A DEEP REINFORCEMENT LEARNING APPROACH TO STOCK PORTFOLIO OPTIMIZATION

PAVANPREET SINGH GANDHI

**A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
BACHELOR OF SCIENCE (APPLIED MATHEMATICS)
MAHIDOL UNIVERSITY
INTERNATIONAL COLLEGE
2023**

Thesis
entitled

A DEEP REINFORCEMENT LEARNING APPROACH TO STOCK PORTFOLIO OPTIMIZATION

was submitted to the Science Division, Mahidol University
for the degree of Bachelor of Science (Applied Mathematics)
on
April 14, 2023

.....
Mr. Pavanpreet Singh Gandhi
Candidate

.....
Assoc. Prof. Dr. Chatchawan Panraksa,
Ph.D. (Mathematics)
Major advisor

.....
Dr. Sunsern Cheamanunkul,
Ph.D. (Computer Science)
Co-advisor

.....
Dr. Brian J. Phillips
Division Chair
Science Division
Mahidol University

.....
Assoc. Prof. Dr. Thotsaporn
Thanatipanonda,
Ph.D. (Mathematics)
Program Director
Bachelor of Science Programme
in Applied Mathematics
Mahidol University

ACKNOWLEDGEMENTS

I would like to express my sincerest gratitude to my professors, friends, family, and community for their continuous support throughout my undergraduate endeavor.

Pavanpreet Singh Gandhi

A DEEP REINFORCEMENT LEARNING APPROACH TO STOCK PORTFOLIO OPTIMIZATION

PAVANPREET SINGH GANDHI 6280559

B.Sc. (APPLIED MATHEMATICS)

THESIS ADVISORY COMMITTEE: CHATCHAWAN PANRAKSA, Ph.D.,
SUNSERN CHEAMANUNKUL, Ph.D.

ABSTRACT

This study investigates the use of deep reinforcement learning as an alternative to mean-variance analysis for stock portfolio optimization. The literature review introduces the necessary background of reinforcement learning and mean-variance portfolio optimization. The problem statement describes how portfolio optimization can be framed as a Markov decision process. The results indicate that deep Q-learning is comparable to mean-variance analysis, and can even potentially outperform it in certain situations.

KEY WORDS : REINFORCEMENT LEARNING / PORTFOLIO OPTIMIZATION / DEEP LEARNING / MEAN-VARIANCE ANALYSIS

24 pages

CONTENTS

CONTENTS (cont.)

LIST OF TABLES

LIST OF FIGURES

CHAPTER I
INTRODUCTION

Almost always, the quickest way for the average person to passively grow their wealth over time is through investing.

At the most fundamental level, successful investing in financial markets comes down to owning the right securities at the right time. Seasoned investors often develop intuition on how to time the market and pick the right securities. Some investors also use technical analysis combined with general knowledge of how the market moves to be able to do this.

With financial data becoming readily available, and artificial intelligence systems becoming increasingly popular, one is left to wonder whether it is possible to develop a system that mimics human intuition when it comes to investing. This *intuition* could be more concretely thought of as a mapping from market situations to portfolio allocations, both of which can be parameterized by high dimensional vectors.

This idea of learning a mapping between high-dimensional inputs and outputs reminds us of neural networks. However, traditional supervised learning is not a suitable training framework since there is no labeled data. As such, reinforcement learning provides a better framework since we can view portfolio optimization as a sequential decision-making problem.

This report is a preliminary investigation into reinforcement learning for portfolio optimization. We hope to determine whether or not reinforcement learning is a feasible alternative to traditional mean-variance portfolio optimization.

CHAPTER II
LITERATURE REVIEW

2.1 Reinforcement Learning

Reinforcement learning is a framework for training an agent to interact with a complex environment from experience. It is considered a branch of machine learning because the agent learns these control strategies from its own experiences by interacting with the environments with the environment. In this sense, reinforcement learning becomes a machine learning problem with the data being past experiences. Since the agent generates its own data, another name for reinforcement learning is self-supervised learning.

There is also evidence to suggest that human brains learn in a similar way to reinforcement learning. Just as neural networks mimic the brain in some ways, reinforcement learning mimics the trial-and-error learning process of biological creatures like animals and humans [1].

2.1.1 Markov Decision Processes

The goal of reinforcement learning is to train an agent to interact with an environment. This idea of an agent interacting with an environment can be mathematically formalized as a Markov decision process, where the agent exists in some state s_t within the environment and has the choice to take some action a_t resulting in a new state s_{t+1} and some reward r_t. One key detail is that this entire system must also exhibit the Markov property which means that the state-transition property $P_a(s_{t+1}, s_t)$ must be memoryless i.e independent of all previous states and actions.

When we allow the agent to interact with the environment in this way, we get a sequence of states, actions, and rewards that become the data that we use to learn an optimal control policy. Equation 2.1 shows an arbitrary example of one such sequence.

$$s_0 \rightarrow a_1 \rightarrow s_1 \rightarrow a_2 \rightarrow s_2 \rightarrow a_3 \rightarrow s_3 \rightarrow r_3 \rightarrow a_4 \rightarrow s_4 \ldots \tag{2.1}$$

One very important point to note is that equation 2.1 contains only one reward r_3. This is to reflect the fact that rewards can sometimes be sparse which makes it extremely difficult to determine which action sequence is responsible for producing rewards. This is known as the credit assignment problem and is one of the central challenges of reinforcement learning [2].

