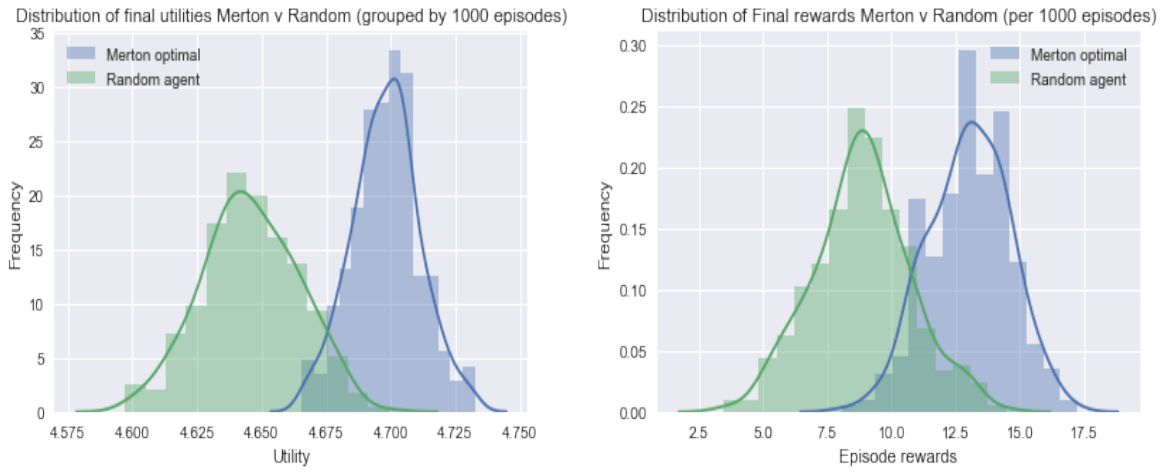


Figure 5.19: Batched Utilities and Reward distributions - Random v Merton Optimal

5.2.1.3 (S&P) volatility: Conclusion

I consider these results to be excellent. Recall that in this case the Q-learner knows nothing of the stochastic process or any parameters. However on the unseen test set the agent's distribution of utilities, rewards, wealth and sharpe ratios is similar in both size and variation to merton optimal. Merton optimal remains a stronger performer, but despite higher volatility the Q-learner was able to learn to trade over multiple periods at a comparable level.

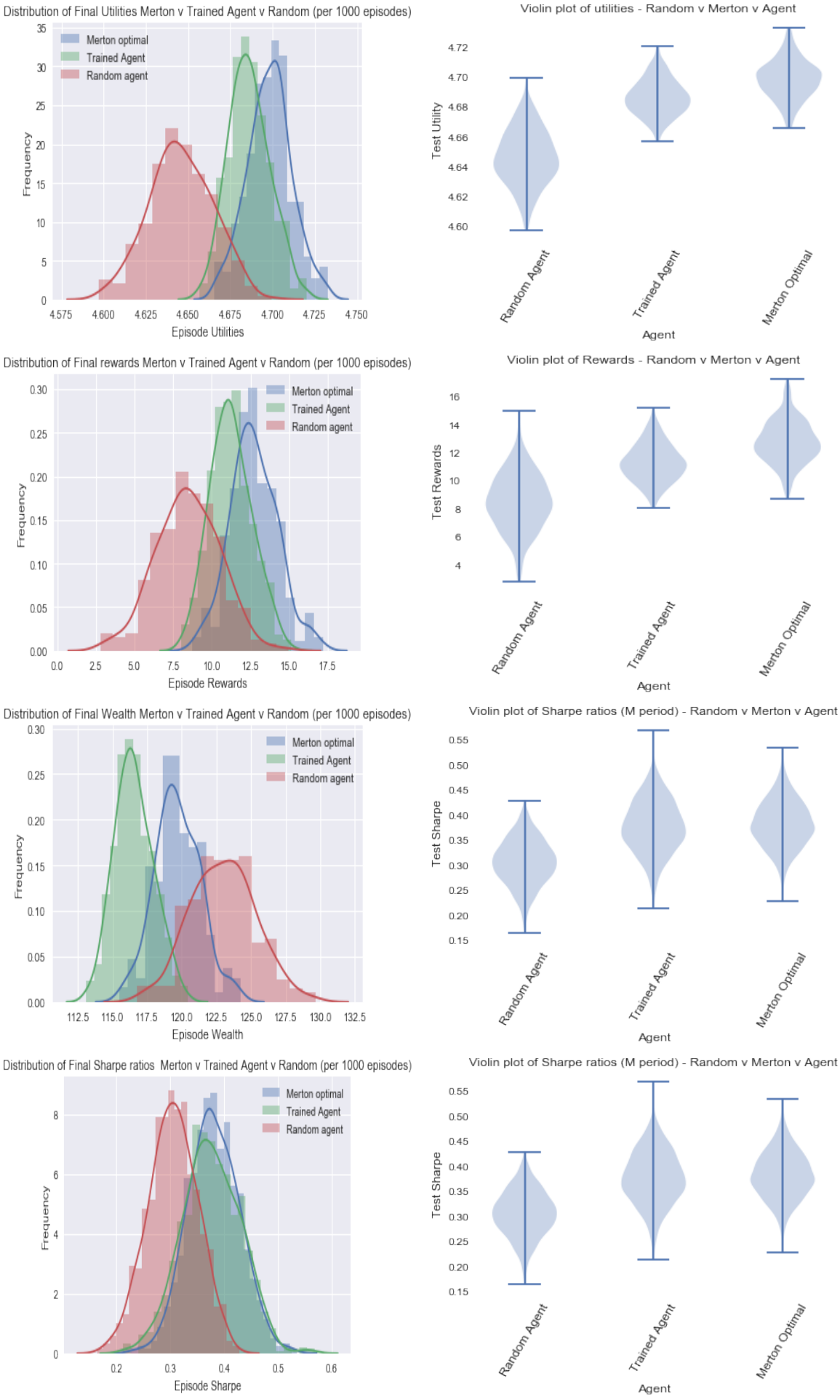


Figure 5.20: Q learning: Utilities verses Merton, $\sigma = 0.2$

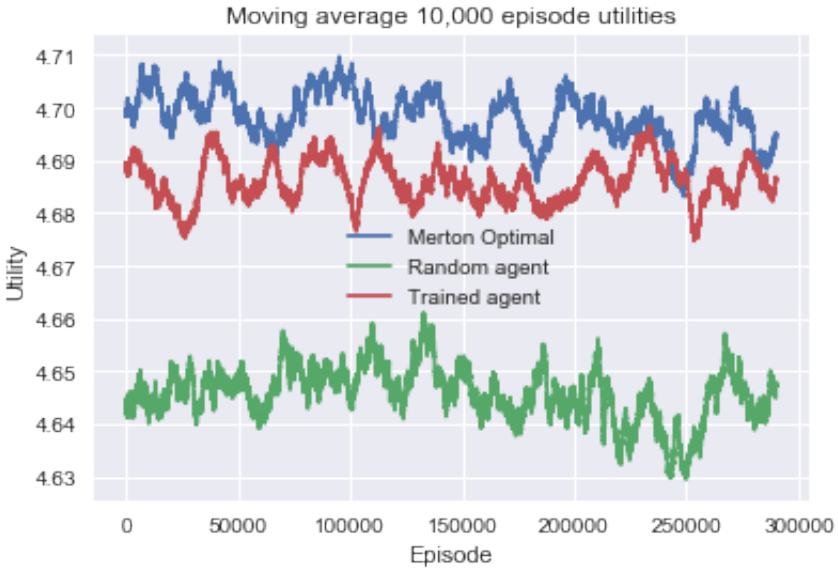


Figure 5.21: Moving average utilities Q v Merton, $\sigma = 0.2$

	Random Agent	Q	Merton Optimal
Utility Distribution	4.6457	4.6861	4.6981
Rewards Distribution	8.46	11.25	12.72
Wealth Distribution	122.97	116.54	119.68
Sharpe Ratio Distribution	0.30	0.37	0.38

Table 5.13: Q, test performance v Merton - Mean

	Random Agent	Q	Merton Optimal
Utility Distribution	0.0185	0.012	0.013
Reward Distribution	2.04	1.34	1.53
Wealth Distribution	2.34	1.39	1.64
Sharpe Ratio Distribution	0.04	0.05	0.04

Table 5.14: Q, test performance v Merton - Std

5.2.2 Fat tails

Real markets are not nearly as well behaved as geometric brownian motion. They have fat tails, jumps and are heteroscedastic. Thus far, the merton optimal agent has had a variety of advantages, the stochastic process is known and deliberately sourced to have an exact analytic solution, the parameters of this process also being known in advance. In real markets the underlying distribution is not known, and nor are the parameters stable. In these cases the merton agent becomes increasingly ill-specified, whereas a model free RL agent should be capable of learning from the data regardless of the underlying process.

