A comparison of evolutionary computation and deep reinforcement learning for portfolio optimization

K Pooe

orcid.org/0000-0002-7169-9248

Dissertation accepted in fulfilment of the requirements for the degree Master of Engineering in Computer and Electronic Engineering at the North-West University

Supervisor:	Prof P van Vuuren
Co-supervisor:	Prof Alwyn Hoffman

Graduation:	June 2021
Student number:	29661609

Abstract

In portfolio management, asset allocation is one of the most crucial and difficult challenges investors face. Asset allocation is defined as a decision making process of spreading available funds into various financial assets. The most famous and widely used models for tackling asset allocation problems are mean variance, mean value-at-risk and Sharpe ratio. These models are solvable by quadratic programming, and they all rely heavily on the mean and standard deviation with the assumption that the data distribution is symmetrical. Unfortunately, a majority of the real-world problems exhibit asymmetric distributions; as a result, the modified Sharpe ratio is introduced to include skewness and kurtosis as the third and fourth moments of return.

The results obtained in this study are based on the modified Sharpe ratio, and they apply and compare genetic algorithm, particle swarm optimisation, and deep deterministic policy gradient to solve the asset allocation problem. The former algorithms (genetic algorithms and particle swarm optimisation) are widely employed to generate high quality solutions in optimisation problems whilst the latter (deep deterministic policy gradient) has proved to be more effective in solving complex problems that cannot be solved by conventional techniques. The algorithms learn to evolve portfolio weights in maximising the modified Sharpe ratio. The dataset used is extracted from the banking sector of the Johannesburg stock exchange and well-known stocks in the United States stock exchange.

In measuring the performance of the three algorithms, a uniform allocation is used as a baseline asset allocation strategy. Uniform allocation divides portfolio weights equally among the assets in a portfolio. The results presented show that all three algorithms outclass the uniform allocation on numerous occasions. In general, the genetic algorithm and particle swarm optimisation provide relatively better results than deep deterministic policy gradient. The results are then tested on buy-and-hold. Even though the deep deterministic policy gradient did not perform well in evolving portfolio weights and took too long to run in training, it is comparable

with the uniform allocation. The genetic algorithm outperforms the other algorithms with particle swarm optimisation following.

Keywords

Asset allocation, Modified Sharpe ratio, Deep deterministic policy gradient, Genetic algorithms, Particles swarm optimisation, Buy-and-hold strategy, Uniform allocation, Skewness, Kurtosis

Acknowledgments

I would like to thank my supervisor and my co-supervisor, Prof Pieter Van Vuuren and Prof Alwyn Hoffman, for their support, patience and guidance throughout the research period. My lovely wife, Marelebogile Pooe for her unwavering support, encouragement and proofreading. I would also like to gratefully acknowledge the generous financial assistance from my employer Metropolitan Lesotho. Lastly, I would like to extend my thanks to Dr Senekane Makhamisa for his support in the suggestion of the research field.

This dissertation is dedicated to the memory of my mother, Manyakallo Anna Pooe, who always believed in my ability to succeed in the academic arena.

Contents

Listing of figures

Listing of figures - Continued

Listing of tables

List of common abbreviations

DDPG Deep deterministic policy gradient

DP Dynamic programming

DRL Deep Reinforcement Learning

EA Evolutionary Algorithm

GA Genetic Algorithm

JSE Johannesburg Stock Exchange

MDP Markov decision process

PSO Particle Swarm Optimisation

VaR Value-at-Risk

1

Introduction

This introductory chapter briefly correlates the research problems to the relevant areas of research in a non-technical manner. Furthermore, the scope and objectives of this dissertation are explained without digging deep into details. An overview of the contents of the dissertation is provided at the end.

1.1 Background

Portfolio construction is a classic problem faced by investors in investment. Generally, portfolio construction has two main components that directly influence investment results: security

selection and asset allocation. Security selection is defined as a process of determining which financial securities are included in a portfolio. On the contrary, asset allocation is an investment strategy in which an investor distributes accessible capital among various financial securities such as stocks, cash, and bonds.