2.1.2 Policy Functions, Value Functions, and Quality Functions

In most reinforcement learning methods, the agent learns how to interact with the environment by learning functions that map states to actions or expected reward values. The following functions are the core of many reinforcement learning algorithms.

Policy functions represent any policy that we have learned to interact with the environment. They map state-action pairs to probabilities of taking that action in that state.

$$\pi(s, a) = \Pr(a = a \mid s = s) \tag{2.2}$$

Alternatively, a deterministic policy maps a single action to a given state.

$$\pi(s) = a \tag{2.3}$$

Value functions represent the value of a given state under a certain policy. This can be defined a little more concretely as the expectation of the discounted cumulative reward given that you start in state s and follow policy π, where γ is the discount factor (a number between 0 and 1).

$$V_\pi(s) = \mathbb{E}\left(\sum_t \gamma^t r_t \mid s_0 = s\right) \tag{2.4}$$

The intuition behind the discount factor is that future rewards are valued less than immediate rewards. It is also required for the convergence of some algorithms that will be mentioned in section 2.1.4 and 2.1.5.

Quality functions are just like value functions but for state-action pairs. They represent the value of taking a given action in a given state under a certain policy. Similarly, they can be defined as the expectation of the discounted cumulative reward given that you start in state s, take action a, and continue to follow policy π.

$$Q_\pi(s, a) = \mathbb{E}\left(\sum_t \gamma^t r_t \mid s_0 = s, a_0 = a\right) \tag{2.5}$$

In fact, a quality function contains all the information of a value function and a deterministic policy function as shown in equations 2.6 and 2.7.

$$V_\pi(s) = \max_a Q_\pi(s, a) \tag{2.6}$$

$$\pi(s) = \arg\max_a Q_\pi(s, a) \tag{2.7}$$

The goal of reinforcement learning is to find the policy π that maximizes the value and quality functions at all states or state-action pairs.

2.1.3 Taxonomy of Reinforcement Learning Algorithms

Reinforcement learning is an extremely large field that merges ideas from control theory, data science, optimization theory, dynamic programming, behavioral science, and even neuroscience [3]. There are many reinforcement learning algorithms that have been developed over the years, however, they can all be categorized into two broad buckets.

The first is model-based approaches, which essentially use a model to predict the next state s_{t+1} and reward r_t given some current state s_t and action a_t. Perhaps the most well-known and easiest to understand of all these approaches are value iteration [4] and policy iteration [5] where the dynamics of the Markov decision process are known.

Alternatively Model-free approaches don't use such models as an intermediate step and instead take samples from the distribution of all possible state-action-reward sequences as they iterate through the Markov decision process. Among the model-free methods, there is also a distinction between on-policy versus off-policy approaches. On-policy approaches optimize the same policy as the one that the agent is using to iterate through the environment, for example, Sarsa [6]. Off-policy approaches, on the other hand, optimize a different policy than the one that the agent is using to iterate through the environment, for example, Q-learning [7].

2.1.4 Model-Based Reinforcement Learning

Model-based reinforcement learning requires a model to predict the next state s_{t+1} and reward r_t given some current state s_t and action a_t. This subsection

explores two of the simplest model-based approaches, namely value iteration and policy iteration. These form the cornerstone upon which the field of reinforcement learning was built.

Both policy iteration and value iteration operate under the assumption that the agent has full knowledge of the Markov decision process that represents the environment i.e. equation 2.8 and 2.9 are known functions.

$$P(s' \mid s, a) = \Pr(s_{t+1} = s' \mid s_t = s, a_t = a) \tag{2.8}$$

$$R(s', s, a) = \Pr(r_{t+1} \mid s_{t+1} = s', s_t = s, a_t = a) \tag{2.9}$$

Given this information, Richard Bellman devised the value iteration algorithm [4] which leverages dynamic programming to solve for the optimal value function. If we define the optimal value function as

$$V_\star(s) = \max_\pi V_\pi(s) = \max_\pi \mathbb{E}\left(\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s\right) \tag{2.10}$$

then Bellman showed that if we follow the optimal policy and get a new state s' and some reward r, the optimal value function can be written recursively as

$$V_\star(s) = \max_\pi \mathbb{E}\left(r + \sum_{t=1}^{\infty} \gamma^k r_t \mid s_1 = s'\right) = \max_\pi \mathbb{E}\left(r + \gamma V_\star(s')\right) \tag{2.11}$$

Equation 2.11 is known as the Bellman optimality equation. If we find a function that satisfies this equation, then Bellman showed that it must be an optimal value function. Finding this optimal value function can be done by breaking down the problem into sub-problems using dynamic programming. This can be done by simply changing the equality to an assignment as in equation 2.12. Bellman showed that repeatedly sampling from the environment and enforcing this recurrence relation will result in convergence to the optimal value function.

$$V(s) \leftarrow \max_a \sum_{s'} P(s' \mid s, a)\left(R(s', s, a) + \gamma V(s')\right) \tag{2.12}$$