5.2.2.1 Fat tails: Description

In experiment 7, I shift very mildly towards some real market effects. Instead of gaussian increments I add increments from a student-t distribution. Volatility $\sigma = 0.35$ is also very high. The effect of this is to give a less well behaved and slightly fat-tailed market environment. With 10 degrees of freedom for the student-t distribution the fat tails are by no means extreme, but the idea behind the experiment is to test if the RL agent will close the gap to the merton agent even with slightly fat tails. The underlying stochastic process is no longer a geometric brownian motion, but still not too extreme.

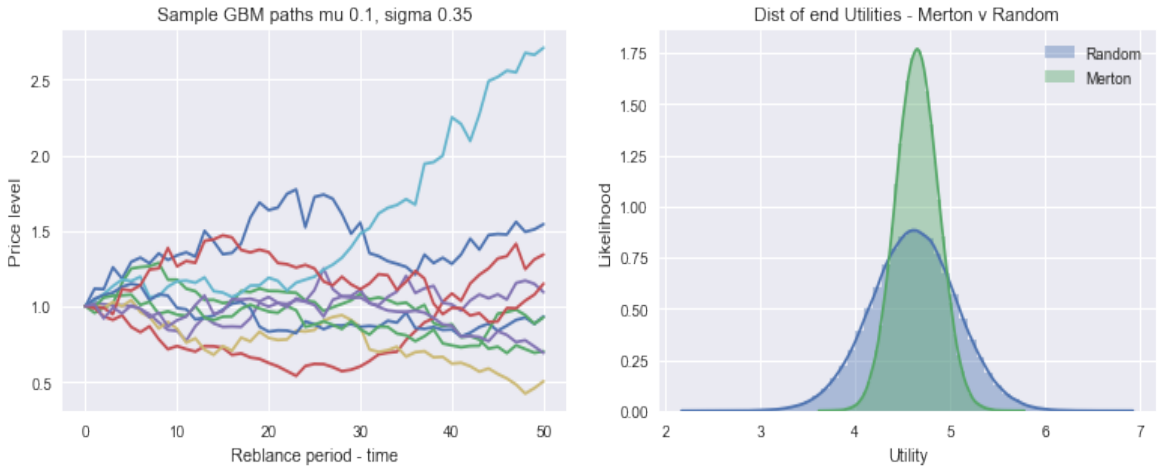


Figure 5.22: Sample fat tail paths/ Episodic distribution, Random agent v Merton

Figure 5.22 illustrates some sample realisations of the new process, notice the wider range of states the market can reach. In order to accommodate these I slightly increased the range of state spaces for the RL agent to 150, I also increased the number of possible actions to 25 different investment choices at each time-step. The right hand figure shows the distribution of end utilities between a merton agent and random. Clearly the random agent has a far wider variety of utility outcomes.

In figure 5.23 I show the difference in utility and reward distribution between a merton agent and a random agent.

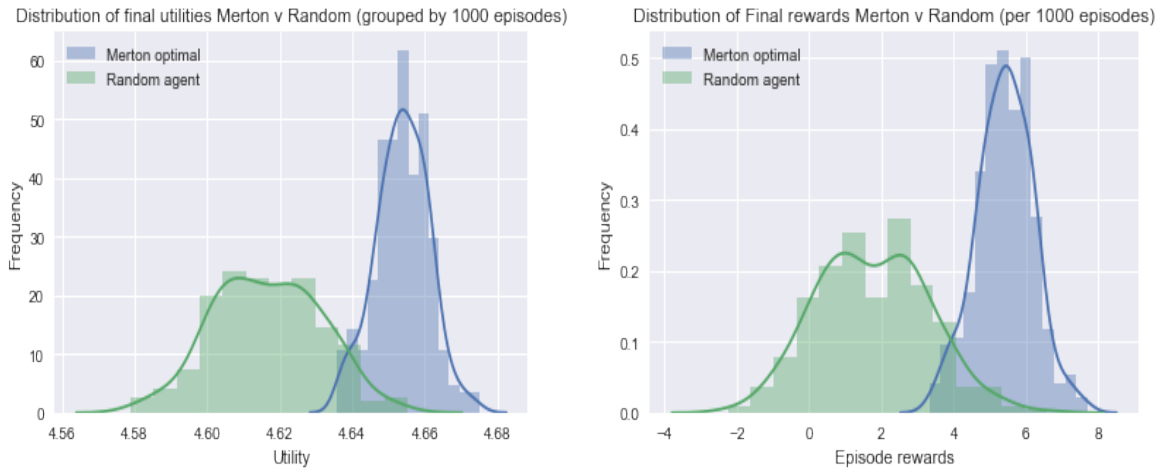


Figure 5.23: $\sigma = 0.35$ fat-tails, Log-utility, End episode utilities v Step-wise rewards $\kappa = 0.006$

