Designing an Optimizing Algorithm for Algorithmic Trading Through Reinforcement Learning

Max Gillham
Bryony Schonewille
Eric Grewal
Yuankang Zhang

April 16, 2019

Abstract

This report investigates the algorithmic trading problem where profit is maximized while risk is minimized. This problem was posed as a stochastic control problem in order to design different solutions with various mathematical machinery. A starting initial capital is invested in different stocks. The capital is partitioned such that money is invested in each stock, with an additional option not to invest at all. After an iterative design process, Q-learning techniques were implemented to design the model that would solve the problem. Testing was then performed on both independently identically distributed and Markovian simulated data. The algorithm was then applied to historical data of three stocks. After testing on real data, it was determined that a Markovian model best resembles the given stock market data, leading to a better design solution. Throughout the engineering design process, the team considered social, economic, environmental, and regulatory concerns when making decisions regarding optimal solutions.

Acknowledgements

The team would like to give a special thanks to Professor Yüksel for all his patience, help, knowledge and guidance through out this project. Professor Yüksel succeeded in teaching us a great deal over the course of our final year at Queen's, and allowed us to use many of the skills we had developed during our undergraduate careers in Applied Mathematics Engineering

Contents

1 Introduction

1.1 Portfolio Management Problem

Consider a trading environment, where the common goal of any investor is to obtain added utility or value by buying, selling and holding assets. These assets can have fixed return rates, like bonds, or be volatile, like stocks, and are issued by the government or corporations. In this kind of an environment, the return rate for stocks are erratic and difficult, if not impossible, to predict. An investor may be able to make informed guesses or assumptions based off of historical data or trends for specific situations, however, the actual result is always unknown to everyone since there are so many factors contributing to the change of stock return rates. These factors include economic fluctuations, large trade, rumors, or inter-company relationships. Due to this inherent randomness in return rates, it is a real challenge to determine how one should distribute their funds between assets, a problem made worse as the amount of assets available increases. For long term investors, capital is largely allocated into fixed assets where the return on investment is guaranteed. For other investors, the distribution of capital into stocks may be desirable to earn larger sums of money if the investor accepts the risk that is associated with stock trading. This capital allocation problem to increase value while reducing risk is known as the portfolio management problem. This paper investigates how an algorithm can be employed to determine a capital allocation policy to find an optimal solution to the portfolio management problem.

1.2 Algorithmic Trading

Algorithmic trading is a method of making a large quantity of trades by using an algorithm to generate trading instructions. It should be noted that algorithmic trading is not necessarily just for generating profit from the stock market. Rather, it is a way to evaluate many options while reducing manual workload and thus labour costs. Many different forms of algorithmic trading exist, and one specific example that aims to maximize profit is high frequency trading. In high frequency trading, a large quantity of stocks are purchased and subsequently sold with a high turnover rate. This kind of algorithmic trading will be the focus for the design solution that is constructed.

1.3 Motivation

The team's objective is to redesign how an investor manages their portfolio. For an investor in high frequency trading, their goal is to maximize the capital attained from a spread of stochastic and fixed assets, while actively trying to select trade ideas to match risk requirements.

One problem lies within the randomness of various assets in a given portfolio. The value of a given stock is influenced by a multitude of factors such as a company's proposed business strategy, debt issuance, news reports, or credit ratings. These assets are of high entropy and the properties that drive investments are not easily modelled. These properties are the main difficulty encountered when trading short term stochastic assets.

The other problem an investor encounters is deciding how to spread capital across a series of financial instruments. While trying to increase the value of a portfolio, they also have to control the amount of risk accepted. The rate of return on non-stochastic assets is fixed and therefore will reduce the amount of risk a portfolio encounters. By hedging some determined fraction of capital, an investor can control and minimize a portfolio's risk.

1.4 Existing Practice

Electronic trading started to emerge in 1971 with the National Association of Securities Dealers Automated Quotation becoming the first electronic stock market. By 1990 many major security exchanges were fully electronic and matching algorithms for determining prices were already being implemented. Over the years, as stock buyers became increasingly aware of trading costs, brokers introduced methods that would allow for direct market access, such as algorithmic trading. This eliminated the need for a market intermediary in many cases, reducing the trading costs for the buyer. Different trading algorithms that are in place can be qualified and categorized by:

- the strategies they employ
- the algorithmic complexity
- the algorithm's mechanics

Impact-driven algorithms target trades that could considerably change the market price and attempt to minimize the effect of the trade on the asset's price. Cost-driven algorithms attempt to minimize the overall trading cost by minimizing implicit costs, like the impact-driven algorithms, as well as the explicit costs. Newsreader algorithms are a more recent development and like the title suggests, they perform investment decisions based on inputs like news and other real-time distributed information [9]. Some of the most popular algorithms for are called automated market making, low latency arbitrage, and liquidity rebate trading. These methods employ "neural networks" and "genetic algorithms" so that the rules and assumptions that the market could operate on can compete and combine to create the most successful outcomes [8].

2 Design Problem Analysis

The team would like to propose a solution to solve the problem of how a fixed amount of capital can be best portioned across a set of assets to maximize the expected value that is returned. The problem naturally gave way to the following list of constraints:

- the mathematical model for the stock market is unknown since there are many visible and non-visible factors such as company reputation, economic stability, large trades, company interactions, etc. that contribute to the changes of the stock return rates
- the only available information is a set of past return rate data, which may be limited in the amount of data points available
- the computational memory needed for the solution must be minimized
- the time needed for the solution to operate must be minimized
- the solution of investing in stocks must be more profitable for the user than just investing in bonds

2.1 Iterative Design

To design a solution to meet these criterion, the iterative engineering design process was used. First due to the constraints and the stochastic nature of stock price, the problem was framed as a stochastic control problem. A centralized system where a single decision maker acting on the system will be used. In this given scenario, the single decision maker will not have access to the exact probabilistic characterizations of the environment. This follows directly from the fact that the stock market cannot be predicted.

Next, potential design options were considered and investigated. The decision maker will examine the data and attempt to determine effective policies for the stochastic control problem. During the problem solving process, the team investigated Q-learning and dynamic programming as different tools. Q-Learning is a known method that can be implemented to determine these policies. Dynamic programming has also been considered, as its role in high frequency trading has been documented.

After investigation, the team customized the selected tool, Q-learning, and designed our solution for the problem by applying iterative design process. The team iteratively tested and evaluated different mathematical model decision and different quantization methods used in the Q-learning algorithms to best fit the problem. Finally, analytic tests were conducted to validate the design and to allow for design adjustments to be made. In summary, following iterative design process is applied to solve the problem:

- Design mathematical model for the problem
- Research potential solutions
- Simulate the solution with generated data

- Evaluate and select optimal solution based on the simulation
- Iteratively design and evaluate the solution with real data
  - Customize the solution to adapt to this unique problem
  - Customize the solution to adapt to the real data scenario
  - Evaluate the design and repeat above process
- Discuss solution performance and design improvement
- Discuss and summarize the economic, environmental, social and regulatory factor the team has considered when designing the solution for the problem

Please see further discussion in the following sections.

3 Mathematical Design Formulation

3.1 State Space

The physical representation of the variables used in the state space are defined below.

- Let X_t = the amount of capital the portfolio has at time t, $X_t \in \mathbb{R}_{\geq 0}$
- Let n_s = the number of stocks available for investment, $n_s \in \mathbb{R}_{\geq 0}$
- Let n_b = the number of bonds available for investment, $n_b \in \mathbb{R}_{\geq 0}$
- Let n = the total number of investments available, $n = n_s + n_b$, $n \in \mathbb{R}_{\geq 0}$
- Let $i = 0, 1, ..., n$ be the index of the investments
- Let r_t^i = the return rate of the i^{th} stock or bond at time t, reasonably $r_t^i \in [-1, 1]$, but could exist in $\mathbb{R}$

For formulaic simplification, take r_t^0 to be the return rate for choosing not to invest. With this definition, $r_t^0 = 0 \quad \forall\, t \in \mathbb{R}_{\geq 0}$. With all of these statements, the state space $\mathbb{X}$ is then as follows:

$$\mathbb{X} = \mathbb{R}_{\geq 0} \times [-1, 1]^{n+1} \tag{1}$$

which can be extended to:

$$\mathbb{X} = \mathbb{R}^{n+2} \tag{2}$$

This means that an element of the state space at time t has the below form:

$$x = (X_t, r_t^0, r_t^1, ..., r_t^n) \tag{3}$$

Therefore an element of the state space is an (n+2)-tuple.