Policy iteration [5] takes this a step further by decoupling the evaluation (equation 2.13) and the improvement (equation 2.14) of the policy which can often result in faster convergence to the optimal value function.

$$V_\pi(s) \leftarrow \max_a \sum_{s'} P(s' \mid s, a)\left(R(s', s, a) + \gamma V(s')\right) \tag{2.13}$$

$$\pi(s) \leftarrow \arg\max_{a} \sum_{s'} P(s' \mid s, a) \left(R(s', s, a) + \gamma V(s')\right) \tag{2.14}$$

2.1.5 Model-Free Reinforcement Learning

Value iteration was the first-ever reinforcement learning algorithm, however, its applications are greatly limited since knowing the dynamics of the Markov decision process is not always feasible. Model-free methods provide a framework for an agent to learn from its own experience without the need to model the reward or next state. This section explores Q-learning [7], the first ever model-free reinforcement learning algorithm.

As the name suggests, Q-learning works by learning the quality function instead of the value function. In a similar manner to equation 2.11, the optimal quality function can be written recursively as

$$Q_\star(s, a) = \max_{\pi} \mathbb{E}\left(r + \gamma \max_{a'} Q_\star(s', a')\right). \tag{2.15}$$

Since the state transition probabilities (equation 2.8) are not known, the expression inside the expectation must be estimated based on samples from the environment. Each sample is a tuple of length 4 containing the current state s, action taken a, reward received r, and next state s'. This estimate is called the temporal difference target estimate R_Σ.

$$R_\Sigma = r + \gamma \max_{a'} Q(s', a') \tag{2.16}$$

The optimal quality function can now be computed in a similar manner to equation 2.12, but with a learning rate (α) to smooth out the randomness of the temporal difference target estimate

$$Q(s, a) \leftarrow Q(s, a) + \alpha(R_\Sigma - Q(s, a)). \tag{2.17}$$

In the original Q-learning paper [7], Watkins et al. showed that repeatedly iterating through the environment and performing this update on the Q-function resulted in guaranteed convergence to the optimal Q-function.

When computing R_Σ according to equation 2.16, the original Q-learning algorithm [7] adopts an exploration strategy where a random action is selected with

a probability ϵ. This allows the algorithm to explore alternative strategies to avoid it getting stuck into local maxima. Since exploration results in the agent taking actions other than its best-known policy, Q-learning is an off-policy algorithm. An alternative approach is to select action a as the action that maximizes the quality function, thereby never going off-policy. This approach is known as SARSA [6] which is classified as an on-policy algorithm. In practice, both methods have their own use cases with SARSA often being used in situations that need to be conservative with exploration - for example robotics, and Q-learning being used where exploration is easily doable - for example in simulated environments.

An extension of Q-learning is double Q-learning [8], which uses two quality functions for increased stability during the training process. The algorithm interchangeably updates one quality function while using the other to compute the temporal difference target estimate. Other model-free algorithms include policy gradients [9] and actor-critic methods [10] - both of which are incredibly useful but are beyond the scope of this report.

2.1.6 Deep Reinforcement Learning

Deep reinforcement learning is one of the most up-and-coming branches of machine learning and control theory as it has achieved remarkable results over the past decade. The main idea of Deep reinforcement learning is to use deep neural networks as functional approximators for policy, value, and/or quality functions.

This works so well because neural networks are great at dealing with high dimensional state spaces without having to enumerate the policy/value/quality function for every state/state-action pair. It even naturally extends to continuous state spaces. Broadly speaking, deep reinforcement learning can be classified into three main categories [2]:

1. Deep Q-learning methods
2. Deep policy gradient methods
3. Deep actor-critic methods

This section explores deep Q-learning, an algorithm pioneered by DeepMind in 2013

when they demonstrated how it could learn to play Atari video games from sensory inputs [11].

Deep Q-learning methods train a neural network as a functional approximator for the quality function. This neural network is called the Q-network.

$$Q(s, a) \approx Q(s, a; \theta) \tag{2.18}$$

where θ is a vector representing the parameters of the Q-network.

These parameters are trained just like any other neural network using back-propagation. The loss function represents the difference between the actual Q-values and the temporal difference target estimate of the Q-values (equation 2.16). An example with mean squared error loss is shown in equation 2.19.

$$\mathcal{L} = \Big[Q(s, a; \theta) - (r + \gamma \max_{a'} Q(s', a'; \theta)\Big]^2 \tag{2.19}$$

In practice, since deep Q-learning is off-policy, the training data set can be created on the go by playing the game and collecting experience tuples of the form (s, a, r, s'). We then randomly sample mini-batches of experience tuples for training and perform back-propagation on entire batches at a time. Simultaneously, we add new experience tuples based on our current Q-network and exploration strategy. This mechanism is called experience replay.

There is a lot of flexibility when choosing the architecture of the Q-network. The input of the Q-network is a state vector, and the outputs are the Q-values for each action. However, within the network, there can be any combination of network architectures including fully connected layers, batch normalization, convolutional layers, recurrent blocks, attention mechanisms, transformer architectures, etc.