The portfolio optimisation problem is an asset allocation problem focusing on the distribution of funds to find an optimum portfolio. The optimum portfolio is determined by maximising a return or minimising risk. A risk-averse investor may be willing to impose realistic constraints such as bounding, transaction cost, and cardinality constraints. Portfolios are usually built from many asset types for diversification, even though it is possible to be constructed from a single asset type. In an effort to solve the portfolio optimisation problem, Markowitz formulated a compromised mean-variance model which seeks to maximise the expected return for a given level of risk, or alternately, minimise the anticipated risk for a specific return. The mean-variance model is solvable by a quadratic programming with the following drawbacks [1, 2]:

- Quadratic programming fails when the real-world constraints, such as transaction costs and bounding constraints, are incorporated into the standard model.
- The model employs standard deviation as a risk measurement with an assumption that the data is normally distributed. Unfortunately, the stock market data is mostly asymmetrically distributed in nature.
- The model assumes that the mean, variance and autocorrelation are stationary whereas stock market series are usually non-stationary.

This research addresses the portfolio optimisation problem by integrating kurtosis and skewness, in addition to the mean and standard deviation, to accommodate asymmetric distribution nature of stock market data. It also assimilates transaction costs as one of the realistic constraints in the modified Sharpe ratio. The problem compares the performance of the three

algorithms, namely, particle swarm optimisation (PSO) , genetic algorithm (GA) and deep reinforcement learning (DRL) . The former algorithms are widely used to generate high-quality solutions in optimisation problems, whilst the latter has been proved to be more effective in solving complex problems that cannot be solved by conventional techniques [3].

1.2 Literature review

This section provides a brief overview of the work done in the field of a portfolio optimisation problem that involves PSO, GA, and DRL.

1.2.1 Metaheuristic optimisation

For several years great effort has been devoted to the study of portfolio management in a constrained portfolio optimisation problem. Most studies tend to focus on heuristic algorithms, particularly PSO.

In [4] a PSO model was introduced to solve portfolio selection. The model was implemented in three instances: 8 stocks, 15 stocks and 46 stocks for both unrestricted portfolio and restricted portfolio. They used data extracted from the Shanghai stock exchange 50 indexes consisting of daily returns. Their PSO model outperformed GA and VBA solver (an excel tool used for optimisation and simulation of business and engineering models) on numerous occasions by applying Sharpe ratio as a fitness measure. Moreover, the PSO algorithm produced better optimal portfolios compared to the other two algorithms.

A two-stage process was suggested for portfolio contribution [5]. In the first stage, investment satisfied capability index (ISCI) was employed for selecting stocks with better performance measured by the return on investment (ROI). In the second stage, PSO was successfully used to adjust stock weights. The results demonstrate that the PSO is ideally suited to tackle the asset allocation problem.

The use of mean-variance has been criticised because it is only applicable to quadratic objective functions [6]. In overcoming this shortcoming, coherent risk measures were proposed contrary to the variance for portfolio selection problem. Particle swarm optimisation was used as a training algorithm. In comparison with standard mean-variance, their results produced better returns.

Alternatively [3] compared GA and PSO for a constrained portfolio optimisation problem applying Value-at-Risk (VaR) as a risk measure. Although both algorithms produced optimised solutions, PSO converged quicker.

Particle swarm optimisation is not the only heuristic algorithm capable of solving the portfolio optimisation problems. In [7] non-dominated sorting genetic algorithm (NSGA-II) was considered for optimisation of mean-VaR tested on a portfolio with 40 best-performing stocks of S&P 100 for the period between 15 January 2008 to 6 September 2013. They also proposed calculating VaR from univariate generalised auto-regressive conditional heteroskedasticity (GARCH). For testing purposes, NSGA-II was applied to optimise mean-historical VaR and NSGA-II outperformed Linear programming (LP). The proposed mean-univariate GARCH VaR was then compared with two benchmark models: mean-multivariate GARCH VaR and mean-historical VaR, where the proposed model produced a better frontier.

The clustering-based portfolio approach was put forward using a GA based on investor information. Stocks were selected from Korean Composite stock price index (KOSPI) in terms of three investor clusters: foreign, institutional and individual investors. Genetic algorithm was then applied to the selected clusters. The clustered-based approach appeared to have yielded higher returns, contrary to the buy-and-hold strategy [8].

1.2.2 Deep reinforcement Learning approach

In recent years there has been a growing interest in applying DRL due to the significant amount of its progress. Recurrent reinforcement learning (RRL) was proposed for optimising portfolio weights with two assets taking Sharpe ratio as an objective function. Their neural network took price series as input and softmax as output [9]. Recurrent reinforcement learning was further extended to regime-switching recurrent reinforcement learning (RSRRL) in an effort to cater for non-linearity on financial data. They applied two variations of RSRRL, threshold version (TRRL) and smooth version (STRRL) to produce trading signals which determine whether to go long or short on each asset in the portfolio resulting in an adjustment of portfolio weights [10].