3.2 Action Space

For the action space, the only other variable that needs to be defined is u_t^i.

- Let u_t^i = the proportion of capital that is invested into investment i at time t, $u_t^i \in [0,1]$ and $\sum_{i=0}^{n} u_t^i = 1$

The action space is then:

$$\mathbb{A} = \{(u_0, u_1, ..., u_n) \in [0,1]^{n+1} \mid \sum_{i=0}^{n} u_i = 1\} \tag{4}$$

This means that an element of the action space at time t has the below form:

$$u = (u_t^0, u_t^1, ..., u_t^n) \tag{5}$$

Therefore an element of the action space is an (n+1)-tuple.

3.3 Transition Kernel

The relation that governs the evolution of the capital from time t to time t+1 is seen in the equation below.

$$X_{t+1} = \sum_{i=0}^{n} X_t \; u_t^i \; (1 + r_{t+1}^i) \tag{6}$$

Let $R_t^i = 1 + r_t^i$, then:

$$X_{t+1} = \sum_{i=0}^{n} X_t \; u_t^i \; R_{t+1}^i \tag{7}$$

Since X_t is independent of i, the proportion $\frac{X_{t+1}}{X_t}$ is:

$$\frac{X_{t+1}}{X_t} = \sum_{i=0}^{n} u_t^i \; R_{t+1}^i \tag{8}$$

3.4 Reward Function

For an investor, the time interval to obtain some utility of capital is often unknown. The benefit of the amount of capital earned at each time iteration is also relative to the amount of funds the investor currently has. The team chose the reward function $c : \mathbb{X} \times \mathbb{A} \longrightarrow \mathbb{R}$ to be $c = \log(x)$ as part of our design. The log as a reward function captures the decline of monetary valuation as more money is accumulated. This function also allows for its unique properties to be used. For this problem, we are looking for max $E[\log(X_T)]$ for some $T \in \mathbb{Z}_{\geq}0$.

$$E[\log(X_T)] = E[\log(\frac{X_T}{X_{T-1}} \cdot \frac{X_{T-1}}{X_{T-2}} \cdot \dots \cdot \frac{X_1}{X_0} \cdot X_0)] \tag{9}$$

$$= E[\log(\prod_{k=0}^{T-1} \frac{X_{k+1}}{X_k}) + \log(X_0)] \tag{10}$$

$$= E[\sum_{k=0}^{T-1} \log(\frac{X_{k+1}}{X_k}) + \log(X_0)] \tag{11}$$

Now using Equation 8 from the relation of the future and current capital, the equation can be rewritten as the following.

$$= E[\sum_{k=0}^{T-1} \log(\sum_{i=0}^{n} u_k^i R_{k+1}^i) + \log(X_0)] \tag{12}$$

$$= E[\sum_{k=0}^{T-1} \log(\sum_{i=0}^{n} u_k^i R_{k+1}^i)] + E[\log(X_0)] \tag{13}$$

$$= \sum_{k=0}^{T-1} E[\log(\sum_{i=0}^{n} u_k^i R_{k+1}^i)] + \log(X_0) \tag{14}$$

Equation 14 depicts the general cost function for this system. Let R_k be the column vector of all the return rates at time k and u_k be the vector of all proportions. By conditioning each past vectoral return rate, the law of iterated expectations can be used to expand the expression. Allowing each of the tilde variables to be one possible value the non-tilde variable can obtain, this equation then becomes:

$$\begin{aligned}
&= E[\sum_{k=0}^{T-1} E[\log(u_k^\top R_{k+1}) | R_k, R_{k-1}, \dots, R_0]] + \log(X_0) \\
&= E[\sum_{k=0}^{T-1} E[\log(u_k^\top R_{k+1}) | R_k = \tilde{R_k}, R_{k-1} = \tilde{R_{k-1}}, \dots, R_0 = \tilde{R_0}]] + \log(X_0) \\
&= E[\sum_{k=0}^{T-1} [\sum_{\tilde{R_{k+1}}} P(R_{k+1} = \tilde{R_{k+1}} | R_k = \tilde{R_k}, R_{k-1} = \tilde{R_{k-1}}, \dots, R_0 = \tilde{R_0}) \\
&\quad \cdot \log(u_k^\top \tilde{R_{k+1}})]] + \log(X_0)
\end{aligned} \tag{15}$$

Equation 15 holds true for any distribution of the return rates. If $\{R_k\}_{k=0}^n$ is taken to be a Markov Source with memory M, the equation remains the same. For a Markov Source with memory M, where $M \leq k$, the equation is:

$$\begin{aligned}
Mem\ M = E[\sum_{k=0}^{T-1} [\sum_{\tilde{R_{k+1}}} P(R_{k+1} = \tilde{R_{k+1}} | R_k = \tilde{R_k}, \dots, R_{k-M} = \tilde{R_{k-M}}) \\
\cdot \log(u_k^\top \tilde{R_{k+1}})]] + \log(X_0)
\end{aligned} \tag{16}$$

This concept can reduce the equation if the source is Markov with a memory of order 1.

$$Mem\ 1 = E[\sum_{k=0}^{T-1}[\sum_{\tilde{R_{k+1}}} P(R_{k+1} = \tilde{R_{k+1}}|R_k = \tilde{R_k}) \cdot \log(u_k^\top \tilde{R_{k+1}})]] + \log(X_0) \tag{17}$$

If instead the return rates are independent and identically distributed (IID) for each time step, the equation simplifies to the below equation.

$$IID = E[\sum_{k=0}^{T-1}[\sum_{\tilde{R_{k+1}}} P(R_{k+1} = \tilde{R_{k+1}}) \cdot \log(u_k^\top \tilde{R_{k+1}})]] + \log(X_0) \tag{18}$$

$$= E[\sum_{k=0}^{T-1}[\sum_{\tilde{R}} P(R = \tilde{R}) \cdot \log(u_k^\top \tilde{R})]] + \log(X_0) \tag{19}$$

In this equation the individual R_{k+1}'s are replaced with just R to symbolize the identical distribution of all return rates at each time step.

4 Solution Design Tool Selection

4.1 Analytic Solution - Dynamic Programming

After the team derived the evolution model, one of the solution options we discussed was to use dynamic programming to solve the problem analytically. Recall that this project aims to determine the optimal policy which maximizes the overall profit by maximizing the profit gain in each investment episode:

$$E[\log(\sum_{i=0}^{n} u_k^i R_{k+1}^i)|R_{[0,t]}, u_{[0,t-1]}^i] \tag{20}$$

We assume that at the end of the time, the investor is no longer investing so the optimal action at t = T-1 is to not invest. And therefore the reward at t = T-1 will be 0:

$$J_{T-1}(X_{T-1}) = \max_{u_{T-1}\in\mathbb{A}} E[\log X_{T-1}] \tag{21}$$

$$= \max_{u_{T-1}\in\mathbb{A}} E[\log \sum_{i=0}^{n} u_{T-1}^i R_{T-1}^i] \tag{22}$$

$$= \max_{u_{T-1}\in\mathbb{A}} E[\log \sum_{i=0}^{n} 0 \times R_{T-1}^i] = 0 \tag{23}$$

At t = T-2, the optimal policy is to maximize the expecting reward at t = T-2.

$$J_{T-2}(X_{T-2}) = \max_{u_{T-2}\in\mathbb{A}}(E[\log\sum_{i=0}^{n} u^i_{T-2}R^i_{T-2} + J_{T-1}(X_{T-1})|R_{[0,T-2]}, u_{[0,T-3]}]) \tag{24}$$

$$= \max_{u_{T-2}\in\mathbb{A}}(E[\log\sum_{i=0}^{n} u^i_{T-2}R^i_{T-2}|R_{[0,T-2]}, u_{[0,T-3]}]) \tag{25}$$

Recursively, the dynamic programming recursion is obtained.

$$J_t(X_t) = \max_{u_t\in\mathbb{A}} E[\log X_t] \tag{26}$$

$$= \max_{u_t\in\mathbb{A}}(E[log(\sum_{i=0}^{n} u^i_k R^i_{k+1})|R_{[0,t]}, u^i_{[0,t-1]}]) \tag{27}$$

However, since the model of the stock return rates is unknown, the optimal policy can not analytically be determined using this method. However, in reality, we generally cannot have a proper model to actually reflect the financial system, therefore this dynamic programming stated as above cannot be applied.