2.2 Portfolio Optimization

Now we switch our attention to portfolio optimization. Simply put, a portfolio is a basket of tradeable securities such as stocks, bonds, commodities, cryptocurrencies, etc. Investors hold portfolios with the hope that they will grow in value over time. A common piece of advice when crafting a portfolio is to diversify into multiple

markets so as to not "keep all your eggs in the same basket". Portfolio optimization refers to the problem of weighting the securities in a portfolio to achieve the greatest increase in value.

2.2.1 Fundamentals of Risk and Return

Suppose an investor purchases a security at price p_0 and time t_0 and sells it at time t_1 for a price p_1. The time period between t_0 and t_1 is called the holding period, suppose this is one day. The daily rate of return denoted by r_t is defined as the percentage change in price, as shown in equation 2.20.

$$r_t = \left(\frac{p_1 - p_0}{p_0} \right) \tag{2.20}$$

When investors purchase a security for a period of time, they obtain a finite sequence of daily returns, as shown in equation 2.21.

$$\{r_1, r_2, r_3, \ldots, r_n\} \tag{2.21}$$

Equation 2.22 shows how to compute the annualized mean historical return of a security given a sequence of daily returns.

$$\mu = \frac{252}{n} \sum_{i=1}^{n} r_i \tag{2.22}$$

where we assume are 252 trading days in a year.

The annualized historical risk of the security is quantified by the standard deviation of the sequence of returns i.e how much these returns deviate from the mean. The formula to compute this is shown in equation 2.23.

$$\sigma = \sqrt{\frac{252}{n} \sum_{i=1}^{n} (\mu - r_i)^2} \tag{2.23}$$

2.2.2 Portfolio Risk and Return

Suppose an investor holds a portfolio of k securities for n days. Each security will have its own sequence of daily returns. The portfolio can be characterized by a sequence of weights as shown in equation 2.24.

$$\{w_1, w_2, \ldots, w_k\} \tag{2.24}$$

The annualized historical return of a portfolio can be computed by taking a weighted sum of the annualized historical return of each security, as shown in equation 2.25.

$$\mu_{\text{port}} = \sum_{i=1}^{k} w_i \cdot \mu_i \tag{2.25}$$

The annualized historical risk of a portfolio can be computed by taking the square root of a weighted sum of the covariance matrix of the returns, as shown in equation 2.26

$$\sigma_{\text{port}} = \sqrt{\sum_{i=1}^{k} \sum_{j=1}^{k} w_i \cdot w_j \cdot \sigma_{ij}} \tag{2.26}$$

where σ_{ij} is the covariance between the returns of security i and security j.

2.2.3 Mean-Variance Optimization

In 1952, Harry Markowitz proposed a systematic approach to constructing portfolios using mean-variance analysis [12]. The proposed method involves solving a nonlinear bi-objective optimization problem (equation 2.27) that maximizes portfolio return while minimizing portfolio risk, with the constraints being that all the weights must sum to 1 and be between 0 and 1.

$$\begin{aligned} &\max \sum_{i=1}^{k} w_i \cdot \mu_i \\ &\min \sqrt{\sum_{i=1}^{k} \sum_{j=1}^{k} w_i \cdot w_j \cdot \sigma_{ij}} \\ &\text{subject to} \\ &\sum_{i=1}^{k} w_i = 1 \\ &0 \leq w_1 \leq 1 \end{aligned} \tag{2.27}$$

Solving this bi-objective optimization results in an infinite number of weight allocations that provide the highest returns given an expected level of risk (or the lowest risk given an expected return). Graphing these portfolios with portfolio return on the *x-axis* and portfolio risk on the *y-axis* leads to a curve known as the efficient frontier. Any portfolio that lies on this efficient frontier is optimal in the sense that it achieves the highest return for a given level of risk.

The Sharpe ratio is a metric introduced by William F Sharpe to evaluate the performance of mutual funds according to their expected risk and return [13]. We can optimize for the Sharpe ratio as shown in equation 2.28 [14], where R_f is the risk-free rate.

$$\begin{aligned} &\max \left(\frac{\sum_{i=1}^{k} w_i \cdot \mu_i - R_f}{\sqrt{\sum_{i=1}^{k} \sum_{j=1}^{k} w_i \cdot w_j \cdot \sigma_{ij}}} \right) \\ &\text{subject to} \\ &\sum_{i=1}^{k} w_i = 1 \\ &0 \leq w_1 \leq 1 \end{aligned} \tag{2.28}$$

Another way to think about the Sharpe ratio optimization problem is that it maximizes the ratio of the risk premium (the return in excess of the risk-free rate) over the risk level. This method provides an effective way to construct an optimal portfolio with the highest reward-to-risk ratio. It also turns out that the solution to this optimization problem lies on the efficient frontier, tangent to the capital allocation line [14].

CHAPTER III
PROBLEM STATEMENT

Having understood the basics of reinforcement learning and portfolio optimization, we now move on to how to apply reinforcement learning to learn a portfolio optimization strategy.

Reinforcement learning deals with finding optimal strategies to navigate complicated environments. However, it can also be viewed as a framework for learning optimal decision-making. The problem of portfolio optimization can be viewed as a Markov decision process; with the states being the current state of the market, the actions being how to change the portfolio weights, and the reward being some performance metric of the portfolio.