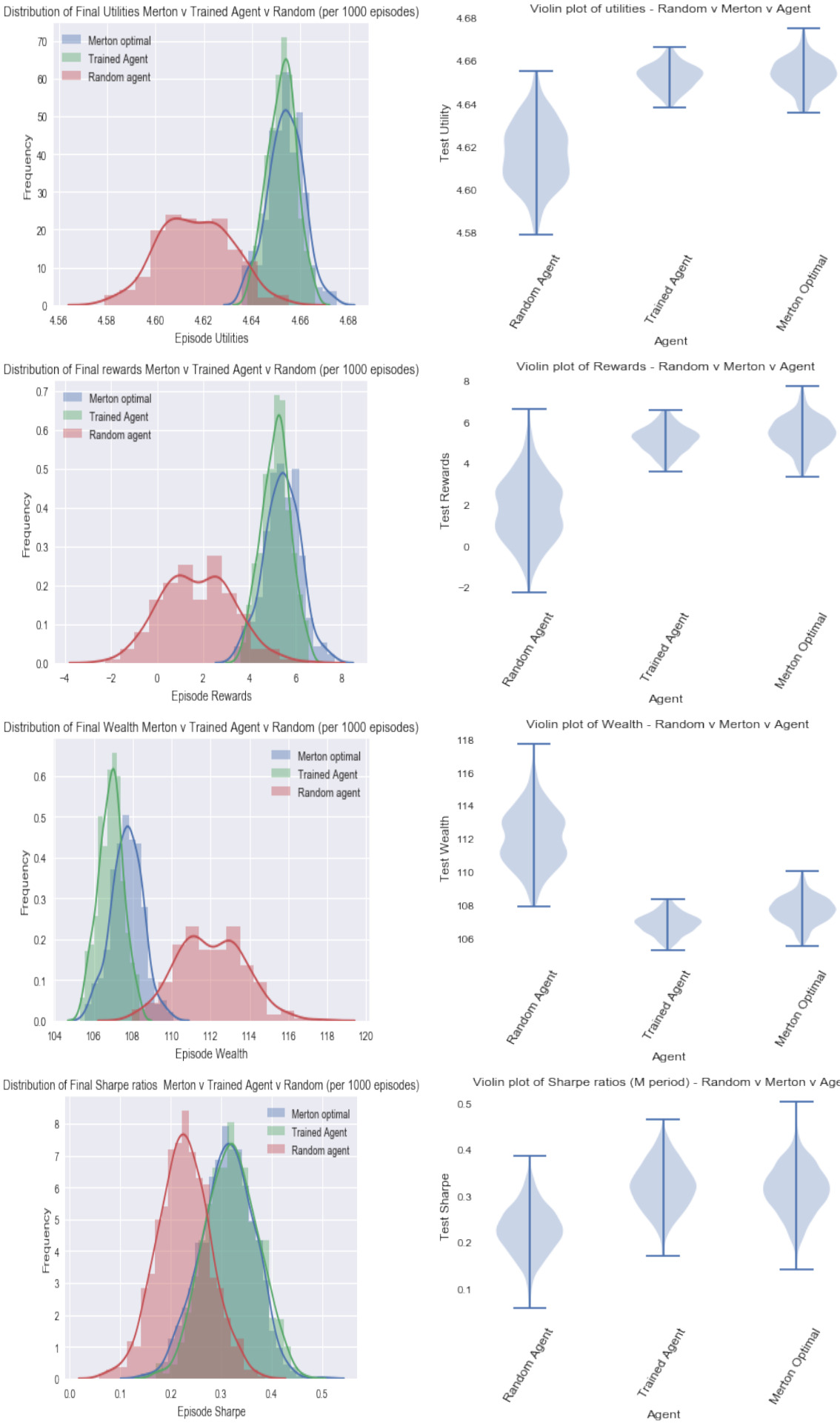


Figure 5.24: Q learning: Utilities verses Merton, $\sigma = 0.35$, fat tails

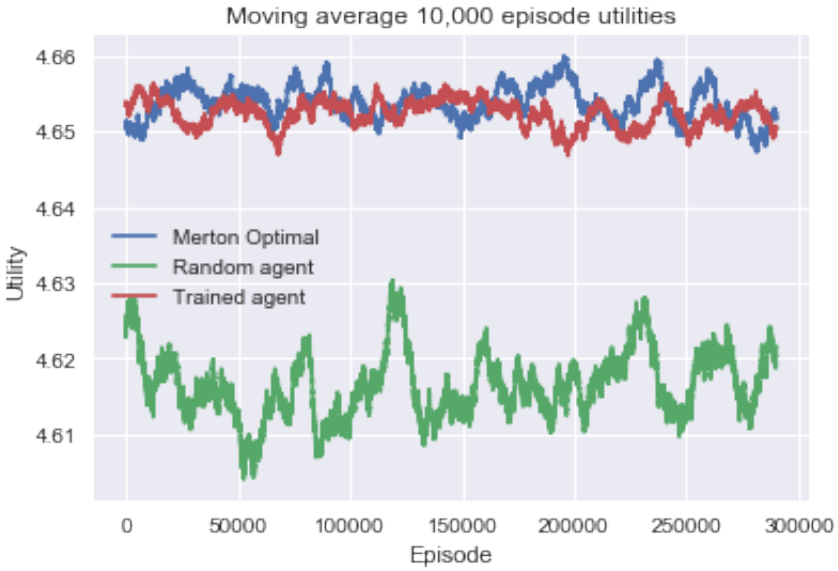


Figure 5.25: Moving average utilities Q v Merton, $\sigma = 0.35$, fat tails

	Random Agent	Q	Merton Optimal
Utility Distribution	4.6165	4.6523	4.6537
Rewards Distribution	1.7574	5.1524	5.4188
Wealth Distribution	112.02	106.87	107.69
Sharpe Ratio Distribution	0.2255	0.3201	0.3117

Table 5.15: Q, test performance v Merton - Mean, fat tails

	Random Agent	Q	Merton Optimal
Utility Distribution	0.01474	0.0058	0.0073
Reward Distribution	1.5255	0.6100	0.7847
Wealth Distribution	1.7025	0.6307	0.8041
Sharpe Ratio Distribution	0.0519	0.0515	0.0526

Table 5.16: Q, test performance v Merton - Std, fat tails

5.2.2.2 Fat tails: Conclusion

Figures 5.24, 5.24 and tables 5.15, 5.16 illustrate the test results. Results are excellent. The introduction of very mild fat tails has resulted in the double-Q learner matching the performance of the merton agent for utilities, resulted in a better performance as regards expected rewards and expected sharpe ratio and a favourable

distribution of results in that it has achieved them with less variability in terms of outcome. As an aside, do not be fooled by the strong performance as regards terminal wealth for the random agent, this is simply a function of the range of actions I made available and the relative drift between the risk-free bond and risky asset. Indeed the random agent has horrible performance as regards utilities, rewards, and sharpe ratios.

I believe this illustrates one of the potential advantages of a model free reinforcement learner...in the real world we do not know the exact underlying stochastic process, nor indeed its parameters, thus the merton model may be mis-specified. The RL will learn regardless. (Of course RL has other disadvantages such as needing a lot of data, however compromise semi-parametric approaches, where one narrows down the possible range of stochastic processes, learns parameters and generates data for the RL may be one practical approach here - for example using Gaussian or t-processes [24]).

5.3 Experiment group 3: Adding power, alternative RL agents

5.3.1 Tabular agents Double Sarsa and Dyna

Here the experimental setting is exactly the same as previously, that is a high volatility market environment with fat-tails, however this time I will examine the test performance of double-Sarsa and Dyna.

5.3.1.1 Double Sarsa

Figures 5.26 and 5.26 show the distributions of utilities, rewards and the moving average of utilities again for a random agent, compared to a merton optimal, with our RL agent being Double-Sarsa with an $\varepsilon - greedy$ exploration parameter set to 0.1. Like Q-learning, Sarsa can also exhibit bias, thus my use of a Double-Sarsa agent. Also recall that Sarsa is an on-policy learning algorithm, that is that the q-values reflect the expected reward from executing an ε-greedy policy, not a greedy one. Again for test purposes the agent acts greedily with respect to the q-values.

Although the Double-Sarsa agent also learned reasonably well and was far superior to a random agent, in general I found that the performance was weaker than the earlier double-Q agents.

A classic example where Sarsa performs better than Q-learning is the windy gridworld example given by Sutton and Barto [34]. The Q-learner chooses an optimal path close to the edge of the cliff, with no margin for error, but where the environment being windy causes losses to the agent's rewards. In contrast the Sarsa agent takes a wider path around the cliff edge (due to it learning from random exploration in training the dangers of being next to the edge of the cliff). However, in this environment Sarsa tended to be slightly weaker.

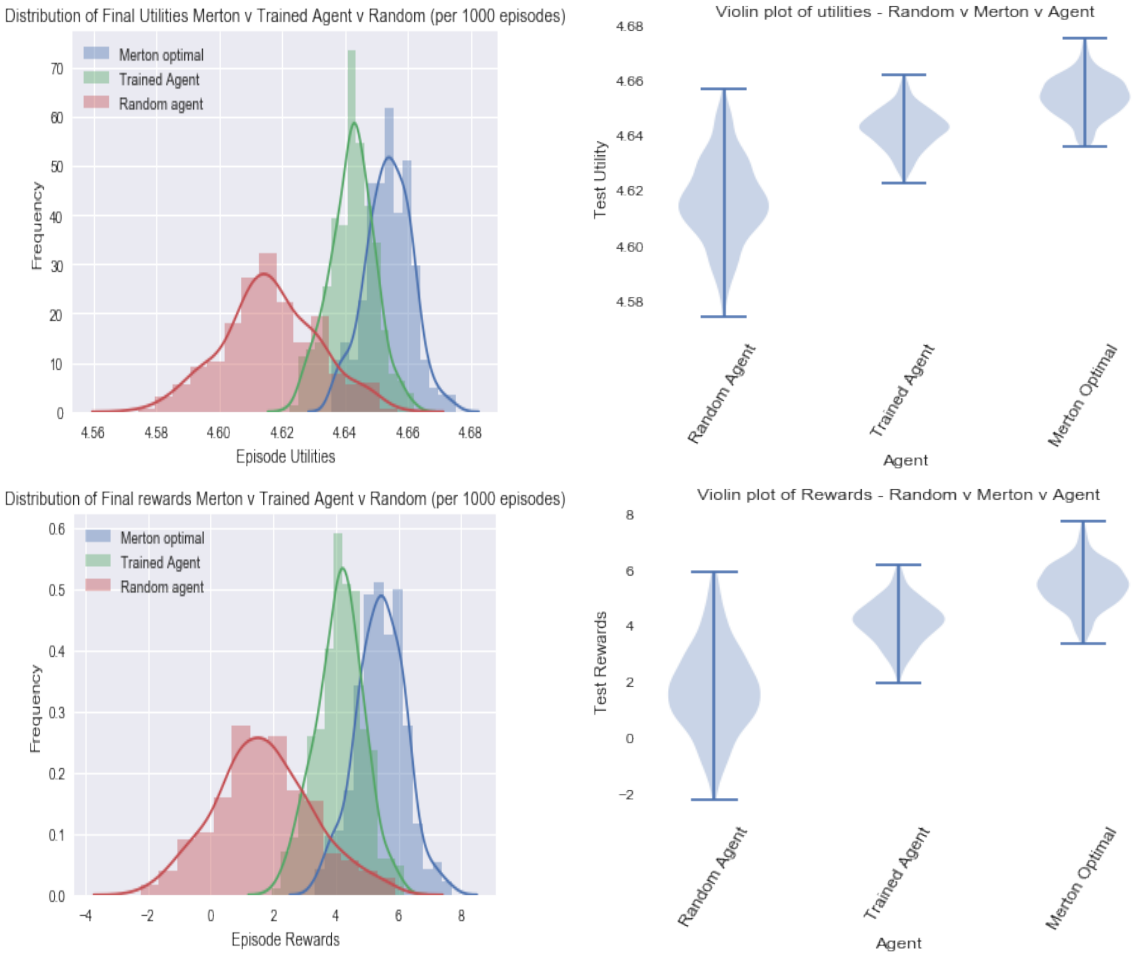


Figure 5.26: Double Sarsa learning: Utilities verses Merton, $\sigma = 0.35$, fat tails