4.2 Model-free Solution - Q-learning

Model-based algorithms are heavily dependent on the ability to be able to estimate the model well. If the true transition probabilities are drastically different from what is assumed, the estimation of the expected value is unlikely to do well. Therefore, the team began to search for model-free solutions to solve the problem. By model-free, the team means the solution does not require the knowledge of transition kernel or reward function. We discovered Q-learning was one of the most commonly used model-free algorithm in the field. Therefore, we began to study, testify, and modify Q-learning to adapt our project.

Q-Learning is a stochastic approximation algorithm that is used to determine an optimal policy given a simulated path[16]. The optimal policy is determined using a table, called a Q-table, where the rows are states and the columns are actions. The algorithm calculates an expected reward for each state-action pair and stores it in the Q-table. The Q-table is updated after every episode of training. Once training is complete, the optimal policy is the set of state-action pairs obtaining the maximum expected reward. Q-table update algorithms, also called Q-functions, are based on a reformulation of the Bellman equation. The Bellman equation uses the dynamic programming paradigm to provide a recursive definition for the optimal Q-function.

$$Q_{t+1} = Q_t(x_t, u_t) + \alpha_t[r_t(x_t, u_t) + \beta \times \max_v(Q_t(x_{t+1}, v)) - Q_t(x_t, u_t)] \tag{28}$$

Here, Q_t represents the Q-table matrix at time t, $x_t \in \mathbb{X}$ is the state at time t, $u_t \in \mathbb{A}$ is the action taken at time t, r_t is the reward of taken action u_t

after observing state x_t at time t., and v is an action that maximizes the future estimate Q value under state x_{t+1}. Each α_t is a step-size coefficient which represents how much the algorithms trust the future estimation. The larger the (α_t) is, the higher trust the algorithms put into the estimate of the future Q-value for time t. β is a learning rate coefficient which represents how far in the future the algorithms is predicting. The larger the β is, the further the algorithms is predicting the future at time t. Note that, in order for Q-table to converge to a fixed optimal solution, each state-action pair must be visited infinitely many times[15].

Equation 28 is applied to data based on Markovian model. In order to apply Q-learning to data with an IID model, modifications should be made in order to address that independence characteristic of the IID model. Therefore, the Q-function for this model is:

$$Q_{t+1} = Q_t(u_t) + \alpha_t[r_t(u_t) + \beta \times \max_{u_t}(Q_t(u_t)) - Q_t(u_t)] \tag{29}$$

where past states are not observable.

4.3 Comparing Dynamic Programming and Q-learning

When comparing the solution options, the team considered the following factors:

- ease of implementation
- efficiency of calculation
- ease of interpretation
- performance

In order to determine which model the team should focus on, the following comparison was established:

Trade-off Analysis		
Criteria	Dynamic Programming	Q-learning
Ease of Implementation	Relatively easy to determine the analytic function but **impossible to obtain the proper distribution model for the system**.	Relatively complex to implement. Extra effort is required since the state space and action space must be quantized.
Efficiency of Calculation	High efficiency of calculation through an analytic process.	In order for the Q-table to converge to a fixed optimal solution, each state-action pair must be visited infinitely many times. Therefore, the algorithm requires large amounts of data and calculation.
Ease of Interpretation	Easy to interpret due to its analytic characteristic.	Easy to interpret due the well-defined analytic Q-function.
Performance	Provide accurate results through an analytic process.	Provide optimal results given enough training.

According to the comparison above, we can see that dynamic programming as an analytic approach can provide more accurate result, is easier to understand and is more efficient in terms of calculation. However, due to fact that no proper distribution model can be determined for the return rate to accurately reflect the financial system, the advantages of dynamic programming lose their value. Therefore, the team decided to choose Q-learning as a part of the design. The algorithm would need to be modified to better adapt to the system. For example, to address the constraint that the Q-table must be finite, quantization of both state space and action space must be performed.

5 Design Experimentation and Simulations

In order to verify the effectiveness of Q-learning for investment problems, the team first simulated stock data to test the Q-learning algorithm for both the IID data model and the Markovian data model.

To better address the constraints of Q-learning, the team designed the environment in the following way. Please note that the assumptions and design used in this simulation stage might be modified or improved for training and testing on the real data in the later phases. For example, a more robust quantization process would be established.

5.1 Simulation Assumptions and Design

5.1.1 Modeling the Stock Environment

When modelling the stock environment, the team created four equities. For a given equity, the team generated its daily return rates to satisfy either IID or Markov properties. The set of daily return rates for all equities were then fed into the Q-learning algorithms as the observation of the environment.

When quantizing the state space, the team assumed any return rate $r \in [-0.05, 0.05]$ to be $r = 0$, $r \in [-1, -0.05)$ to be $r = -1$ and $r \in (0.05, 1]$ to be $r = 1$. This assumption was made due to the fact that an effective Q-learning process requires the state space to be a finite space.

5.1.2 Modelling the Interaction with the Environment

In the physical world, stock buyers invest in equities by dividing their existing capital for each equity. Therefore, the action could be modeled as the proportions of capital for each equity where all proportions sum up to 1. However, to simplify the experiment, the team simplified the action by limiting the possible choice of proportion as 100% and 0%. This meant that, for each time step, the machine would invest 100% of the capital into one of the equities and invest 0% of the capital into other equity. This assumption was not just to simplify the situation but also to satisfy the finite action space criterion.

Secondly, the team assumed the investment cycle to be one day. This means the buyer would not hold any equity for more than one day and he buys or sells the equities on a daily basis.

For the goal of the interaction, the team assumed the winning condition was to double the capital. In other words, the team was testing if the Q-learning can make correct decisions to double the initial capital.

5.1.3 Summary of Key Assumptions in Simulation

- Environment Design: Any return rate $r \in [-0.05, 0.05]$ was observed to be $r = 0$, $r \in [-1, -0.05)$ was observed to be $r = -1$, and $r \in (0.05, 1]$ was assumed to be $r = 1$
- The return rate is modeled as daily return rate.
- The proportion of the capital investing can only be 100% or 0%, i.e buyers can only buy one equity at any time t.
- The investment cycle is assume to be one day. The buyer would not hold any equity for more than one day and he buys or sells the equities in a daily basis.

5.2 IID Model Simulation

5.2.1 Data Simulation Process

In an IID data set, the stock return rate at time t is independent of the stock return rate at time $t-1$ for a given stock. Therefore, when generating the data points of stocks, the team generated the data points that were uniformly distributed over the interval $[-1, 1]$. In other words, any value within the given interval was equally likely to be generated. When generating the data for bonds, the team generated a return rate that was consistently $r = 0.05$. When generating the data for the dummy equity indicating Not-investing, the team generated a return rate that was consistently $r = 0.00$.

5.2.2 Experiment I: Four IID Equities

In this experiment, the team generated IID equities as four general stocks. Every one of the stocks has a IID source and so no correlation can be found between any two data points.

The learning result met the team's expectation that the Q-learning algorithm's strategies demonstrated randomness and could not determine which stock was the most likely to gain profit.

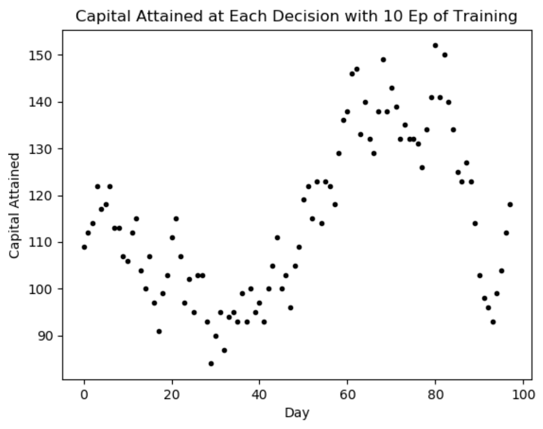

Figure 1: Four Stocks with IID Source

The result in Figure 1 demonstrated that Q-Learning could not effectively determine an effective investment policy within 100 days of investment.

5.2.3 Experiment II: Three IID Stocks with One IID Dummy Equity

In this experiment, the team generated IID equities as one dummy equity, not-investing, and three general stocks. The result met the team's expectation that the Q-learning algorithm could indicate that consistently choosing dummy equity was the best strategy - because dummy equity was the most predictable equity and thus the safest one.