This chapter outlines details regarding the environment for portfolio optimization.

3.1 Data

Ten stocks were arbitrarily selected from the Dow Jones Industrial Average; namely, American Express Company, Apple Inc, Boeing Co, Goldman Sachs Group Inc, Intel Corporation, Johnson & Johnson, Coca-Cola Co, Nike Inc, Procter & Gamble Co, and Walt Disney Co. The Dow Jones Industrial Average was chosen since it contains the 30 biggest US companies, hence their data is readily available and their stock price is not too volatile. Additionally, the risk-free rate was estimated as the 10-year US treasury yield. For each stock, its daily adjusted closing price and daily trading volume were downloaded from *Yahoo Finance* using the *yfinance* Python package.

To make the data easier to work with, the daily returns were computed along with the daily percentage change in trading volume. This makes the data stationary and

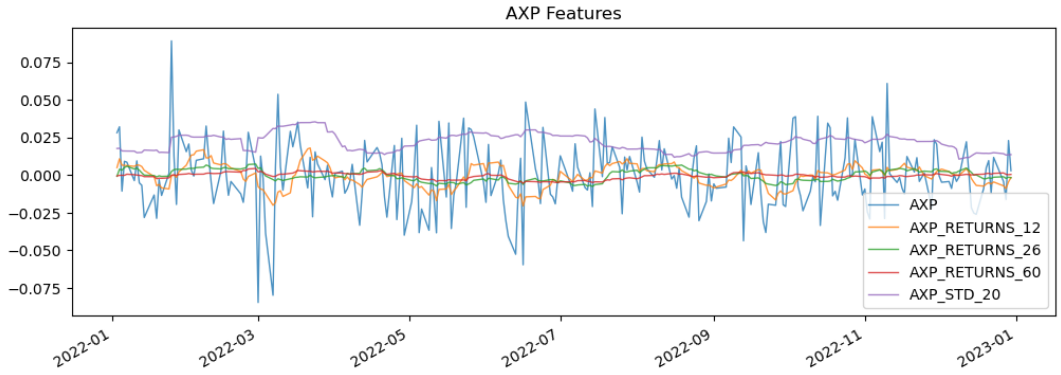

Figure 3.1: Visualization of features for one stock

Dataset	**Start Date**	**End Date**
Training	1 January 2010	31 December 2019
Validation	1 January 2020	31 December 2021
Testing	1 January 2022	31 December 2022

Table 3.1: Data splitting

numerically stable. Occasionally, missing datapoints were imputed to be 0 since this implies no change in the underlying security.

Inspired by conventional trading indicators such as Moving Average Convergence Divergence, Relative Strength Index, and Bollinger Bands - features were created out of the sequence of returns. In particular, rolling returns of window sizes 12, 26, and 60 days were computed along with the rolling standard deviation of returns with a window of 20. For example, the rolling returns with a window size of 12 would be a rolling geometric mean of the previous 12 daily returns.

The final feature set included the following 6 features for each stock; returns, 12-day rolling returns, 26-day rolling returns, 60-day rolling returns, 20-day rolling standard deviation, and percent change in trading volume. Figure 3.1 shows a visualization of these features for the American Express stock price in 2022. The hope is that these features are able to effectively summarise the market trends up to that day. A different feature set may be better or worse depending on how well it captures the market trends, however, this feature set proved effective for our purposes.

The data was downloaded for the time period between 1st January 2010 and 31st December 2022, and split as described in table 3.1.

3.2 States

The state of the environment is the state of the market. The observation is the information about the state that is visible to the agent. For the agent to be able to learn a portfolio allocation strategy, each observation must contain enough information about the market and its own position within it. This can be achieved by providing it with a history of the returns, moving averages of the returns, standard deviations of the returns, and percentage changes of the trading volumes for each stock. A history of the risk-free rate of return is also included. The size of this history is given by a *window size* parameter that is set to 10. To provide the agent with information about its current position in the market, the current portfolio weights are also included in the observation.

Overall, given k stocks, x features per stock, and an n day window size; the size of the observation space can i sgiven by equation 3.1.

$$o(k, n, x) = (k \cdot n \cdot x) + (k + 1) + n \tag{3.1}$$

Given $k = 10$, $n = 10$, and $x = 6$; each observation is a vector of length 621.

3.3 Actions

When viewing the portfolio optimization problem as a Markov decision process, the actions represent changing the portfolio weights. That being said, to allow us to use *deep Q-learning*, the action space must be discrete. To make this action space discrete, we allow for 2 actions per stock, buy or sell, along with a hold action which makes no change to the portfolio weights. The total sum of the weights must always be equal to one, so buying or selling a stock will mean selling or buying an equivalent amount of cash. The amount that is bought or sold depends on the current portfolio value, however, the weights can change by increments represented by an *order size* parameter which is set to 0.1. If an action is invalid, for example, if we do not own enough cash to buy more of a stock, then the action is equivalent to the hold action.

Overall, given k stocks; the size of the action space is given by equation 3.2.

$$a(k) = 2k + 1 \tag{3.2}$$

Given $k = 10$, the action space has a size of 21.