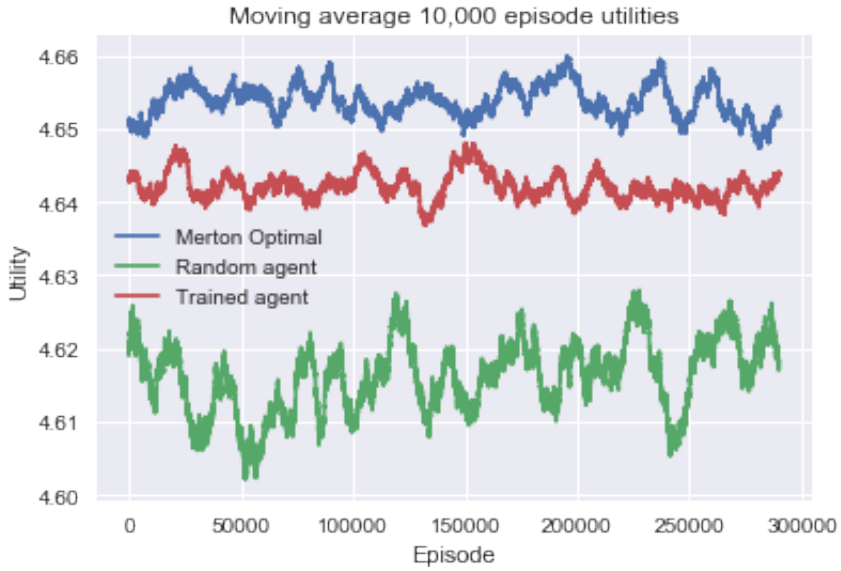


Figure 5.27: Moving average utilities Double Sarsa v Merton, $\sigma = 0.35$, fat tails

5.3.1.2 Dyna

My Dyna implementation used a very small amount of experience replay (I used replays between 1 and 5, due to already executing 3 million episodes in training), as well as a linear model and tabular implementation. A tabular setting is not entirely suitable for this environment due to the stochasticity, thus the results in figures 5.28, 5.29 and tables 5.17, 5.18 are really to demonstrate that Dyna was capable of learning within this environment again at close to optimal levels. I believe improvements can be gained by using a more sophisticated model within Dyna. However results are still reasonable and my goal was implementation rather than optimisation here.

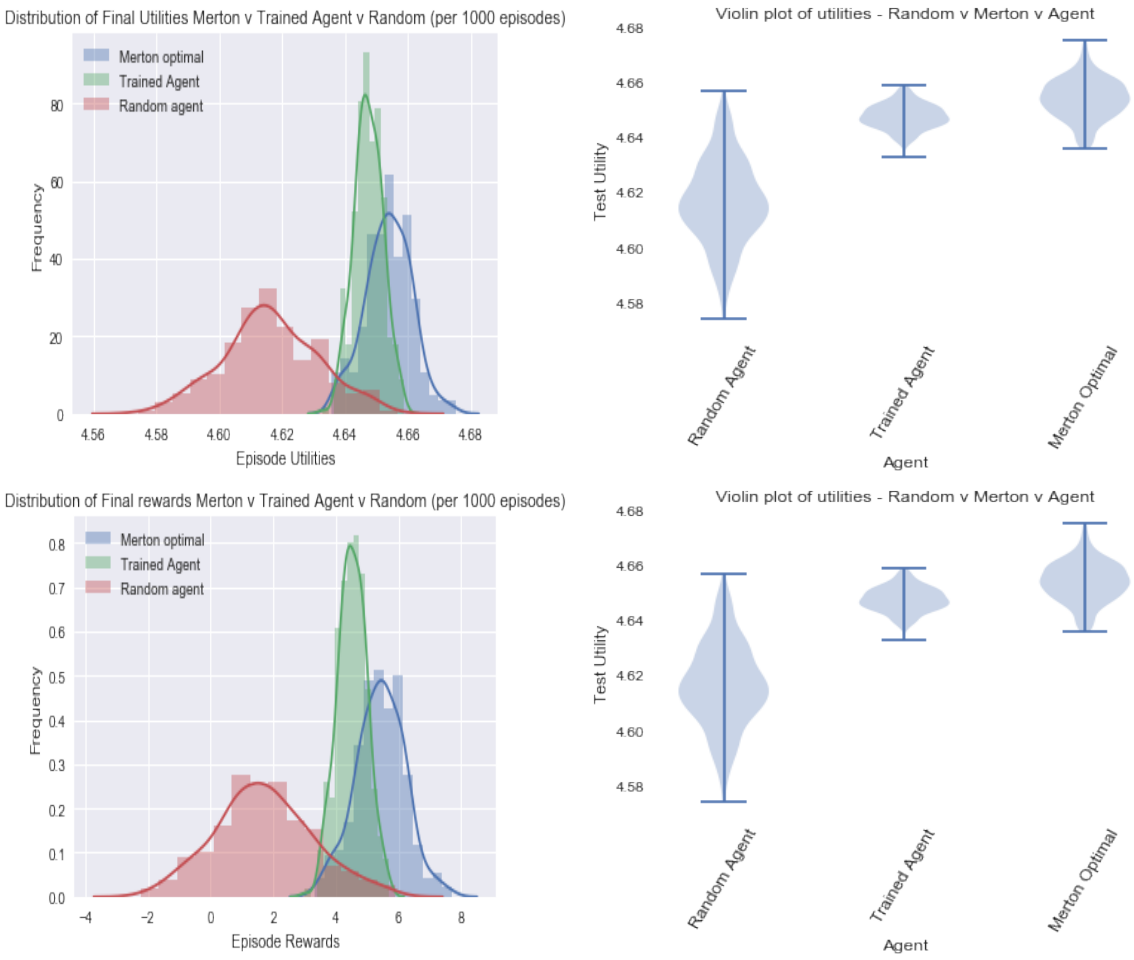


Figure 5.28: Dyna with experience replay: Utilities verses Merton, $\sigma = 0.35$, fat tails

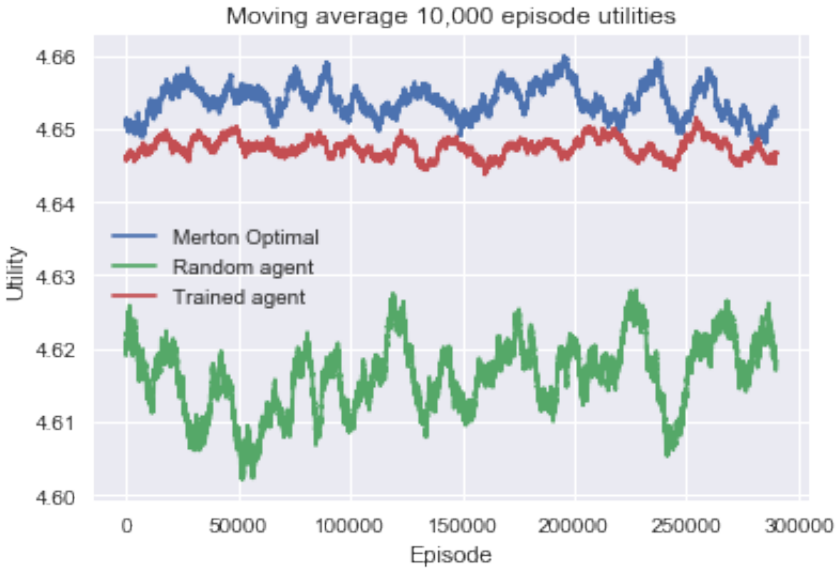


Figure 5.29: Moving average utilities Dyna with experience replay v Merton, $\sigma = 0.35$, fat tails

	Random Agent	Dyna	Merton Optimal
Utility Distribution	4.6162	4.6363	4.6537
Rewards Distribution	1.7101	3.9179	5.4188
Wealth Distribution	111.96	106.51	107.69
Sharpe Ratio Distribution	0.2246	0.2466	0.3117

Table 5.17: Dyna-experience replay, test performance v Merton - Mean, fat tails

	Random Agent	Dyna	Merton Optimal
Utility Distribution	0.0150	0.0083	0.0073
Reward Distribution	1.5482	0.8581	0.7847
Wealth Distribution	1.7195	0.8663	0.8041
Sharpe Ratio Distribution	0.0522	0.0601	0.0526

Table 5.18: Dyna-experience replay, test performance v Merton - Std, fat tails

5.3.1.3 Tabular agents: Conclusion

In experiment 8, I tested a couple of alternative RL agents, Double Sarsa and Dyna. Both successfully learned in the noisy fat tailed environment. Sarsa in general had slightly weaker performance than the earlier Double-Q learners but still performed way better than a random agent. Dyna is somewhat different as we have now added

an internal model, which can range from a basic tabular model unsuited to a stochastic environment to any sort of machine learning model designed to predict the response from the environment of our actions. Dyna performed well, but I believe can be improved given time and effort improving the model and further tweaking parameters such as the number of experience replays performed.