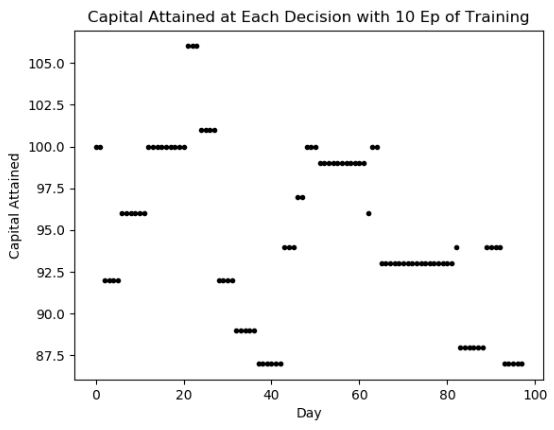

Figure 2: Three IID Equities with One of Them Indicating Not-investing

After the team introduced a dummy stock, Figure 2, with constant return rate 0% among the IID sources in order to represent the choice of not investing, the Q-learning table often computed a maximum expected future reward by choosing not to invest.
By examining the Q-table, the team can see that the program had learned to reward the action of choosing dummy equity "Equity 0". In other words, it had successfully developed a policy of choosing the most predictable equity, the dummy equity, so that the risk could be reduced.

5.2.4 Experiment III: Three IID Stocks with One of IID Bond

In this experiment, the team generated IID equities as one bond and three general stocks. The result met the team's expectation that the Q-learning algorithm consistently chose the bond as the best strategy, since equity was the most predictable equity, thus the safest one, and it provided a greater return than a dummy would.

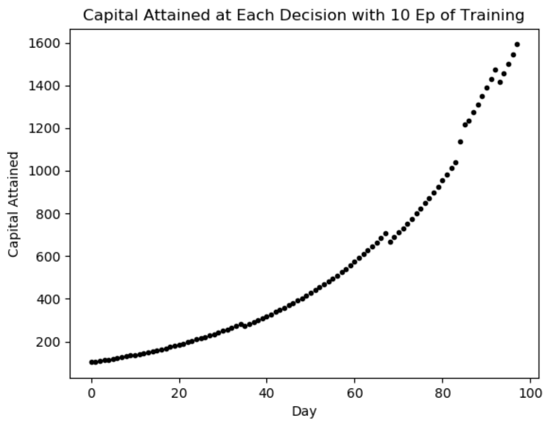

Figure 3: Four IID Equities with One of Them Indicating Bond Equity

As shown in Figure 3, if the team introduced a fixed asset, or bond, the Q-learning developed a policy that it would often choose to invest in it, rather than the stocks with IID return rates.

5.2.5 Experiment IV: Two IID Stocks with One IID Bond and One IID Dummy Equity

In this experiment, the team generated IID equities as one dummy equity, one bond, and two general stocks. As expected, due to the predictability, the algorithm chose bond and not-invest most of the time. Comparing with Experiment III, having the option of not-invest reduces the profitability.

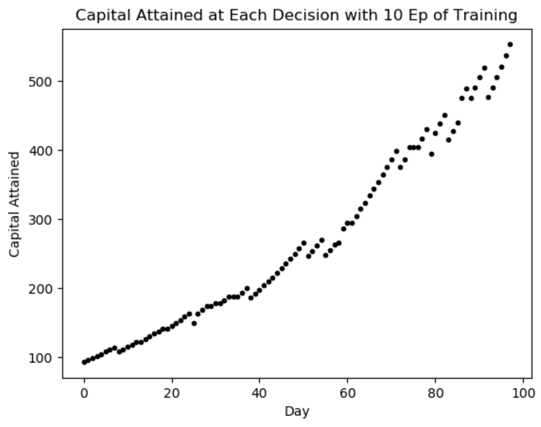

Figure 4: Four IID Equities with One of Them Indicating Not-investing and One of Them Indicating Bond Equity

When both dummy equity and bond were introduced, the Q-learning developed a strategy similar with Experiment III: Investing in bond, while also adding action of not-investing sometimes, Figure 4.

5.3 Markov Model Simulation

5.3.1 Data Generating Process

In a Markov data set, the stock return rate at time t is dependent on the stock return rate at time $t-1$ for a given stock.

Therefore, when generating the data points of equities, the team first made an assumption for the potential return rates of the given stock, then the team made an assumption for the 1-step transition matrix.

For this set of the experiment, the team generated the data for three kinds of equity: bond, stocks, and dummy equity indicating not-investing.

For a low reward but more predicable equity, a bond, the team assumed the possible return rates:

$$\begin{bmatrix} -0.05 & 0.00 & 0.05 \end{bmatrix}$$

with the transition matrix:

$$\begin{bmatrix} 0.8 & 0.1 & 0.1 \\ 0.1 & 0.8 & 0.1 \\ 0.1 & 0.1 & 0.8 \end{bmatrix}$$

The team denoted this kind of equity as “Stock 0”.

For a larger reward but less predictable equity, the team assumed the possible return rates to be:

$$\begin{bmatrix} -0.25 & 0.00 & 0.25 \end{bmatrix}$$

with the transition matrix

$$\begin{bmatrix} 0.8 & 0.1 & 0.1 \\ 0.1 & 0.8 & 0.1 \\ 0.1 & 0.1 & 0.8 \end{bmatrix}$$

The team denoted this kind of equity as “Stock 1”.

For a big reward but even less predictable equity, the team assume the possible return rates to be

$$\begin{bmatrix} -0.05 & 0.00 & 0.05 \end{bmatrix}$$

with the transition matrix

$$\begin{bmatrix} 0.2 & 0.4 & 0.4 \\ 0.4 & 0.2 & 0.4 \\ 0.4 & 0.4 & 0.2 \end{bmatrix}$$

The team denoted this kind of equity as “Stock 2”.

For a big reward but even less predictable equity, the team assume the possible return rates to be

$$\begin{bmatrix} -0.25 & 0.00 & 0.25 \end{bmatrix}$$

with the transition matrix

$$\begin{bmatrix} 0.2 & 0.4 & 0.4 \\ 0.4 & 0.2 & 0.4 \\ 0.4 & 0.4 & 0.2 \end{bmatrix}$$

The team denoted this kind of equity as “Stock 3”.

For the dummy equity indicating not-investing, the team generated the return rate that was consistently $r = 0.00$. The team denoted this kind of equity as the “Dummy”.

5.3.2 Experiment I: Four Stocks of Markov Source

Consider now the return rates being modeled by a Markovian process of memory 1. As expected, this yielded better results than the IID scenario.

In this experiment, the team generated four Markovian source stocks: “Stock 0”, “Stock 1”, “Stock 2”, “Stock 3”. Therefore in this case the algorithm would not have the option of not-invest. Obviously, “Stock 3” with largest possible return rate and highest predictability was the optimal solution. The following figure shows what policy Q-learning developed.

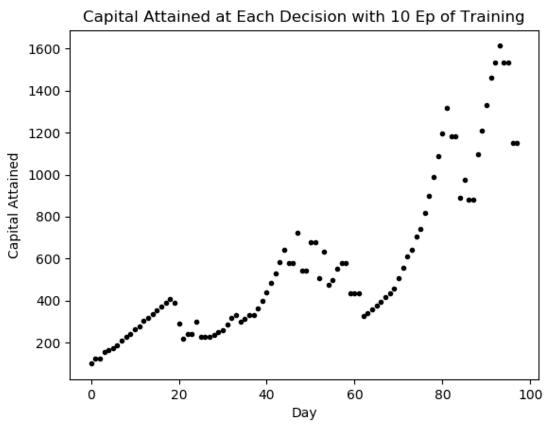

Figure 5: Four Markov Source Stocks

As expected, the program developed a policy of investing the stock with highest probability of higher profit, this is seen in Figure 5. We can also examine the Q-table to see if the program was making the correct decision in the process.

5.3.3 Experiment II: Three Markov Source Stocks With One Dummy Equity

In this experiment, the team generated three Markov source stocks with one Dummy equity. As expected, this yields better results than the IID process. The capital earned in this scenario was the highest amount among all the test as shown below.

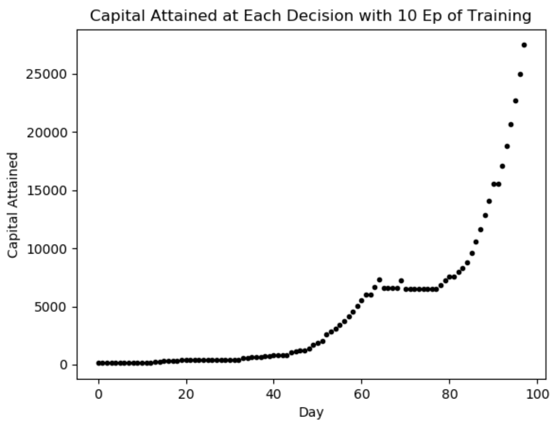

Figure 6: Three Markov Source Stocks with One Dummy Equity

The capital earned in this trial was higher than the previous case. The team believed that the existence of a Dummy equity in this experiment helped the algorithm to preserve capital when the risk of investing was high, which helped to achieve higher profit.