3.4 Reward Structure

Having a good reward structure is critical to ensure the agent is able to learn a good strategy. This is the case because the reward is the driving force behind the agent's selection of actions. The main goal is to grow the portfolio value which corresponds to higher portfolio returns, so naively we could define the reward as the rate of return of the portfolio. However, to encourage long-term strategies over short-term gain, the cumulative return of the portfolio is considered instead. However, one problem arises with this approach. In the initial time-steps, the actions taken have a much greater effect on the cumulative return than in the later time-steps. To mitigate this, one approach is to scale each reward by the number of time-steps that have passed relative to the episode length. This way a high cumulative reward at a later time-step gives a much higher reward than the same cumulative reward at an earlier time-step, allowing *good* actions at later time-steps to be just as valuable as those from earlier time-steps.

Given a sequence of portfolio returns $r_1, r_2, \ldots, r_n$ and a maximum episode length of N, the reward at time-step $n \leq N$ is given by equation 3.3.

$$\text{reward} = \left(\prod_{i=1}^{n} (1 + r_i) - 1 \right) \cdot \frac{n}{N} \tag{3.3}$$

However this reward structure is not without its limitations. One major drawback is that there is no penalty for a high risk portfolio, which is something that many investors would want to discourage. A possible solution would be to introduce a *drawdown penalty* within the reward structure, however, this is outside the scope of this report.

CHAPTER IV
RESULTS

4.1 Model

To train an agent to maximize rewards in the portfolio optimization environment, the deep quality network (*deep Q-learning*) algorithm was used, as implemented in the *Stable-Baselines3* Python package. This algorithm was chosen because of its relative simplicity and intuitiveness compared to some of the other algorthms that build on it. Most of the parameters were left as default with the exception of the *learning rate* being set to 0.0003 and the *batch size* being set to 64, since this is what seemed to allow it to train faster on a typical gaming laptop. Note that the quality network architecture was also left as the default multilayer perceptron with 2 hidden layers, each with 64 nodes. The model was trained on the training data over $3,000,000$ time-steps.

4.2 Baselines

We evaluate the model performance against two baselines. The first baseline is the Dow Jones Industrial Average, which represents the market. The second baseline is the maximum Sharpe ratio portfolio re-balanced daily. To compute this baseline, we use the historical annualized return and historical annualized risk to compute the maximum Sharpe ratio portfolio. This portfolio is computed daily according to a 126-day window size (half a trading year), and the weights are adjusted accordingly. The overall result is a portfolio whose weights change daily based on the Sharpe ratio and mean-variance analysis. A visualization of the maximum Sharpe ratio portfolio is shown in figure 4.1.