5.3.2 Deep-Q learning

Figure 5.30: Sample paths and log-utilities - constant investment, $\sigma = 0.20$

Layer(type)	Output shape	Parameters
Linear-1	[-1, 128]	384
ReLU-2	[-1, 128]	0
Linear-3	[-1, 128]	16,512
ReLU-4	[-1, 128]	0
Linear-5	[-1, 15]	1,935
Input dimension:2 (risky asset price and wealth)		Total params: 18,831 Trainable params: 18,831 Non-trainable params: 0

Table 5.19: Deep Q model specification

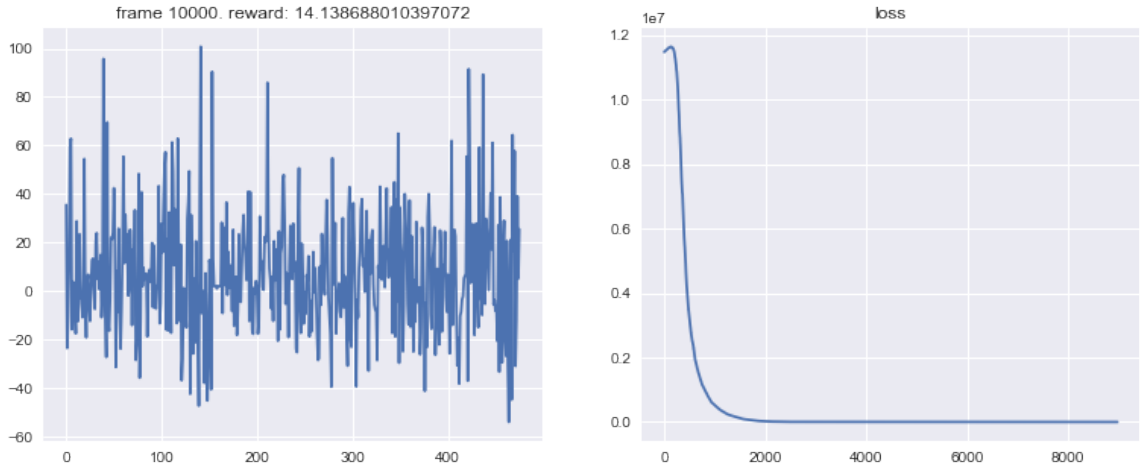
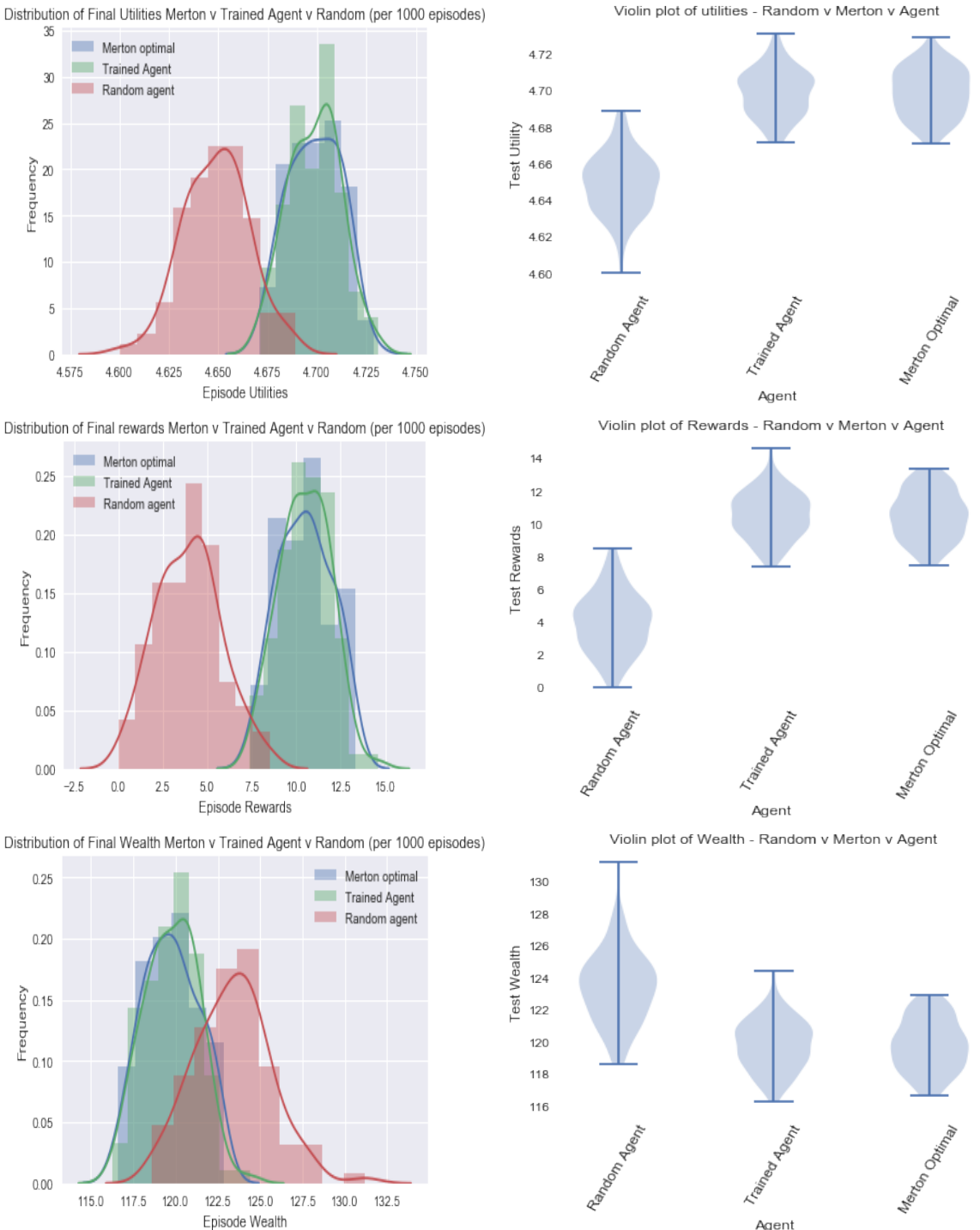


Figure 5.31: Final training rewards and loss function training DQN



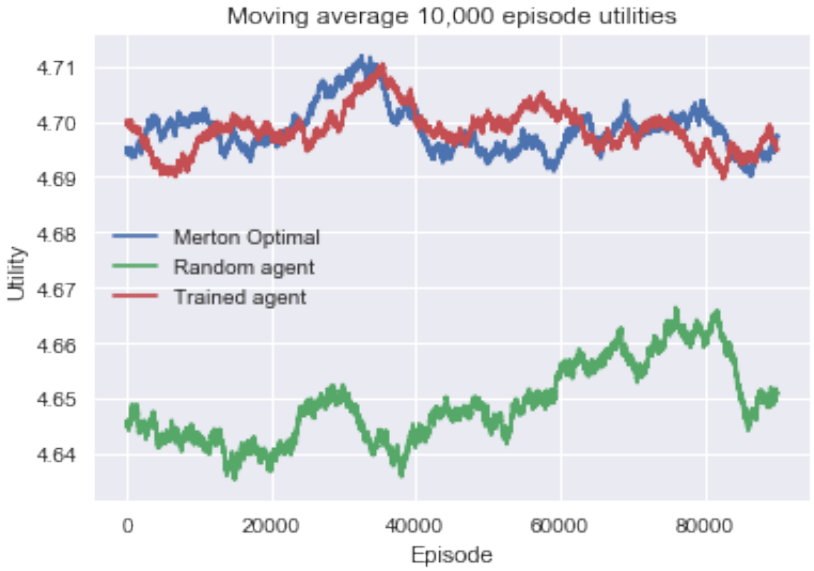


Figure 5.33: Moving average utilities Deep Q v Merton, $\sigma = 0.20$

	Random Agent	Deep Q	Merton Optimal
Utility Distribution	4.6489	4.6987	4.6987
Rewards Distribution	3.9576	10.4992	10.4936
Wealth Distribution	123.36	119.77	119.74
Sharpe Ratio Distribution	0.3098	0.3831	0.3837

Table 5.20: Deep Q, test performance v Merton - Mean

	Random Agent	DeepQ	Merton Optimal
Utility Distribution	0.0163	0.0130	0.0132
Reward Distribution	1.8240	1.4019	1.4837
Wealth Distribution	2.2410	1.5692	1.6197
Sharpe Ratio Distribution	0.0781	0.0833	0.0838