In [11] on-policy (λ) and off-policy Q(λ) was used for the retirement portfolio in the absence of tax and transaction costs. Their model had four discrete states and five actions. The test was run on two portfolio instances each containing two assets. The first instance contained S&P 500 and AGG whereas the second one was made up of the S&P 500 and T-note. Q-learning was used for optimising portfolios with risk-less assets and risky assets. The idea was to produce a series of actions at each state that maximise the total return. They attained a better return and drawdown compared to the market index[12].

1.3 Research gap

The literature review put forward does not cater to the asymmetric nature of stock market prices. Most of them applied the standard deviation as a measure of risk. However, the distribution of stock prices usually deviates from its mean, as a result considering only standard deviation produces misleading results. Besides, none of the literature considered applied deep reinforcement learning to solve the portfolio optimisation problem.

1.4 Research questions

The central questions are: first, can PSO and GA be used in solving the portfolio optimisation problem with real-world constraints (transaction costs)? Second, to investigate the deep reinforcement learning's potential using PSO and GA as benchmark comparison?

1.5 Scope and objective of study

In answering the research questions, the following objectives are considered:

- To apply deep reinforcement learning, GA and PSO to solve portfolio optimisation problems, taking into account transaction costs.
- To compare and evaluate the performance of PSO, GA and DRL in optimising portfolio weights in terms of:
  - Reliability - a measure of an algorithm's capability to reach a good solution consistently.
  - Accuracy - an algorithm's capability to obtain the best solution(s).
  - Efficiency - a measure of how fast an algorithm runs.

- To apply modified Sharpe as an evaluation function that includes all four moments (mean, standard deviation, skewness, and kurtosis).
- To back-test using the real dataset from listed companies in Johannesburg stock exchange (JSE) as the training and testing data.
- To employ buy-and-hold trading strategies, and backtest based on uniform weights allocations and equity financial portfolios.

- To use cumulative returns as a measure of algorithm performance.

1.6 Out of Scope

This dissertation does not attempt to provide the best learning algorithms but compares the performance on each learning algorithm.

1.7 Subsequent structure

Apart from this introductory chapter, the remaining chapters of the dissertation are organised as follows:

2 Portfolio theory

The purpose of this chapter is to present the reader with a thorough understanding of portfolio theory and its conceptual terms. Furthermore, this chapter deals with the most famous asset allocation formulations that include mean-variance, mean-VaR and Sharpe ratio. Besides, the chapter gives a brief explanation of the modified Sharpe ratio that takes into account the third and fourth moments of portfolio returns.

3 Evolutionary algorithms

This chapter provides insight into the theoretical background of PSO including various topologies employed. The chapter also addresses in detail the concepts and operators used in GAs that include crossover, mutation, and selection. It explains the origin of GAs as inspired by Darwin's theory of natural evolution. Finally, the chapter applies the GA and PSO in solving the asset allocation problem explained in chapter 2.

4 Deep reinforcement learning

The purpose of this chapter is to introduce notations used to formulate reinforcement learning problem. Reinforcement learning is presented in terms of Markov decision process and the underlying mathematical concepts. Furthermore, dynamic programming as the natural way of solving reinforcement learning problem is described. Furthermore, some of the algorithms used in an imperfect environment are outlined. The chapter also explains how the deep deterministic policy gradient can be used in the asset allocation problem.

5 Numerical results and analysis

This chapter deals with the implementation of the learning algorithms discussed in chapters 3 and 4 as applied to the asset allocation problem and trading strategy. It also describes the modified Sharpe ratio with a self-financing budget constraint as the evaluation function. The algorithms are trained and tested on the historical stock price time-series of the risky assets obtained from JSE. The chapter also highlights the performance of the three algorithms compared to one another and then with the benchmark.

6 Conclusion

This chapter summarises the results and findings obtained on applying algorithms defined in chapters 3 and 4. The future suggestions and recommendations on applications of portfolio optimisation problems are provided in this chapter.

2

Portfolio theory

The purpose of this chapter is to present the reader with a thorough understanding of portfolio theory and its conceptual terms that have been applied to obtain numerical results later in this dissertation. These theories and concepts include among others the most famous asset allocation formulations that include mean-variance, mean value-at-risk and Sharpe ratio. Furthermore, the chapter gives a brief explanation of the modified Sharpe ratio that takes into account the third and fourth moments of portfolio returns.