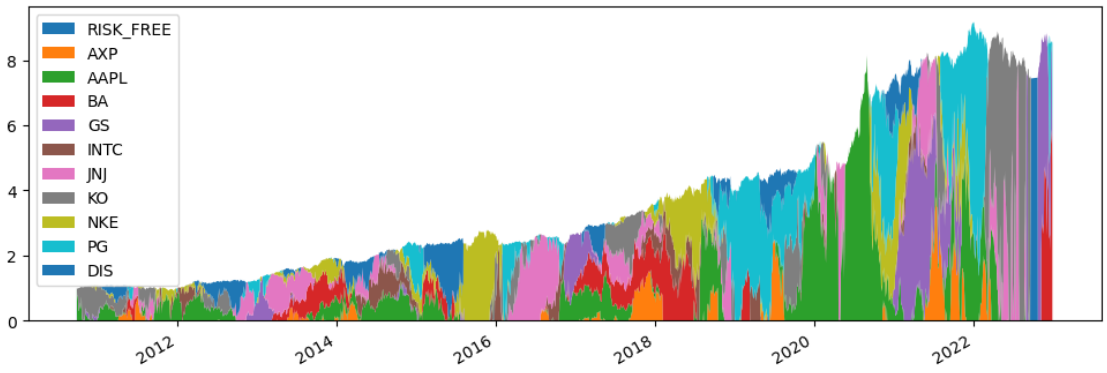

Figure 4.1: Portfolio weights of maximum Sharpe ratio portfolio over time

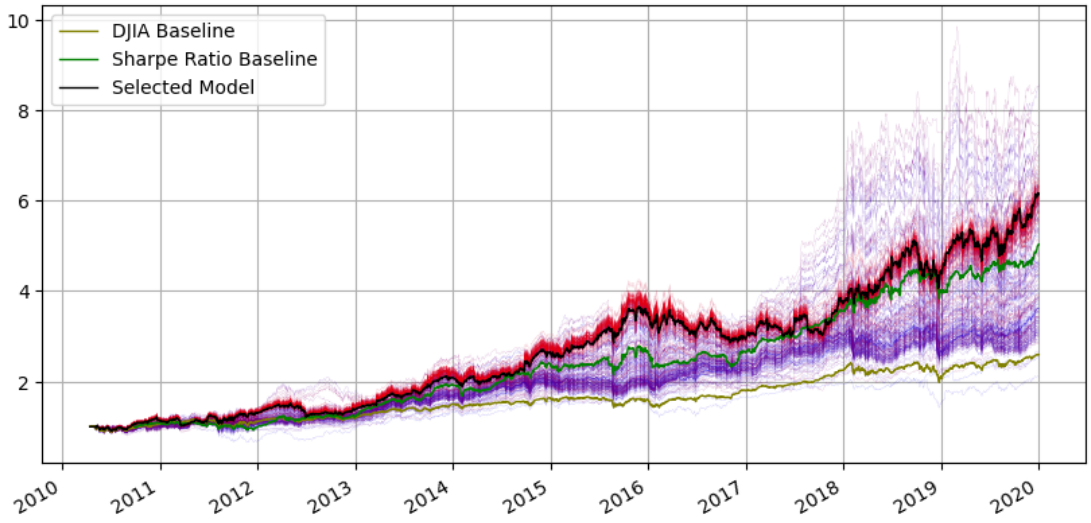

Figure 4.2: Model performance on training data

4.3 Performance

By training the model over $3,000,000$ time-steps and saving the model every $10,000$ time-steps, we can evaluate the model as the training process goes on to see how it improves. Figures 4.2, 4.3, and 4.4 graph the value of the portfolio over time at every $10,000$ iterations for the training data, validation data, and testing data respectively. The lines gradually change from blue to red as the number of time-steps increases from $10,000$ to $3,000,000$. Additionally, the black line represents the portfolio value of the model at a time-step that is late during the training process (specifically at $2,900,000$ time-steps). The olive and green lines represent the DJIA and maximum Sharpe ratio baselines respectively. Tables 4.1, 4.2, and 4.3 quantitatively compare the model at

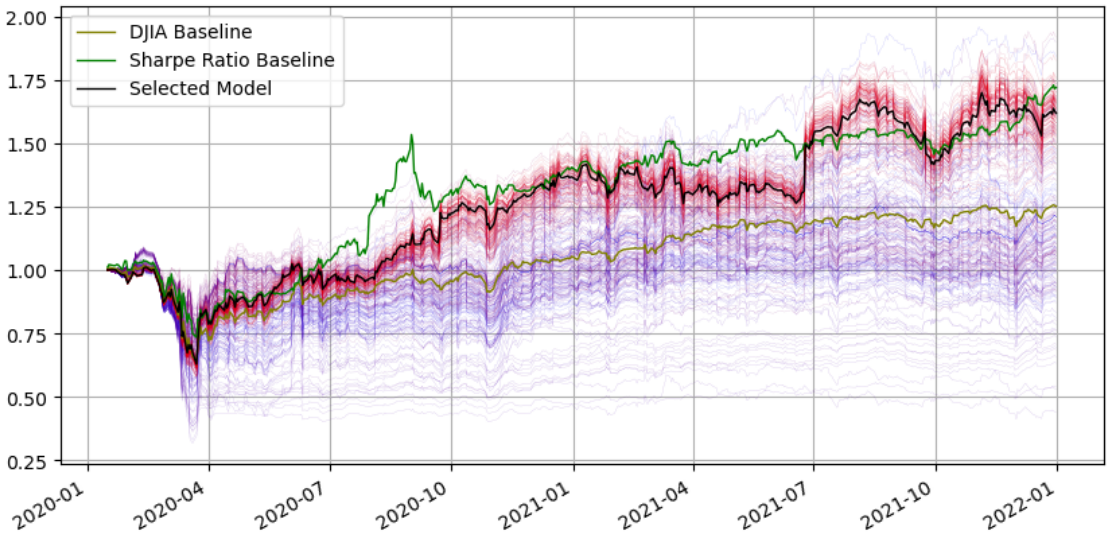

Figure 4.3: Model performance on validation data

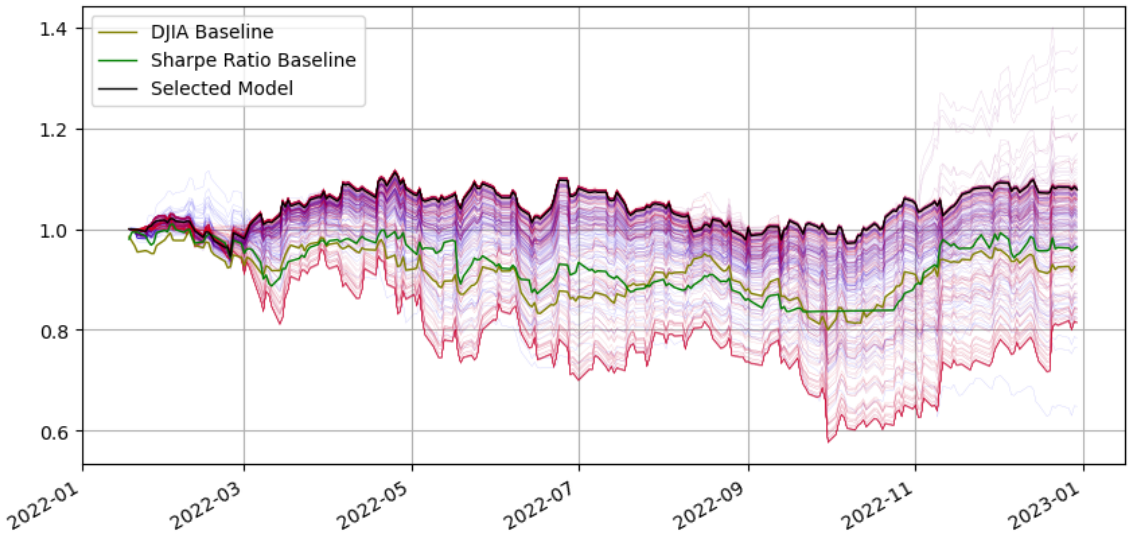

Figure 4.4: Model performance on test data

	Annualized Mean Rate of Return	Annualized Risk	Sharpe Ratio
DJIA	0.103	0.141	0.730
Max Sharpe	0.191	0.170	1.129
Deep Q-Network	0.206	0.216	0.951