Table 5.21: DeepQ, test performance v Merton

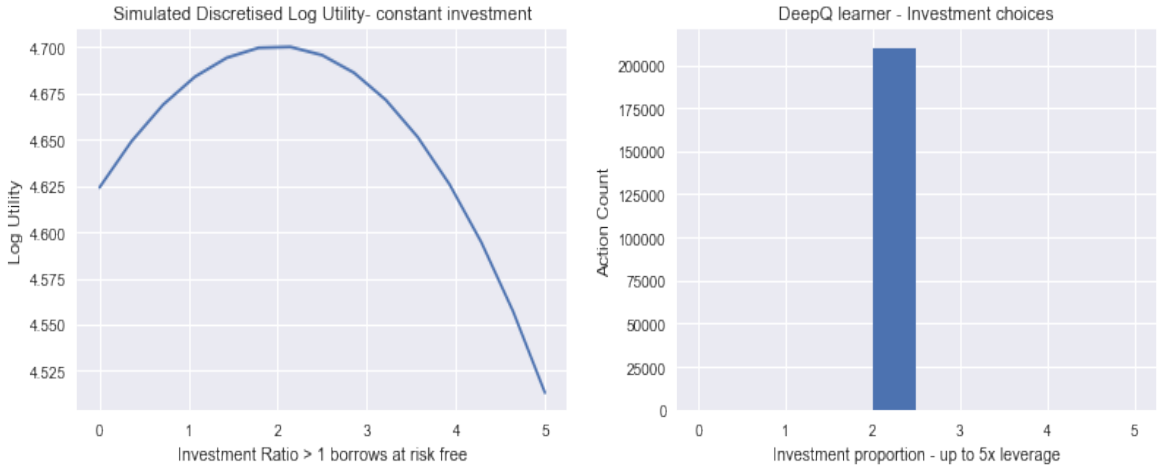
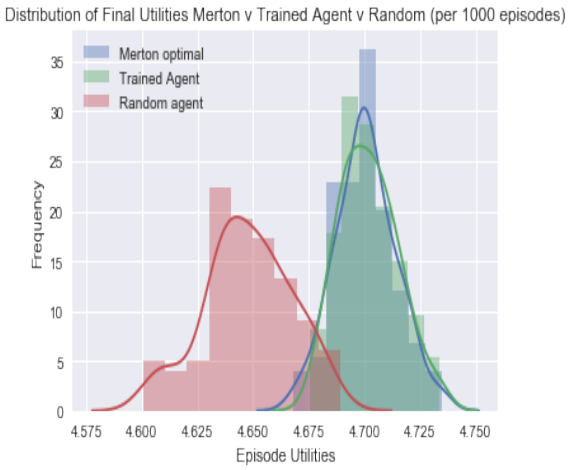
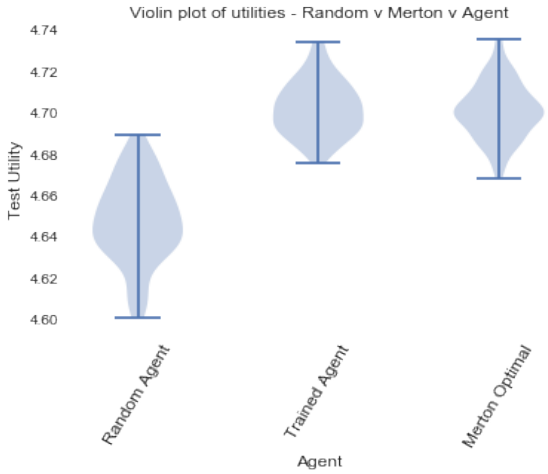
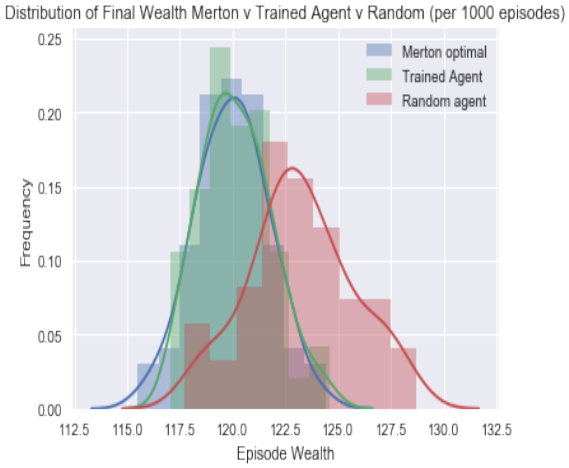


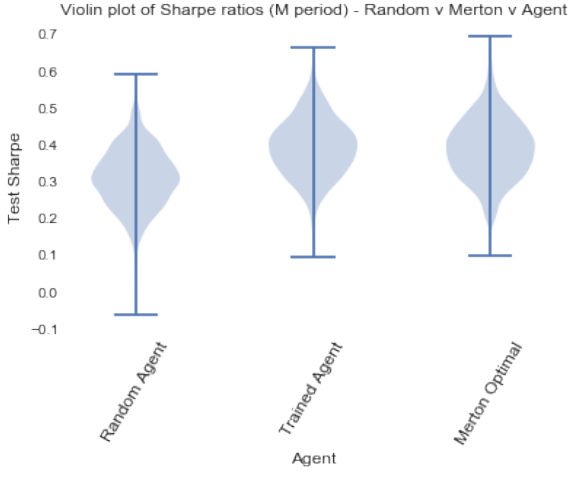
Figure 5.34: DQN investment choices for a variety of states

5.3.3 Double Deep-Q learning









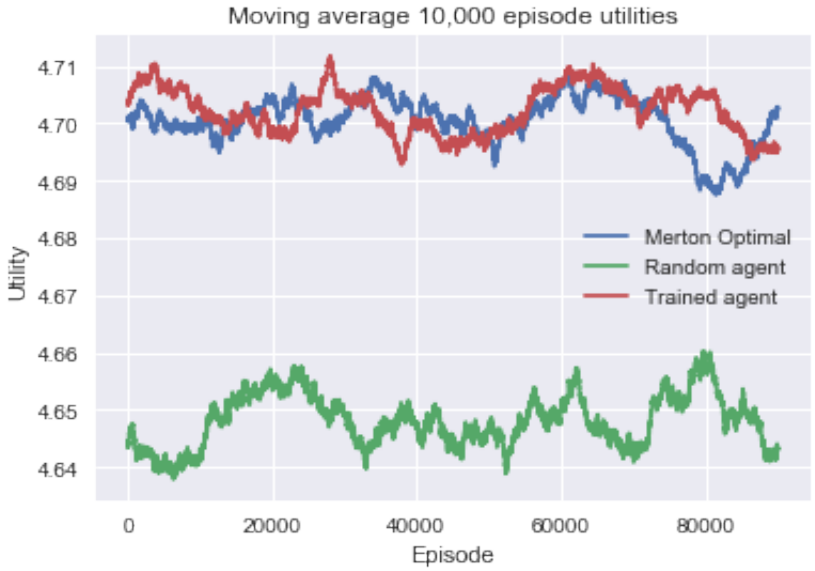


Figure 5.36: Moving average utilities Deep Q v Merton, $\sigma = 0.20$

	Random Agent	Double Deep Q	Merton Optimal
Utility Distribution	4.6489	4.6987	4.6987
Rewards Distribution	3.9576	10.4992	10.4936
Wealth Distribution	123.36	119.77	119.74
Sharpe Ratio Distribution	0.3098	0.3831	0.3837

Table 5.22: Double Deep Q, test performance v Merton - Mean

	Random Agent	Double DeepQ	Merton Optimal
Utility Distribution	0.0163	0.0130	0.0132
Reward Distribution	1.8240	1.4019	1.4837
Wealth Distribution	2.2410	1.5692	1.6197
Sharpe Ratio Distribution	0.0781	0.0833	0.0838