2.1 Stocks

An asset is an investment instrument that can be bought or sold. Assets are classified as either real, financial or intangible. On the one hand, real assets are tangible assets that derive their worth from their physical properties or substances. They include jewellery, land, real estate and commodities like oil and iron. Intangible assets, on the other hand, are valuable properties that are non-physical. Intangible assets comprise patents, trademark, and intellectual properties. In contrast to the intangible and real assets, financial assets are liquid properties that derive their values from contractual rights or ownership claims. Financial assets include stocks, bonds, mutual funds, and bills. When one buys stock from a company, he buys a small portion of the company. If for instance, a company emits 100 shares, and one decides to buy 1 share from the company, he owns 1% stakes rights in decision-making, and he is entitled to 1% share dividend when the company makes profits. Likewise, when the company goes bankrupt he loses big. If at any time, for some reason, one wants to sell shares he has acquired, he looks for a suitable buyer. Looking for the right buyer who meets the required price might be a tedious task. Financial markets, like JSE, make it easy for buyers and sellers to meet with low transaction costs.

2.2 Prices and returns

During a trading period, the prices of assets go up and down. Normally, only four movements are considered, in particular, the opening, lowest, highest and closing price. For a continuous market like stock markets, the closing price of the previous period is taken as the opening of the subsequent period. The period ranges from daily, monthly to yearly, based on the needs of the investor. If the closing price of an asset at time t is p_t, then the closing price of the previous period $t-1$ is p_{t-1}. In this instance, the return R, the rate of return r_t and logarithmic rate of

return r_l of the asset from time $t-1$ to time t are:

Return

$$R = \frac{p_t}{p_{t-1}} \tag{2.1}$$

Rate of return

$$r_t = \frac{p_t - p_{t-1}}{p_{t-1}} = \frac{p_t}{p_{t-1}} - \frac{p_{t-1}}{p_{t-1}} = \frac{p_t}{p_{t-1}} - 1 \tag{2.2}$$

Logarithmic rate of return

$$r_l = \ln(\frac{p_t}{p_{t-1}} - 1) \tag{2.3}$$

2.3 Short selling

An investor can sell assets that he/she does not own. This is called short selling. Short selling is a two-stage process. In the first process, the broker lends an investor an asset from a pool of assets of its customers. The brokerage sells such assets on behalf of the investor, and the sale is credited against the investor's account. The investor's account asset sheet appears as a negative amount in association with the shorted stock. The investor receives a return in monetary form because of the sale of the asset. In the second process, the investor requests the brokerage to repurchase the same asset sold earlier and return it into the pool. Suppose p_{t-1} is the amount in cash that the investor received on the sale of the shorted assets. And p_t is the amount in monetary terms the brokerage used to repurchase the same number of assets formerly short sold. Then the profit is made only if $p_{t-1} > p_t$. The return and rate of return on this transaction are given:

$$R_p = \frac{-p_t}{-p_{t-1}} = \frac{p_t}{p_{t-1}} \tag{2.4}$$

$$r_p = \frac{(-p_t) - (-p_{t-1})}{-p_{t-1}} = \frac{p_t - p_{t-1}}{p_{t-1}} \tag{2.5}$$

Short selling can be too risky and it is discouraged and disallowed by many brokerage firms although, it can be profitable.

2.4 Mean-variance

This section explains the basis of the portfolio optimisation problem as a maximisation of the returns, and a minimisation of the risks associated. Furthermore, the section outlines the importance of portfolio construction in general.

2.4.1 Portfolio and diversification

What happens when an investor puts all his money in a company, and the company goes bankrupt? The obvious answer is that the investor loses all his investment. To alleviate the risk brought by investing in one company, investors diversify their investments by constructing pools of assets to form portfolios. These portfolios consist of assets from more than one asset type in most cases. The importance of diversification is to reduce risk by spreading the risk among assets in the portfolio.

2.4.2 Portfolio return and risk

An investor intends to maximise his returns while keeping his risk as low as possible. In a mean-variance formulation, the return is measured by the mean, while the risk is represented by the variance. Mathematically, the portfolio return and risk are represented as:

Minimise portfolio variance as:

$$\sigma_p^2 = \sum_{i=0}^{N} \sum_{j=0}^{N} w_i w_j \sigma_{ij} \tag{2.6}$$

Maximise portfolio mean as:

$$\mu_p = \sum_{i=0}^{N} w_i \mu_i \tag{2.7}$$

subject to

$$\sum_{i=0}^{N} w_i = 1 \tag{2.8}$$

$$0 \leq w_i \leq 1 \tag{2.9}$$

where w_i is a fraction of asset i in the portfolio, N is the number of different assets, σ_{ij} is a covariance between assets i and j, μ_i is a mean return of asset i. σ_p^2 and μ_p are variance and expected mean return of the portfolio respectively.