Table 4.1: Performance metrics on training data

	Annualized Mean Rate of Return	Annualized Risk	Sharpe Ratio
DJIA	0.121	0.277	0.438
Max Sharpe	0.318	0.319	0.996
Deep Q-Network	0.278	0.333	0.835

Table 4.2: Performance metrics on validation data

$2,900,000$ time-steps against the two baselines over the training data, validation data, and test data respectively.

4.4 Analysis

From the results, it is clear that reinforcement learning is a feasible approach to portfolio optimization. This can be seen as the Deep Q-Network model is comparable to the maximum Sharpe ratio portfolio on the training and validation data. In fact, it performs even better than the maximum Sharpe ratio portfolio on the testing data.

The performance on the training data compared to the validation data does indicate some potential overfitting. However, the starting timestep was randomized during the training process to mitigate this. Additionally, the validation data contains the unprecedented market crash that resulted from COVID-19, which could also explain the lower performance on the validation data. Either way, the model is still comparable to the maximum Sharpe ratio portfolio.

	Annualized Mean Rate of Return	Annualized Risk	Sharpe Ratio
DJIA	-0.079	0.202	-0.389
Max Sharpe	-0.037	0.196	-0.189
Deep Q-Network	0.082	0.172	0.474

Table 4.3: Performance metrics on test data

From the results it can also be seen that the Sharpe ratio of the maximum Sharpe ratio portfolio is higher than the Deep Q-Learning portfolio. One reason that could explain this difference is that the reward function only rewards high returns and does not penalize investment risk e.g. variance or drawdown.

Another interesting observation is that the deep Q-network performs significantly better on the test data than both the maximum Sharpe ratio portfolio and the DJIA. This could be because the test data represents a period of economic recession, and the Q-network was able to react quicker than the Sharpe ratio portfolio by investing higher percentages into cash.

Overall, the results indicate that reinforcement learning approaches are feasible alternatives to traditional mean-variance portfolio optimization.

CHAPTER V
CONCLUSION

In conclusion, deep reinforcement learning is definitely a viable alternative to traditional mean-variance optimization. Practical applications of such a model can range from personal finance investment recommendations to fully automated investment funds based on reinforcement learning.

However, certain limitations of this study must be addressed before applications in the real-world can be considered. First of all, risk penalties must be introduced into the reward function so that a user can input their own risk tolerance. Other potential improvements include verifying with different underlying securities (e.g. small cap stocks, crypto), training different models (e.g. Actor-Critic, DDPG), optimizing hyperparameters, trying different network architectures (e.g. recurrent networks, transformers), and factoring more realistic conditions (e.g. trading costs).

REFERENCES

[1] Daeyeol Lee, Hyojung Seo, and Min Whan Jung. Neural basis of reinforcement learning and decision making. *Annual review of neuroscience*, 35:287–308, 2012.

[2] Steven L Brunton and J Nathan Kutz. *Data-driven science and engineering: Machine learning, dynamical systems, and control*. Cambridge University Press, 2022.

[3] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[4] Richard Bellman. A markovian decision process. *Journal of mathematics and mechanics*, pages 679–684, 1957.

[5] Ronald A Howard. Dynamic programming and markov processes. 1960.

[6] Gavin A Rummery and Mahesan Niranjan. *On-line Q-learning using connectionist systems*, volume 37. University of Cambridge, Department of Engineering Cambridge, UK, 1994.

[7] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8: 279–292, 1992.

[8] Hado Hasselt. Double q-learning. *Advances in neural information processing systems*, 23, 2010.

[9] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999.

[10] Vijay Konda and John Tsitsiklis. Actor-critic algorithms. *Advances in neural information processing systems*, 12, 1999.

[11] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[12] H Markowitz. Portfolio selection the journal of finance, vol. 7, no. 1, 1952.

[13] William F Sharpe. The sharpe ratio. *Streetwise–the Best of the Journal of Portfolio Management*, 3:169–185, 1998.

[14] GA Vijayalakshmi Pai. *Metaheuristics for portfolio optimization: an introduction using MATLAB*. John Wiley & Sons, 2017.

BIOGRAPHY

NAME	Mr. Pavanpreet Singh Gandhi
DATE OF BIRTH	28 April 2001
PLACE OF BIRTH	London, UK
INSTITUTIONS ATTENDED	NIST International School, 2017–2019
	Highschool Diploma (International Baccalaureate)
	Mahidol University, 2019–2023
	Bachelor of Science (Applied Mathematics)
	Mahidol University International College
E-MAIL	pavanpreet.gandhi@gmail.com