5.4 Simulation Conclusion and Analysis

According to the experiment's result, the team concluded that Q-learning worked better for Markov data and was less desirable for IID data in general. For a Markov data set, the Q-learning algorithm could always learn the best strategy. However, for an IID source, the effectiveness of the Q-learning algorithm depended on the combination of the equities - most of the time, it tended to choose the best or the safest option, either to not-invest or the bond.

Note that, some of the experiment assumptions or implementation decisions might have limited the discovery or caused unexpected results for the experiments.

Recall that any return rate $r \in [-0.05, 0.05]$ was assumed to be $r = 0$, $r \in [-1, -0.05)$ was assumed to be $r = -1$, and $r \in (0.05, 1]$ was assumed to be $r = 1$. This threshold of discretizing the return rates might not best simulate reality. Under this threshold, an equity with $r = 0.06$ and an equity with $r = 1$ would be considered as two equities with the same profitability at a given time t. The team might need to consider the effectiveness of this threshold.

Another decision was that the return rate was modeled as a daily return rate. In reality, it is obvious that the stock return rate varies more within one day than within one hour. In other words, the stock return rate generated in the current experiments was more similar to an IID random variable. If hourly data was observed and in a high-frequency trading scenario, Q-learning is expected to work better.

In terms of action, the team assumed that buyers can only buy one equity and on a daily basis. This reduces the possible combination of actions and thus reduces the power of Q-learning.

6 Design Implementation with Real Data

6.1 Design Assumption and Decisions

The success of obtaining a optimal solution for our problem relies on the convergence of the Q-learning. The convergence of the Q-learning has been proved by many mathematicians under different setup of assumptions. The team adopted the proof[15] by Tsitsikis in which the assumptions were relatively more general and were more suitable for our case. Tsitsikis's proof of convergence extended from the classical proof. It allowed the cost per stage (the reward in our case) to be unbounded random variables; it allowed the decision on which the action to take next to depend on past experience (knowing the past return rate in our case); it also allows α to be a random variable (therefore we can test different α values in our design process.)

According to Tsitsikis, in order for $Q_t(x_t, u_t)$ converges to the optimal $Q(x, u)^*$, several conditions need to be satisfied:

Assumption 1: Even though information can be outdated, old information will be eventually discarded.

Assumption 2: The decision on which the action to take next can depend on past experience.

Assumption 3: For every i, $\sum_{t=0}^{\infty} \alpha_i(t) = \infty$, which is a standard assumption for stochastic approximation algorithms.

Assumption 4: $\beta < 1$.

Assumption 5: Denote the mapping $r_t(x_t, u_t) + \beta \times \max_v(Q_t(x_{t+1}, v))$ as $F(Q)$.

We assume there exists a vector $Q^* \in \mathbb{R}^n$, a positive vector v, and a scalar $\beta \in [0, 1)$, such that $||F(Q) - Q^*||_v <= \beta||Q - Q^*||_v, \forall Q \in \mathbb{R}^n$.

Assumption 6: The state space and action space are finite.

Under these assumptions, Q-learning converges. We will not further discuss about the proof of convergence here. In order to satisfy the above conditions, several design decisions were made when implementing the Q-learning algorithms:

- In our design, all state-action pairs were updated in every episode, which means no outdated information existed. Therefore Assumption 1 was satisfied.
- The team established testing in Markov model which satisfies Assumption 2 by definition of Markov process. When the team test the algorithms with I.I.D. model, this assumption is not satisfied and the optimizing result was not satisfactory either. Further discussion will be provided later.
- For each real stock, over 25000 data point (return rate in our case) were collected so that each state would be simulated to be visited infinitely many times, which satisfies Assumption 3.
- β was taken to be 0.85, which satisfies Assumption 4. The value was determined to be the best value after several testing.
- $\alpha = 0.001 < 1$. The value was determined to be the best value after several testing.
- Quantization was performed for state space and action space so that Assumption 6 is satisfied. Further discussion of quantization will be provide in the following sections.

Based on the above design decisions, assumptions 1, 2, 3, 4 and 6 were satisfied in our design for the Q-learning to converge. However, Assumption 5 ($F(Q)$ was not fully addressed. Further discussion will be provided in section Design Evaluation. Further discussion of the last decision, quantization of state space and action space, is provided in the following sections.

6.2 Quantization of State Space and Action Space

In order to be able to perform calculations, the state and action spaces needed to be quantized into finite sets. This allows for analysis to be performed in finite time and allows for Q-learning to be employed.

6.2.1 State Space Quantization

A component of a tuple of the state space has no dependence on the other components, therefore the state space can be quantized with a linear quantizer. The first quantization method that was considered was the Lloyd max quantization algorithm. This algorithm was chosen first because it only utilizes a training set, not a distribution and in the definition of this problem, it is assumed that there is accessible training data for the return rates. A Lloyd-Max quantizer is a quantizer that satisfies the centroid and nearest neighbour conditions. Those conditions are presented below.

Nearest Neighbour Condition: For an n-level quantizer with with output points $y_1, ..., y_n$, any quantizer with regions such that

$$R_i \subset \{x : d(x, y_i) \leq d(x, y_j), j = 1, ..., n\}\ for\ i = 1, ..., n$$

has minimum distortion.

Centroid Condition: For an n-level quantizer with given regions $R_1, ..., R_n$, then the quantizer with output levels given by

$$y_i = \arg\min_{y \in \mathbb{R}} E[d(X, y)|X \in R_i]\ for\ i = 1, ..., n$$

has minimum distortion.

These two conditions are iterated through in an alternating fashion to first optimize the regions given the codepoints, then the codepoints given the new regions. The full algorithm is seen below.

Step 1: For a training set T, initial codepoints $\mathscr{C}_1 = \{y_1^{(1)}, ..., y_n^{(1)}\}$, and threshold $\epsilon > 0$, start with $m = 1$ and the distortion is $D_1 = E[d(X, Q^{(1)}(X))]$

Step 2: For the current codebook $\mathscr{C}_m$,

- Use the nearest neighbour condition to find the optimal set of partitions $R_1^{(m)}, ..., R_n^{(m)}$
- Find the next optimal codebook $\mathscr{C}_{m+1}$ using the centroid condition

Step 3: Compute $D_{m+1} = E[d(X, Q^{(m+1)}(X))]$

Step 4: If $\frac{D_m - D_{m+1}}{D_m} < \epsilon$, then $\mathscr{C}_{m+1}$ is the optimal quantization. Otherwise, increment m and return to Step 2

While this algorithm is very useful, it could not be applied to the state space since the dynamics of the system do not match the mean squared distortion used in the Lloyd-Max algorithm. Alternatively, a uniform quantizer was explored. A uniform quantizer cuts the interval into equal weighted pieces. For an n-level quantizer with training set T where $|T| = s$ and intervals R_i are bounded by x_{i-1} and x_i, then the bounding points are:

$$x_i = \frac{T(i\frac{s}{n}))) + T(i\frac{s}{n} + 1))}{2} \tag{30}$$

and the output points are:

$$y_i = \frac{x_{i-1} + x_i}{2} \tag{31}$$

For our purposes, any points outside of the interval $[-1, 1]$ are mapped to the nearest interval. This means $x_0 = -\infty$ and $x_n = \infty$ but they are taken to be -1 and 1 respectively for the output point calculations. This quantizer fit the system and was the method of quantization that was used going forward.

6.2.2 Action Space Quantization

The action space quantization needed to be approached differently than the state space quantization because an element of the action space has interdependent components due to the constraint that all components must sum to one. This means that this space must be quantized with a vector quantizer. The process for the vectoral quantization was adapted from a paper on finite model approximations for partially observed Markov decision processes.[13] The algorithm quantizes the space by constructing a type lattice with the nearest neighbour method on the m$^{\text{th}}$ order simplex where m is the order of the space. From the paper, the quantized simplex Z_n is defined as shown below.

$$Z_n = \{(p_1, ..., p_m) \in \mathscr{Q}^m : p_i = \frac{k_i}{n}, \sum_{i=1}^{m} k_i = n\} \tag{32}$$

For the investment situation, m would be the number of stocks and bonds that are being considered, and n is the number of quantizations over the simplex that are desired. In order to find the best code point, k_i, corresponding to a given element of the action space, the algorithm below was used.