2.5 Modern portfolio theory (MPT)

In an effort to solve the portfolio construction problem, Markowitz formulated a compromised mean-variance model which seeks to maximise an expected return for anticipated risk or alternately, minimise the risk for the specific rate of return, thereby setting one of the objective functions as part of constraints [1]. In this model, variance and mean are considered as risk and return measurements respectively. The model considers the problem as a single-objective optimisation problem by taking one of the objectives as one of the constraints as to:

Minimise portfolio variance as:

$$\sigma_p^2 = \sum_{i=0}^{N} \sum_{j=0}^{N} w_i w_j \sigma_{ij} \tag{2.10}$$

subject to

$$\sum_{i=0}^{N} w_i \mu_i = \mu_p \tag{2.11}$$

$$\sum_{i=0}^{N} w_i = 1 \tag{2.12}$$

$$0 \leq w_i \leq 1 \tag{2.13}$$

OR

Maximise portfolio mean as:

$$\mu_p = \sum_{i=0}^{N} w_i \mu_i \tag{2.14}$$

subject to

$$\sum_{i=0}^{N} \sum_{j=0}^{N} w_i w_j \sigma_{ij} = \sigma_p^2 \tag{2.15}$$

$$\sum_{i=0}^{N} w_i = 1 \tag{2.16}$$

$$0 \leq w_i \leq 1 \tag{2.17}$$

2.5.1 Efficient frontier

The efficient frontier is a curve that shows all optimal portfolios in a risk-return framework [1]. The efficient portfolio is defined as the portfolio that maximises the expected return for a given amount of risk, or a portfolio that minimises the risk subject to a given expected return. Fig. 2.1 demonstrates all portfolios with the efficient portfolios on the left boundary of the arc. A risk-averse investor will want to invest in the efficient portfolio with the least amount of risk disregarding the return. On the other hand, a risk lover will prefer to invest in an efficient portfolio with the maximum expected return without considering the risk.

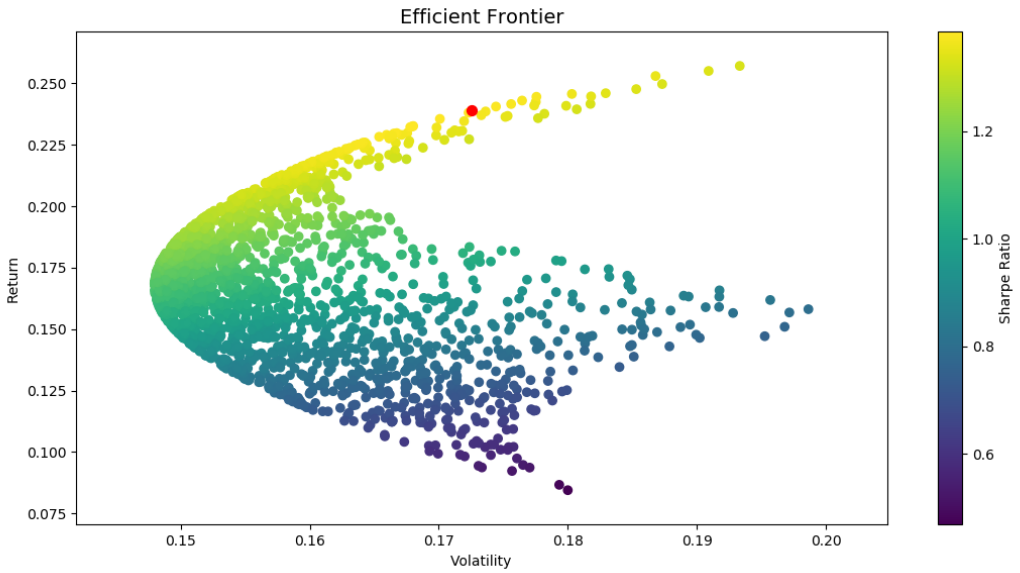


Figure 2.1: Efficient frontier

[1] plot of the return as a mean against volatility in the form of the variance. Efficient frontier is similar to the concept of a Pareto front which is found often in multi-objective optimisation

2.6 Other portfolio optimisation formulations

2.6.1 Mean-VaR

Apart from the mean-variance model, which employs the variance as a risk measure, there are other risk measures used, among others is VAR [13]. Value-at-risk is widely utilised by many financial firms to quantify the risk, replacing variance in the standard mean-variance model to form a new model called mean-VaR. The VaR measures how much a portfolio might lose in a certain period, such as a day or a month, given a normal market condition. There are three types of VaR, namely parametric-based on variance-covariance; non-parametric, which uses historical simulation; and Monte Carlo (MC) simulation method.