Table 5.23: DeepQ, test performance v Merton

5.3.4 Noisy Net Deep-Q Networks

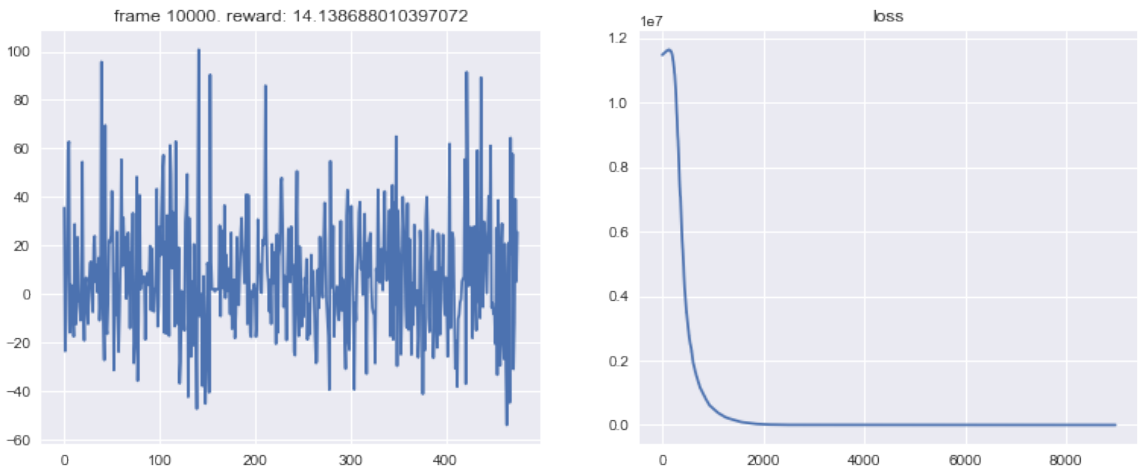


Figure 5.37: Final training rewards and loss function training DQN

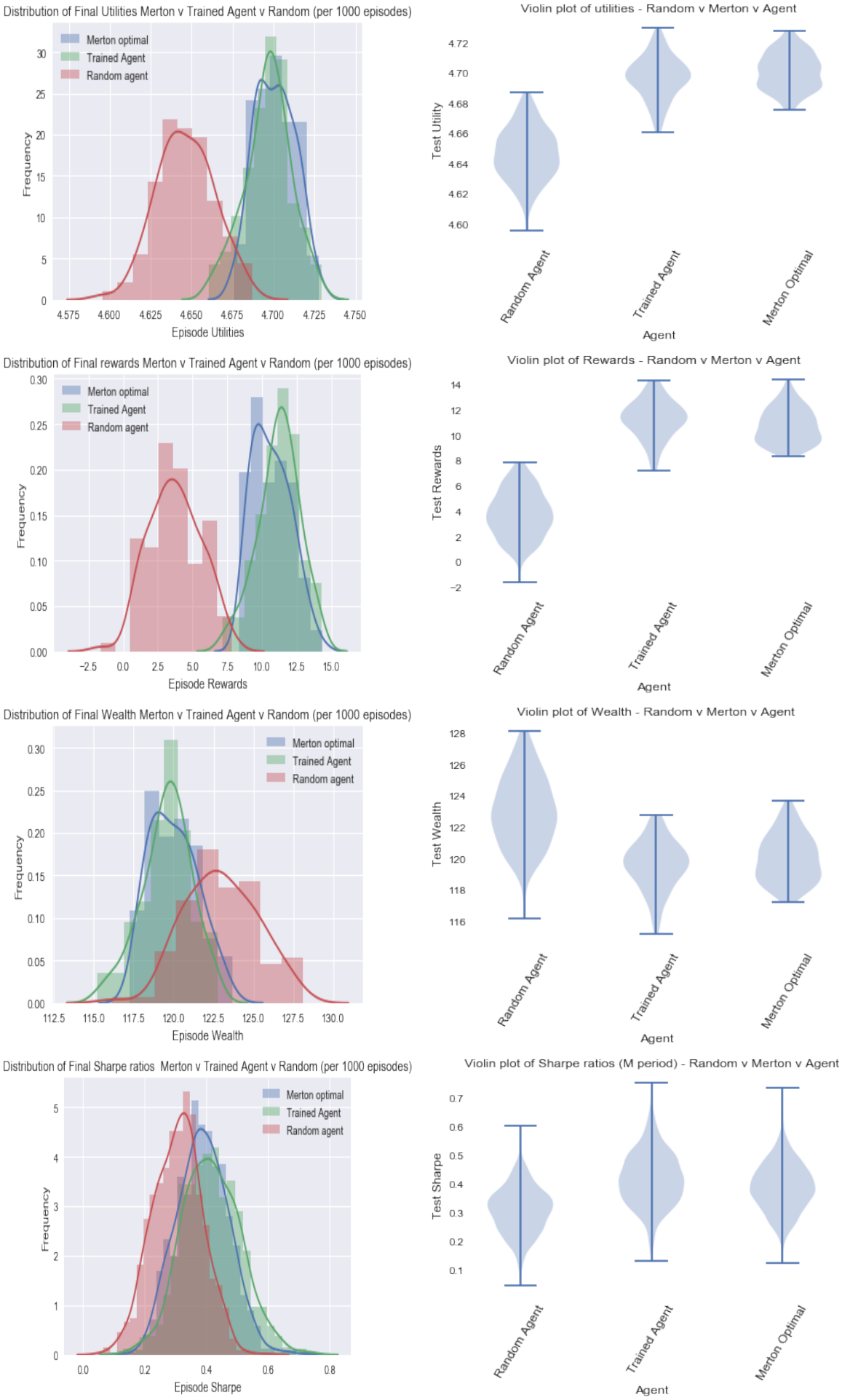


Figure 5.38: Noisy Deep Q learning: Utilities verses Merton, $\sigma = 0.20$

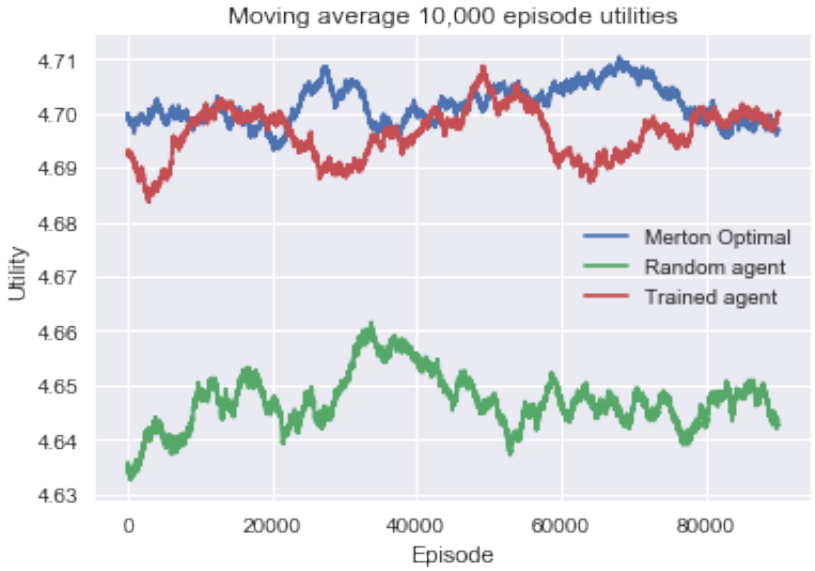


Figure 5.39: Moving average utilities Noisy Deep Q v Merton, $\sigma = 0.20$

	Random Agent	Noisy Deep Q	Merton Optimal
Utility Distribution	4.6462	4.6965	4.7008
Rewards Distribution	3.7337	11.2096	10.6893
Wealth Distribution	122.98	119.55	120.00
Sharpe Ratio Distribution	0.3064	0.4162	0.3881

Table 5.24: Noisy Deep Q, test performance v Merton - Mean

	Random Agent	Noisy Deep Q	Merton Optimal
Utility Distribution	0.0172	0.0140	0.0120
Reward Distribution	1.8445	1.4985	1.3590
Wealth Distribution	2.2585	1.5621	1.4887
Sharpe Ratio Distribution	0.0802	0.0952	0.0845

Table 5.25: Noisy DeepQ, test performance v Merton

Chapter 6

General Conclusions

6.1 Main Conclusions

In this thesis I used model free reinforcement learning techniques and applied them to the Merton problem. An analytical solution of Merton's problem is only possible for specific known stochastic processes, with known parameters and then derived for a specific utility curve (in chapter 2 I show one of the derivations which involves techniques from stochastic optimal control). In contrast an RL agent was able to learn to trade purely from the data over multiple time periods often closely matching the test performance of a Merton optimal agent.

A major part of solving an RL problem lies in creating the environment and agent. In this particular problem I had to reshape an end of period expected utility of wealth $\mathbf{E}[U(w_T)]$ into a reward suitable for an RL agent to learn from on a step-wise basis. I approached this first by choosing a particularly amenable utility curve such as log-utility and a 'friendly' stochastic process such as geometric brownian motion. Here some very basic mathematics which I show in chapter 4 enables an exact step-wise reward matching the required end of period expected utility.

From there I wished to generalise the RL to any utility curve (and indeed a wider range of stochastic processes), here I used work by Ritter, 2017 [25] where he used a mean-variance approximation to formulate rewards in order to learn to

trade an Ornstein-Uhlenbeck process with transaction costs. By applying this to the Merton problem I was able to move beyond log-utility and could also tune the RL agent's risk aversion.

A second issue which may perhaps be the key issue in financial markets is noise. Many recent RL successes have come from high dimensional environments (such as Atari), where the environment is relatively stable and rewards although perhaps delayed are not necessarily very noisy. Indeed the agent often also has considerable control not only of his future rewards, but indeed the next state he is likely to find himself within. These problems are often solved by reducing the dimensionality using a function approximator such as a neural network.

In contrast the problem setting here was relatively low dimensional, for the most part I had 100-150 buckets of wealth, 20-50 time periods to consider and 15-25 investment actions that the agent could choose at each time period. This is still non-trivial, after all for the later experiments this resulted in 25^{50} possible actions for each episode and over 3,000 state action q-values to consider. But this is also not intractable in the sense of needing a neural network, indeed I use straight forward tabular methods (for the most part Double-Q learning). The issue though is that rewards are very noisy and the difference in the distribution of rewards and utilities for even a perfect Merton optimal agent and a random agent can quite difficult to ascertain unless one replays millions of episodes, and more difficult still as noise levels increase as the distributions are heavily over-lapping.

This problem resulted in my being very keen to understand the actual test set distributions of rewards, wealth and sharpe ratios of my trained RL agent, compared to Merton and Random. The trained RL often managed to achieve a far superior distribution of outcomes compared to the random agent, and was frequently very close in distribution to the Merton agent, however it did also require a lot of episodes to do so (for the most part I used 3,000,000 episodes), given the noisy rewards.

The key conclusion is of course that for the most part the RL agent learned to trade at levels close to Merton optimal, but in a completely model free way.