Step 1: For i = 1, ... , m and element z of the action space simplex, compute the following:

$$k'_i = \left\lfloor nz_i + \frac{1}{2} \right\rfloor$$

$$n' = \sum_{i=1}^{m} k'_i$$

Step 2: if n' = n, then the nearest element in the lattice is $(\frac{k'_1}{n}, ..., \frac{k'_m}{n})$

Step 3: otherwise find the errors of the values

$$\delta_i = k'_i - nz_i$$

Step 4: sort the above errors such that the jth smallest error is labelled

$$\delta_{ij}$$

Step 5: define Δ such that $\Delta = n' - n$

Step 6: if $\Delta > 0$, then using the new indices from the errors,

$$k_{ij} = \begin{cases} k'_{ij} & \text{if j = 1, ... , m - } \Delta \text{ - 1} \\ k'_{ij} - 1 & \text{if j = m - } \Delta \text{, ... , m} \end{cases}$$

Step 7: otherwise:

$$k_{ij} = \begin{cases} k'_{ij} + 1 & \text{if j = 1, ... , } |\Delta| \\ k'_{ij} & \text{if j = } |\Delta| \text{ + 1, ... , m} \end{cases}$$

Step 8: With these new k's defined, the set $(\frac{k_1}{n}, ..., \frac{k_m}{n})$ becomes the nearest element in the lattice given any element z in the simplex.

By choosing different n values, different orders of quantizations can be obtained and used to balance the design constraints of precision, computation time, and memory. The relation between the quantization step size and the amount of memory needed for the Q-table is summarized in the graph below. It clearly depicts an exponential increase of memory with the decrease of step size between quantizers.

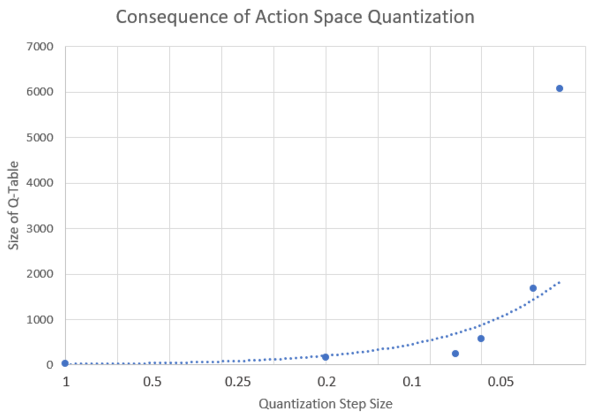

Figure 7: Effect of Action Space Quantization on the Size of the Q-table

7 Iterative Design with Real Data

To test the the performance design, the team observed the performance of Q-learning on real data while iteratively varying Markov degree, state space quantization and action space quantization. Real data was obtained for Microsoft, IBM and Qualcomm. For the initial investigation, the team observed the performance of Q Learning with the return rates treated as I.I.D. and the action space uniformly quantized with size 0.1. Below is the performance of this configuration on 1000 samples of testing data.

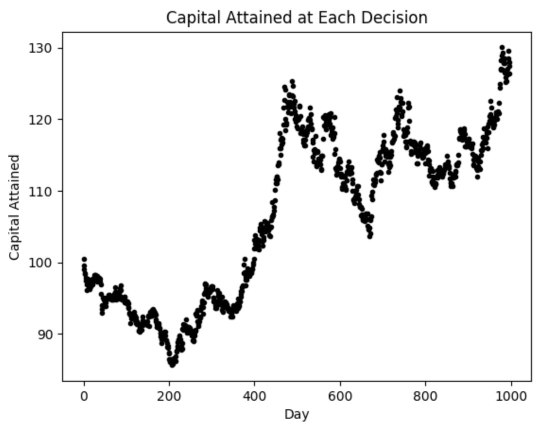

Figure 8: Training on real data treated as I.I.D. with action space quantizer length of 0.1

This design configuration yields a positive performance, however, the value of the portfolio is very volatile. This is due to the policy yielding the same action regardless of the previous states of the stocks as they are assumed to be I.I.D. in this experiment. Therefore, the value of the portfolio varies with the inherit randomness of the 3 stocks. Expanding on this idea, the team repeated the experiment with the return rates treated as Markov memory 1. By doing so, the Q-Learning agent approximates a stationary policy based off the previous state of all assests. Training on the real data and applying the policy obtained to the testing data yeilds the following performance.

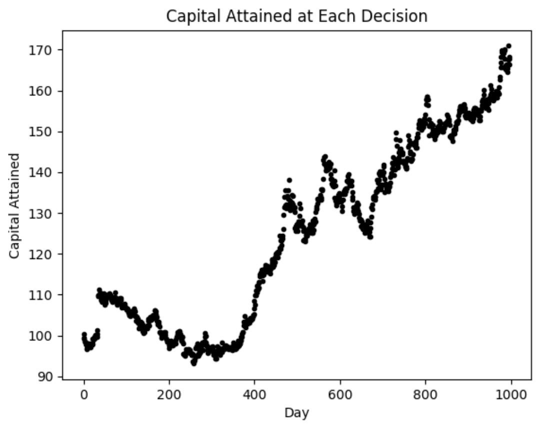

Figure 9: Training on real data treated as Markov 1 with action space quantizer length of 0.1

This configuration appears to have a more consistent performance over the testing period than the I.I.D. configuration. Furthermore, increasing the degree of memory once again yields the following performance. In this situation, the Q-Learning agent is approximating a stationary policy based off the previous two realizations of each of the 3 assets.

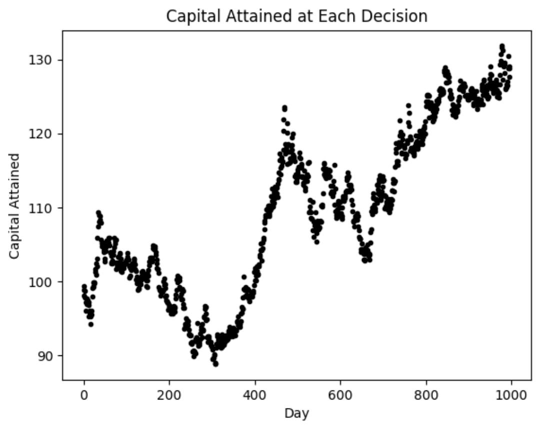

Figure 10: Training on real data treated as Markov 2 with action space quantizer length of 0.1

Here the performance appears to decrease in comparison to the Markov 1 configuration. This result is likely due to the configuration not meeting the requirement of entering each state infinitely often over the training period. In this experiment, the Q-table is much larger than previous experiments and the length of the data is kept the same. Such a design configuration requires a larger training data set or increased action space quantizer length to meet the criteria of Q-learning to arrive at a approximately optimal policy. As we do not have more data, consider the following performance where the action space quantizer length is increased.

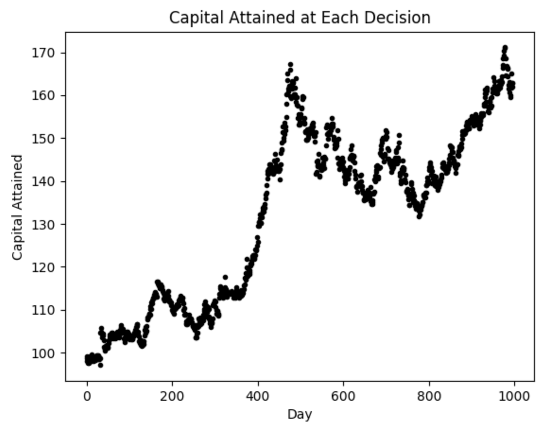

Figure 11: Training on real data treated Markov 2 with action space quantizer length of 0.2

This configuration yields a better performance than the previous experiment, where the action quantizer length was larger, however, the policy obtained does not perform better than the Markov 1 configuration.

The team arrived at the conclusion that the state space quantization, Markov order and action space quantization must be balanced to ensure each state action pair is visited infinitely often in the training period to ensure the Q-Learning agent can approach an approximate optimal policy.

In terms of evaluation session of this iterative design process, the team determined the empirical transition matrices and evaluated the above result by determining the optimal optimal using MatLab. The evaluation process is discussed in the following section.