2.6.2 Sharpe

Sometimes an investor's preference is not to invest in an efficient portfolio that yields maximum return or minimum risk, but to invest in the efficient portfolio with the maximum Sharpe ratio. The Sharpe ratio is defined as the expected return per unit risk and the ratio was originally called reward-to-variability [14, 15, 16]. Generally, any Sharpe ratio above 1.0 is acceptable by investors. A ratio above 2.0 is considered good and a ratio of 3.0 and higher is regarded excellent. Mathematically, the ex-ante Sharpe ratio is expressed as

$$S = \frac{\mu_p - r_f}{\sigma_p^2} \tag{2.18}$$

Where S is the Sharpe ratio, r_f is the risk-free return such as Treasury bill, μ_p is the asset mean return. $\mu_p - r_f$ is the expected value of the excess of the asset return over the benchmark, and σ_p^2 is the standard deviation of the asset excess return.

The Sharpe ratio characterises how well the return of an asset compensates the investor for

the risk taken thus risk-free return is taken as a benchmark.

2.7 Higher moments

The mean-variance, the Sharpe ratio, and the mean-VaR models consider only the first two statistical moments: mean and standard deviation, which assume returns to be normally distributed. In addition to the first two moments, there are two more to consider: skewness and kurtosis [17].

2.7.1 Skewness

Portfolio skewness is defined as the third moment, and is calculated as:

$$Skewness = S = \sum_{i,j,k=1}^{N} w_i w_j w_k s_{ijk} \tag{2.19}$$

where

s_{iii} is the skewness coefficient for an individual asset

s_{iij} is the co-skewness coefficient between assets i and j

s_{ijk} is the coefficient of co-skewness between assets i, j and k.

A symmetrical distribution has a skewness of zero. A positive skewness is a distribution with a long tail to the right, whilst a negative skewness has a long tail to the left.

2.7.2 Kurtosis

Kurtosis (fourth moment) is characterised as a measure of peakedness or flatness of a distribution relates to the Gaussian distribution given by:

$$Kurtosis = K = \sum_{i,j,k,l=1}^{N} w_i w_j w_k w_l k_{ijkl} \tag{2.20}$$

where

k_{iiii} is the kurtosis coefficient for an individual asset

k_{iijj} is the co-kurtosis coefficient between two assets

k_{ijjj} is the coefficient of co-kurtosis between three assets

k_{ijkl} is the coefficient of co-kurtosis between four assets

A normal distribution has a kurtosis of zero (also known as mesokurtic). A distribution with a positive kurtosis is called leptokurtic. A distribution with a negative kurtosis (platykurtic) has a flat top near the mean and a shorter, thinner tail. Figure 2.2 demonstrates the kurtosis and skewness.