6.2 Further Directions

I feel this thesis gives a proof of concept, but that there are many somewhat obvious extensions possible both on the machine learning side and the finance side, a thesis is time limited unfortunately.

On the finance side the problem setting really needs to be far more practical. Clearly we could increase the number of inputs into the state space by giving more features and information that might could be relevant for future prices. Real markets are not exact stochastic processes and the agent may learn to exploit this. This would of course increase the dimensionality of the problem, but here we have a well trodden path and policy gradients methods or deepQ-learning seem obvious choices to accommodate this. Real markets also have costs, transaction costs such as commissions, spread and slippage, however this can be accommodated within the reward shaping for the RL. Another extension is of course not just using multiple features, but multiple assets, here again dimensionality increases rapidly.

Remaining on the finance side, I believe RL can be applied to a variety of problems and indeed the Merton problem may not actually be the most suitable one (despite a relatively small success in this thesis). Problems such as market making, optimal execution, liquidation and some areas of statistical arbitrage may be highly suitable. In particular I would look for problems where the agent not only learns from rewards, but where he is capable of having greater influence over the next state he finds himself in. In the Merton problem he is a 'price taker', he can learn actions affecting the future probability distribution of his wealth, but much of the state of this environment comprises the price process of the market and over this the agent has no control. In contrast if we consider an RL agent trained to optimise the

utility of wealth through market making on the full limit order book, then the agent can directly affect that order book through his own trades and trade cancellations. In this game speed is of the essence and I believe current algorithms are relatively quick, but I don't believe RL algorithms are yet the norm in this area and think they could be well suited to the problem.

The RL agent is also impractical because it needs millions of episodes in order to learn. There are clearly more sophisticated model based reinforcement learning methods than I have used (where the agent learns his own model of the environment) which should be more sample efficient in this respect. But from a finance perspective the more immediate and practical option could be to attempt a semi-parametric approach. I have previously had some success using Gaussian processes in finance, which given a range a prior kernels were highly efficient at learning the underlying process as a posterior with relatively small amounts of data. The benefit here is that we are using a bayesian approach and can train a gaussian process (GP) on a small amount of real life data, but then use the trained GP to generate as much data as we need, which can in turn train the RL agent. Of course GP's being gaussian are not necessarily the best fit for real life markets, but there are extensions such as t-processes as shown by Shah and Gharamani at Cambridge University [28].

My final suggested directions come back to the heart of the problem in this thesis. If we wish to maximise only expected wealth then the reward formulation for an RL is immediate, if however we wish to maximise the expected future utility of terminal wealth it is certainly not. This requires some effort to shape rewards, but what we are actually doing in reality is risk-aware reinforcement learning with an end of period utility (which has risk aversion embedded into it) that we wish to maximise. This is an active research area which again may be applicable to these types of finance problems.

Finally, I find the RL setting where we learn to maximise expected rewards

(albeit where I have shaped those rewards to account for risk) perhaps limiting as it may not use all the information available. For this noisy problem setting I was actually more interested in the overall distribution of rewards, utilities and wealth - not their expected values. The RL agents I used q-values which approximate an expected reward. However Bellemere et al. 2017, derive algorithms for performing reinforcement learning from a distributional viewpoint, i.e. instead of learning state values, they learn the value distributions and indeed derive some state of the art results. To my mind this is appealing as I would immediately be able to examine not just q-values but the whole q-distributions and examine not only if the RL agent is learning, but also its certainty, in finance where there is a great deal of uncertainty this could prove very useful to know.

Chapter 7

Colophon

(Gaussian Process packages) For coding I used Python 3.6, together with the Anaconda environment as well as the PyTorch libraries.

Bibliography

[1] Ang, A. (2014). *Asset management: A systematic approach to factor investing*. Oxford University Press.

[2] Bellemare, M. G., Dabney, W., and Munos, R. (2017). A distributional perspective on reinforcement learning. *arXiv preprint arXiv:1707.06887*.

[3] Bishop, C. M. (2006). *Pattern recognition and machine learning*. springer.

[4] Björk, T. (2009). *Arbitrage theory in continuous time*. Oxford university press.

[5] Borch, K. (1969). A note on uncertainty and indifference curves. *The Review of Economic Studies*, 36(1):1–4.

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

[7] Chan, E. (2009). *Quantitative trading: how to build your own algorithmic trading business*, volume 430. John Wiley & Sons.

[8] Chan, E. (2013). *Algorithmic trading: winning strategies and their rationale*. John Wiley & Sons.

[9] Chapados, N. (2011). *Portfolio choice problems: An introductory survey of single and multiperiod models*. Springer Science & Business Media.

[10] Chapados, N. and Bengio, Y. (2007). Forecasting and trading commodity contract spreads with gaussian processes. In *13th International Conference on Computing in Economics and Finance*.

[11] Dickey, D. A. and Fuller, W. A. (1979). Distribution of the estimators for autoregressive time series with a unit root. *Journal of the American statistical association*, 74(366a):427–431.

[12] Farrell, M. T. and Correa, A. (2007). Gaussian process regression models for predicting stock trends. *Relation*, 10:3414.

[13] Gillespie, D. T. (1996). Exact numerical simulation of the ornstein-uhlenbeck process and its integral. *Physical review E*, 54(2):2084.

[14] Grinold, R. C. and Kahn, R. N. (2000). Active portfolio management.

[15] Halperin, I. (2017). Qlbs: Q-learner in the black-scholes (-merton) worlds. *arXiv preprint arXiv:1712.04609*.

[16] Hamilton, J. D. (1994). *Time series analysis*, volume 2. Princeton university press Princeton.

[17] Hastie, T., Tibshirani, R., and Friedman, J. (2002). The elements of statistical learning: Data mining, inference, and prediction. *Biometrics*.

[18] Levy, H. and Markowitz, H. M. (1979). Approximating expected utility by a function of mean and variance. *The American Economic Review*, 69(3):308–317.

[19] Markowitz, H. (1952). Portfolio selection. *The journal of finance*, 7(1):77–91.

[20] Merton, R. C. (1975). Optimum consumption and portfolio rules in a continuous-time model. In *Stochastic Optimization Models in Finance*, pages 621–661. Elsevier.

[21] Moody, J. and Saffell, M. (2001). Learning to trade via direct reinforcement. *IEEE transactions on neural Networks*, 12(4):875–889.

[22] Murphy, K. P. (2012). *Machine learning: a probabilistic perspective*. MIT press.

[23] Neuneier, R. (1996). Optimal asset allocation using adaptive dynamic programming. In *Advances in Neural Information Processing Systems*, pages 952–958.

[24] Rasmussen, C. E. and Williams, C. K. (2006). *Gaussian processes for machine learning*, volume 1. MIT press Cambridge.

[25] Ritter, G. (2017). Machine learning for trading.

[26] Roncalli, T. (2013). *Introduction to risk parity and budgeting*. CRC Press.

[27] Scholkopf, B. and Smola, A. J. (2001). *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press.

[28] Shah, A., Wilson, A., and Ghahramani, Z. (2014). Student-t processes as alternatives to gaussian processes. In *Artificial Intelligence and Statistics*, pages 877–885.

[29] Sharpe, W. F. (1964). Capital asset prices: A theory of market equilibrium under conditions of risk. *The journal of finance*, 19(3):425–442.

[30] Shawe-Taylor, J. and Cristianini, N. (2004). *Kernel methods for pattern analysis*. Cambridge university press.

[31] Shreve, S. E. (2004). *Stochastic calculus for finance II: Continuous-time models*, volume 11. Springer Science & Business Media.

[32] Smith, W. (2010). On the simulation and estimation of the mean-reverting ornstein-uhlenbeck process. *Commodities Markets and Modelling*.

[33] Spooner, T., Fearnley, J., Savani, R., and Koukorinis, A. (2018). Market making via reinforcement learning. *arXiv preprint arXiv:1804.04216*.

[34] Sutton, R. S. and Barto, A. G. (1998). *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge.

[35] Uhlenbeck, G. E. and Ornstein, L. S. (1930). On the theory of the brownian motion. *Physical review*, 36(5):823.

[36] van den Berg, T. Calibrating the ornstein-uhlenbeck (vasicek) model. `https://www.sitmo.com/?p=134`. Accessed: 2017-08-02.

[37] Vasicek, O. (1977). An equilibrium characterization of the term structure. *Journal of financial economics*, 5(2):177–188.

[38] Von Neumann, J. and Morgenstern, O. (2007). *Theory of games and economic behavior (commemorative edition)*. Princeton university press.

[39] Watkins, C. J. and Dayan, P. (1992). Q-learning. *Machine learning*, 8(3-4):279–292.

[40] Williams, C. K. (1998). Prediction with gaussian processes: From linear regression to linear prediction and beyond. *Nato asi series d behavioural and social sciences*, 89:599–621.

[41] Williams, C. K. and Rasmussen, C. E. (1996). Gaussian processes for regression. In *Advances in neural information processing systems*, pages 514–520.