8 Evaluation Process of the Iterative Design

The state space quantization, Markov order and action space quantization must be balanced to ensure the Q-Learning agent can approach an approximate optimal policy. When the team iteratively adjust the design in order to optimize the performance, we adopt the following evaluation process. The team determined the transition matrices of the stocks and obtained optimal policy from empirical data using MatLab. Then the MatLab result was compared to the Q-learning result in order to verify the design decisions made when performing the algorithm. This process allowed for the Q-Learning results to be debugged,and then analyzed through out the iterative process.

8.1 Determining Empirical Transition Matrices

While Q-Learning does not need the probability distribution of a source to function, to debug the algorithm and verify the results, MatLab could be used to analytically solve the optimization problem. Though to do this with the stock data that was obtained for the stocks that were used for testing, empirical transition matrices must be generated for the different memory lengths. For a set of data, T, the empirical distributions for the IID case were generated with the following algorithm:

$$P(i) = \left[\frac{\sum_{j=1}^{|T|} 1_{x_j=i}}{|T|}\right] \forall i \in \{1, ..., n\} \tag{33}$$

This was then adapted for Markov memory one with the below equation.

$$P(i,j) = \left[\frac{\sum_{k=1}^{|T|} 1_{x_k=i, x_{k-1}=j}}{\sum_{k=1}^{|T|} 1_{x_{k-1}=j}}\right] \forall i,j \in \{1, ..., n\} \tag{34}$$

The above result is then generalized for a Markov chain of order m below.

$$P(i_0, ..., i_m) = \left[\frac{\sum_{k=1}^{|T|} 1_{x_k=i_0,, ..., x_{k-n}=i_m}}{\sum_{k=1}^{|T|} 1_{x_{k-1}=x_{k-1}=i_1, ..., x_{k-n}=i_m}}\right] \forall \{i_0, ..., i_m\} \in \{1, ..., n\}^{m+1} \tag{35}$$

These results are an application of Baye's rule and provide the empirical distributions needed for the analysis.

8.2 Optimizing the Control Inputs with MatLab

The Q-Learning algorithm that was constructed outputs one action for each extended state that the process could be in, where the extended state is the set of states for the memory length of the particular source. To compare the MatLab optimized results to the Q-Learning, the transition matrices were decomposed into sets of I.I.D. sources, then optimized separately. By considering the transition matrix for a Markov Chain of order 1, it can be seen that the matrix can be deconstructed.

This means that for a first order Markov chain x_t, the probability can be written as:

$$P(x_t = b | x_{t-1} = a) = P(y_t = b) \tag{36}$$

where y_t is an IID source with its probability distribution given by the row corresponding to state 'a' of the transition matrix.

This means that depending on the number of states that the state space is quantized to, 3 in our case for now, the Markov memory one source can be split into that many IID sources as dictated by the rows of the transition matrix. Each IID source can then be processed by the MatLab code independently which will create investment proportions for each source. This can be extended to any memory of Markovian source. For a Markov chain x_t of order N with m states, it can be deconstructed as follows:

$$\begin{aligned}&P(x_t = b|x_{t-1} = a_1, ..., x_{t-N+1} = a_{N-1}, x_{t-N} = a) =\\&P(y_t^a = b|y_{t-1} = a_1, ..., y_{t-N+1} = a_{N-1})\end{aligned} \tag{37}$$

Where $a \in \{0, 1, ..., m-1\}$ and y_t^a is a Markov chain of order N - 1 with distribution equivalent to the m x m x ... x m matrix (N - 1 times) corresponding to the state 'a'. This process can of breaking up the Markov chain can be done recursively until only IID sources are left. In the end, for a Markov source of memory N, you would be left with m^{N-1} IID sources where m is the number of states.

After a set of IID sources has been obtained from the transition matrix, each source can be optimized to find the optimal action for each extended state. This was done with the use of the CVX package for MatLab which minimizes convex functions and maximized concave functions. The algorithm to determine the optimal actions for each extended state is presented below.

```
equation = 0
for all $r \in \mathbb{X}$
    $equation = equation + P(X = x)\log(u(1+r)')$
find vector u to have $\max_{u \in \mathbb{A}} equation$          (38)
where $\sum_{k=0}^{m} a_k = 1, 0 \leq a_k \leq 1, \forall a_k \in u$
```

Put all together, this process allowed for the Q-Learning results to be debugged, and then analyzed.

9 Design Improvements

9.1 Include Coefficients Optimization in the Iterative Design Process

One major improvement can be done in the future is to apply the Q-learning algorithm with more adaptive setup - allowing the the learning rate α and attenuation coefficient β change and adapt to different phases of the training process. Currently, the learning rate α and attenuation coefficient β are deterministic through out an training-testing iteration. (By "deterministic", the team meant that these two coefficient remain unchanged through out a single experiment.) According to Tsitsiklis's paper on the convergence of Q-learning [15], the algorithms can still converge to optimal policy even with α and β being random variables. Therefore, the team can theoretically change the value of these two coefficients during the training process if needed. For example, the team can gradually increase the learning rate *alpha* as the training process move forward, which allows the algorithm to weight the estimate of future value more in the later phase of the training. This is a valid action because the more training is done, the higher the accuracy of the future prediction will be. Similar argument applies to the attenuation rate β. Therefore, by making the learning rate and attenuation rate a dynamic variable, the team expects the Q-learning algorithm to have better performance.

9.2 Customize Memory Order for Different Stocks

nother design improvement would be to adapt the algorithm so that each stock could be a different order of memory length so that the design model for each stock would be more accurate. Currently, when all the stocks are tested simultaneously, an assumption was made such that they all have the same Markov memory order. The only customization for different stock was that they had different transition matrices. This might not fully reflect their nature. The team would like to implement a model such that different stocks can be considered as having different order of memory while being tested simultaneously, Tests with different sets of real data would also allow performance of the designed algorithm on an arbitrary predicted.

9.3 Train with More Data

The last design improvement would be performing more training and tests. Theoretically, all state-action pairs are required to be visited infinitely many times for Q-learning algorithm to converge to the optimal policy. Ideally more training data could be obtained so that the algorithm can more accurately train for Markov chains of higher order. However, one must consider the change in behavior of assets over time. If the team applied larger data sets, the older data may not appropriately represent the general behaviour today.

10 Environmental, Economic and Social Considerations

10.1 Economic Considerations

10.1.1 Maturity of Algorithmic Trading Technology

As with any stochastic process, there is always a chance for a negative outcome. In regard to algorithmic trading, this can come in the form of erroneous trades leading stock market crashes. The International Organization of Securities Commissions Technical committee found that the strong connections between financial markets where algorithmic trading and high frequency trading is used can lead to rapid shocks chaining from market to market [11]. This leads to an overall amplification of risk. One notable example of this was the market crash in May 2010, known as the Flash Crash. During the Flash Crash, a 5 to 6 percent oscillation of stock prices in major US equity in the span of only a few minutes occurred. The Dow Jones dropped approximately 1000 points that afternoon, a record drop at the time of occurrence [10]. The cause of the crash was from a single user placing large amounts of fake orders of an asset, who then cancelled them just before they were completed. This results in the trading algorithm receiving inputs of a certain market trend and making trades accordingly, while there was no market trend to begin with.

10.1.2 Volatility of the Stock Market

Other economic considerations of algorithmic trading include the intensifying volatility of the market. With many investors currently using algorithmic trading, a situation is created where each algorithm is trying to operate more efficiently than each other algorithm. To do this, the algorithms need to operate instantaneously. When markets are unsteady, this will result in algorithms diversifying assets to avoid being forced to take a position in which way to trade, which results in a decreased market liquidity and creates greater market volatility [17].

10.1.3 Reliability of the Algorithm

Another aspect to consider is the possibility of a faulty algorithm. As a result of the instantaneous operating speeds of algorithmic trading, an incorrect algorithm can lead to a cascade of faulty operations, and large sums of lost money. An example of this can be seen in faulty algorithm that Knight Capital used, which lead to a 440 million dollar loss in a 45 minute time window [7]. The error in the algorithm occurred as a result of the algorithm buying stocks at a greater price than needed, then selling them for less than the going value. Other algorithmic trading operations picked up on the mistake, which lead the company so close to bankruptcy that it got acquired by a larger corporation.

10.2 Environmental Considerations

10.2.1 Resource Consumption when Using Technology

Environmental impacts of algorithmic trading, or any other software application for that matter, are not always immediately obvious. However, these operations do not exist in a vacuum, and further context needs to be examined. Firstly, it has to be noted that software cannot exist without hardware. The hardware manufacturing industry uses natural resources such as energy, clean water, heavy metals, and rare earth metals, all of which have a large environmental impact[14]. The harm to the environment is more apparent when viewed in the context of a supply chain. Each component of the hardware must be manufactured, shipped, and assembled, just to operate the algorithmic trading program. Clearly, there is a correlation between the amount of algorithmic trading in the market and environmental impact; greater use of algorithmic trading requires greater hardware requirements. With an estimated 80 percent of stock market trades being algorithmic in nature, this is a significant impact [3].