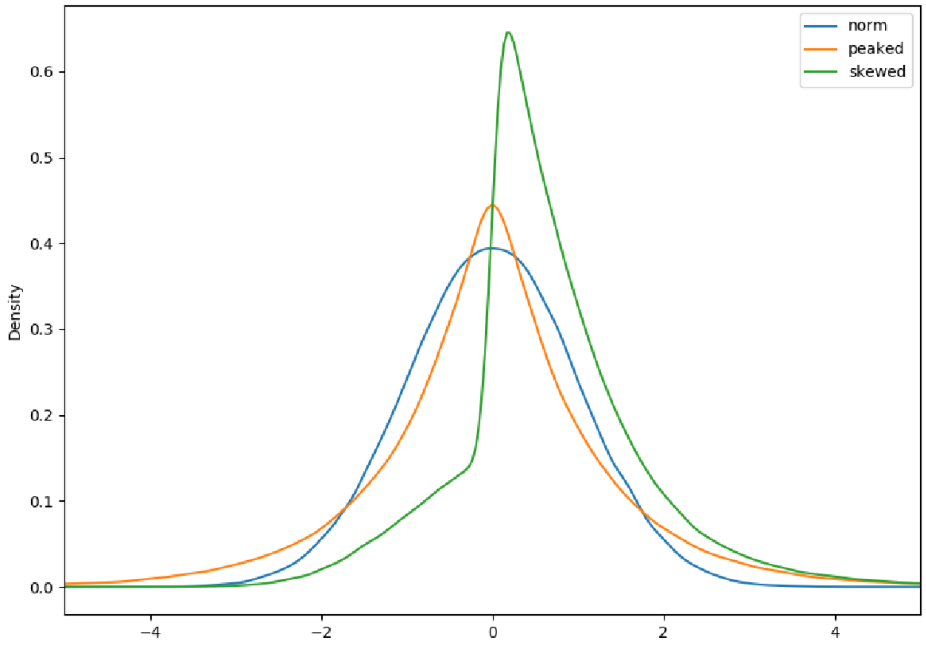


Figure 2.2: Kurtosis and skewness sample

2.7.3 Modified Sharpe ratio

Returns in financial assets usually deviate from their normal distribution, to form kurtosis and skewness different from zero (symmetric distribution). As a result, using standard deviation as a means of measuring risk alone produces misleading results. Consequently, modified VaR (mVaR) was developed to include all four moments, namely mean, standard deviation, skewness, and kurtosis [18, 19]. The modified VaR of the portfolio is calculated as:

$$mVaR_p = \mu_p + Z\sigma_p \tag{2.21}$$

$$Z = \left(z_c + \frac{1}{6}(z_c^2 - 1)S + \frac{1}{24}(z_c^3 - 3z_c)K - \frac{1}{36}(2z_c^3 - 5z_c)S^2 \right)$$

Therefore the Sharpe ratio that includes the modified VaR becomes:

$$S_p = \frac{(\mu_p - r_f)}{mVaR_p} \tag{2.22}$$

where

μ_p = Portfolio mean

r_f = Free-risk return

σ_p = Portfolio standard deviation

Z = Cornish-Fisher asymptotic expansion for the quantile of non-Gaussian distribution

z_c = Quantile of the distribution (-1.64485 for the confidence level of 95%)

K = Portfolio kurtosis

S = Portfolio skewness

S_p = Portfolio Sharpe

2.8 Brief chapter summary

Every investor aims to maximise his/her return with low risk. In the stock market, investors trade stock, or rather buy and sell a small portion of stock called shares. In this instance, the investor is said to have acquired a stake or to own a small portion of the company.

It is not usually easy to maximise the return and minimise the risk simultaneously. Markowitz proposed the mean-variance model which seeks to find a compromised solution. The mean-variance maximises the expected return for a specific level of risk or alternately minimises the anticipated risk for the expected return.

Other formulations apart from the mean-variance model do exist, where the Mean-VaR and the Sharpe ratio are among the most famous formulations. Mean-variance, Mean-VaR and Sharpe ratio consider the first two moments of portfolio returns and they assume the data to be normally distributed. However, most of the real-world problems exhibit features that require more than just the first two moments. This calls for modification of the formulation to include the last two moments of return as well. As a result, the modified Sharpe ratio was introduced to include all the four moments.

3

Evolutionary algorithms

This chapter provides insight into two evolutional algorithms used in this dissertation, in particular PSO and GA [20]. It gives various topologies employed in PSO. The chapter also addresses, in detail, the concepts and operators applied in GAs that include crossover, mutation, and selection. It explains the origin of the GAs as inspired by Darwin's theory of natural evolution.

3.1 Swarm optimisation

3.1.1 Basic particle swarm optimisation

Particle swarm optimisation is a simple, stochastic, population-based and computationally efficient technique for solving continuous and discrete optimisation problems, first pioneered by Kennedy and Eberhart [21]. Particle swarm optimisation initially targeted simulation of the social psychology of the bird flocking or fish schooling. For optimisation problems, individuals referred to as particles represent candidate solutions, with elements of the particles symbolising parameters to be optimised. In search of a solution, particles fly over a search space with distinct velocities while at specific positions. The change in position of particles is driven by the social-psychological behaviour of an individual to mimic the success of others. The position of each particle is updated by the formula:

$$\mathsf{x}_i(t+1) = \mathsf{x}_i(t) + \mathsf{v}_i(t+1) \tag{3.1}$$

where

- $\mathsf{v}_i(t+1)$ is velocity vector of particle *i* at time-step $t+1$
- $\mathsf{x}_i(t)$ is position vector of particle *i* at time-step t

Algorithm 1 : Basic PSO pseudo-code

```
1:  procedure PSO
2:      Initialise swarm of particles
3:      Evaluate each particle
4:      while termination condition not satisfied do
5:          for each particle do
6:              Calculate particle fitness value as
7:              if fitness value is better than pbest then
8:                  set current fitness value as pbest
9:              end if
10:             if pBest is better than gbest then
11:                 set pbest as gbest
12:             end if
13:         end for
14:         for each particle do
15:             Calculate velocity
16:             update position
17:         end for
18:     end while
19:     return gbest
20: end procedure
```

3.1.2 Neighbourhood topologies

Particles in a swarm exchange information based on neighbourhood topologies. Some neighbourhood topologies perform better than others. Some of the most prominent topologies are a star, ring and wheel topology.