10.2.2 Resource Consumption when Performing Complex Calculation

Another environmental aspect to be considered the computational power used by the algorithm. A more complex algorithm with result in more computing power used, and a larger environmental impact. While this may seem insignificant, many people using an inefficient algorithmic trading algorithm would lead to a large amount of unnecessary energy usage. Additionally, a less complex algorithm would lead to less strain on the hardware, and thus a longer lifespan of the hardware. In turn, this would lead to less discarded electronics, which is a significant environmental issue [1]. As a result, effort was taken to ensure our algorithm was developed with complexity in mind. This can be seen in our quantization design process.

10.3 Social Considerations

10.3.1 Reduce Risks of Investment

In our society today, investment is inevitable. If assets are not gaining value in some way, they will depreciate due to inflation. As such, having a safe and accessible way to invest would benefit society as a whole.

Currently, many individuals choose not to invest in the stock market. The volatile market and misunderstanding of the stock market are two key reasons for the lack of investment [4]. Using algorithmic trading, investors can get a sense that their money is being managed properly if they trust the algorithm. To ease concerns about investing in a volatile market, many younger individuals, and a growing number of older individuals, trust computers over humans to make the correct decisions [5].

Although there are many benefits to investing, mismanagement of funds can lead to devastating social outcomes. An example of this mismanagement of funds can be seen in the 2008 recession. Due to a lack of proper risk mitigation and inadvisable mortgage practices, the stock market crashed [2]. This lead to a job losses, foreclosures, and many other harmful social impacts to society.

10.3.2 Case Study of Benefits of Algorithmic Trading

There are many scenarios in which algorithmic trading can be used beneficially for society. One example would be the algorithmic trading used in the Canadian Pension Plan. Currently, the Canadian Pension Plan uses a combination of algorithmic trading and manual trading [6]. This combination of computer use with human oversight leads to optimal trading outcomes without the associated human errors such as letting emotions influence decision making. Additionally, the human oversight can assist in mitigating risks within the algorithm.

10.4 Regulatory Considerations

The Electronic Code of Federal Regulations, Section 240.15c3-5a (b) tell us that "A broker or dealer with market access, or that provides a customer or any other person with access to an exchange or alternative trading system through use of its market participant identifier or otherwise, shall establish, document, and maintain a system of risk management controls and supervisory procedures reasonably designed to manage the financial, regulatory, and other risks of this business activity" [12]. Additionally, from Section 240.15c3-5a (c)(1)(i), we must "Prevent the entry of orders that exceed appropriate pre-set credit or capital thresholds in the aggregate for each customer and the broker or dealer and, where appropriate, more finely-tuned by sector, security, or otherwise by rejecting orders if such orders would exceed the applicable credit or capital threshold".

Firstly, our risk management controls must be addressed. Our algorithm is easily halted, which could quickly stop any erroneous trades if they begin to occur. Secondly, we must address the issue of exceeding a pre-set capital threshold. Since only initial capital is invested, and the proportions of the partitions of our initial capital sum to one, it is impossible for our algorithm to exceed any capital threshold that is set.

References

[1] Syed Faraz Ahmed. *The Global Cost of Electronic Waste*. Sept. 2016. URL: https://www.theatlantic.com/technology/archive/2016/09/the-global-cost-of-electronic-waste/502019/?fbclid=IwAR1nzy739b13DlvsmX8MndlFSVw8dtPsg5t1SGRV1udby-EfzePD5LS21JI.

[2] Kimberly Amadeo. *The Great Recession of 2008: What Happened, and When?* Feb. 2019. URL: https://www.thebalance.com/the-great-recession-of-2008-explanation-with-dates-4056832.

[3] Charlie Bilello. *Algo Trading Dominates 80 Percent Of Stock Market*. Jan. 2019. URL: https://seekingalpha.com/article/4230982-algo-trading-dominates-80-percent-stock-market?fbclid=IwAR0AIRKEq%20%5Cnewline%20KBU2fNyrizSq9zxlHc0L3bnNw3o2JVigRYR_RN68k1cI1IpSFs.

[4] Shawn M. Carter. *Younger Americans aren't investing in the stock market-researchers think this is why*. May 2018. URL: https://www.cnbc.com/2018/05/16/gallup-why-younger-americans-arent-investing-in-the-stock-market.html?fbclid=IwAR0UlAqtBYU2-7ib60quBhgaPHI6y%20%5Cnewline%20jHpWzE0wrX4EeGe5tWcvQTpncklssQ.

[5] Jingjing Jiang. *Millennials stand out for their technology use*. May 2018. URL: https://www.pewresearch.org/fact-tank/2018/05/02/millennials-stand-out-for-their-technology-use-but-older-generations-also-embrace-digital-life/?fbclid=IwAR3maSNwRZlc%20%5Cnewline%20jY3htQXcc0XSK3AApJUUKzn7s4Zx0RHXRuFiBTkMjgLJCEo.

[6] Glenn Lowson. *Fully automated investments? We're not there yet*. Nov. 2017. URL: https://www.theglobeandmail.com/globe-investor/fully-automated-investments-were-not-there-yet/article37006622/.

[7] Matthew. *Knight Shows How to Lose 440 Million in 30 Minutes*. URL: https://www.bloomberg.com/news/articles/2012-08-02/knight-shows-how-to-lose-440-million-in-30-minutes?fbclid=IwAR2Z95u%20%5Cnewline%200p5HgIhHYAiZNRdv78ywRCjpl3s7HoeI86Pb4jKwtsGp9V0EW2sc.

[8] Michael J. McGowan. "The rise of computerized high frequency trading: use and controversy". In: *Duke Law & Technology Review* (2010).

[9] Kai Zimmermann Peter Gomber. "Algorithmic Trading in Practice". In: *The Oxford Handbook of Computational Economics and Finance* (2018).

[10] Elvis Picardo. *Four Big Risks of Algorithmic High-Frequency Trading*. Mar. 2019. URL: https://www.investopedia.com/articles/markets/012716/four-big-risks-algorithmic-highfrequency-trading.asp.

[11] *Regulatory Issues Raised by the Impact of Technological Changes on Market Integrity and Efficiency*. July 2011. URL: https://www.iosco.org/library/pubdocs/pdf/IOSCOPD354.pdf?fbclid=IwAR0KIMRDyf7HvoKQP-Hybkmj9FVUcrF4mBnHyJy0OdtRUPJTKNi8wzmGb4IA.

[12] *Risk management controls for brokers or dealers with market access.* Nov. 2010. URL: https://www.law.cornell.edu/cfr/text/17/240.15c3-5?fbclid=IwAR0PLaRJVHgX-s_AgUL_m_1_VbN_MqXF2DrMG4HFoaGADAp1A3J%20%5Cnewline%20W2BFPFDA.

[13] Naci Saldi, Serdar Yüksel, and Tamás Linder. "Finite Model Approximations for Partially Observed Markov Decision Processes with Discounted Cost". In: *CoRR* abs/1710.07009 (2017). arXiv: 1710.07009. URL: http://arxiv.org/abs/1710.07009.

[14] Subhas K. Sikdar. *Is the software industry environmentally benign?* Apr. 2015. URL: https://link.springer.com/article/10.1007/s10098-015-0953-6?fbclid=IwAR3okr0IpCLbYdZR7Ofylw021Upr6P8rVhF0w5Oy%20%5Cnewline%20GADIQTgzZNdxNOUvLi0.

[15] John N. Tsitsiklis and Richard Sutton. "Asynchronous stochastic approximation and Q-learning". In: *Machine Learning.* 1994, pp. 185–202.

[16] Christopher J. C. H. Watkins and Peter Dayan. "Q-learning". In: *Machine Learning.* 1992, pp. 279–292.

[17] Marvin Wee. *Market volatility is here to stay, but high-frequency trading not all bad.* Sept. 2018. URL: http://theconversation.com/market-volatility-is-here-to-stay-but-high-frequency-trading-not-all-bad-46615?fbclid=IwAR11NXneG79g7RgHtZVbHk_G7Tn6hPZqNi4Gd5%20%5Cnewline%201s7Y7IhpTFSXQDxK7KjFQ.