Star topology

A star topology, also known as gbest, is a topology where all particles are fully connected to one another. As a result, each particle is a neighbour of everyone in the swarm as shown in Figure 3.1 below. The best performing particle, called the global best, is updated at each iteration, and all the other particles move towards its direction. Although this topology converges rapidly, it is susceptible to fall into local optima [22].

Ring topology

In a ring topology as shown in Figure 3.4, particles are directly connected to k neighbours, and each particle is duly influenced by the performance of its immediate neighbours. The ring topology is slow to converge because one region of the ring could attain local optimum while the other region is still searching, yet its performance surpasses that of the star topology in numerous applications. The ring topology reacts more like the star topology when k equals the number of particles in the swarm [23, 24, 25].

Wheel topology

In a wheel topology, there is one central particle called focal that all other particles share information through. The central particle is selected randomly among all particles in the swarm. Refer to Figure 3.4. At each time-step t, particles fly towards the direction of the central particle whereas the central particle flies towards the direction of the best performing particle [25].

Other renowned topologies include cluster, von Neumann, pyramid, and square. Figures 3.3 and 3.5 below show Square topology and Neumann topology respectively.

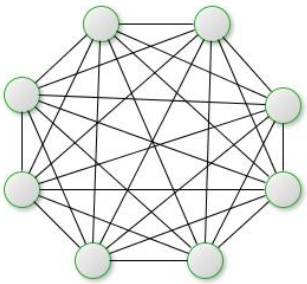


Figure 3.1: Gbest

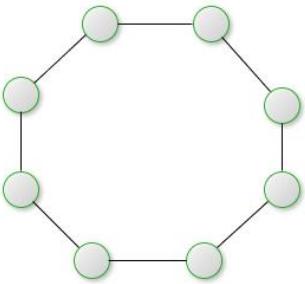


Figure 3.2: Lbest

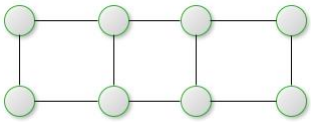


Figure 3.3: Square

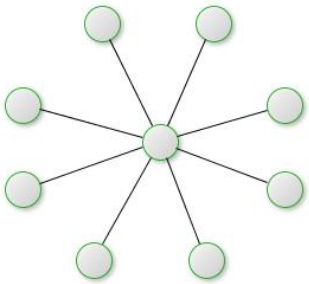


Figure 3.4: Wheel

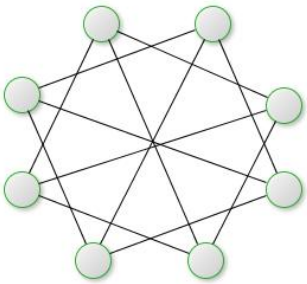


Figure 3.5: von Neumann

3.1.3 Global best PSO

The global best PSO uses a star topology for its communication where there is exactly one global best particle of which all other particles are updated through. Velocity v_i is updated as:

$$v_i(t+1) = c_1 v_i(t) + c_2 \eta_1 (y_i - x_i) + c_3 \eta_1 (g_i - x_i) \tag{3.2}$$

where

- c_1 and c_2 are positive acceleration coefficients that control the trade-off between exploration-exploitation,
- η_1 and η_1 are constants in the range [0,1],

- $y_i(t)$ is the personal best position calculated as (assuming minimisation):

$$y_i(t+1) = \begin{cases} y_i(t) & \text{if} \quad f(x_i(t+1)) \geq f(y_i(t)) \\ x_i(t+1) & \text{if} \quad f(x_i(t+1)) < f(y_i(t)) \end{cases} \tag{3.3}$$

- $g(t)$ is the global best position calculated as:

$$\begin{aligned} g(t) \in & \{y_0(t), \ldots, y_n(t)\} | f(g(t)) \\ = & min\{f(y_0(t)), \ldots, f(y_n(t))\} \end{aligned} \tag{3.4}$$

Where n is the number of particles in the swarm.

3.1.4 Local best PSO

Local best PSO in its basic form uses a ring topology. It reduces information sharing between particles by dividing the swarm into neighbourhoods. Instead of particles having the full knowledge of the global best position, each particle shares information with only its immediate neighbours. Nonetheless, to enable convergence, there must be over-leaping among neighbourhoods [26].

3.2 Genetic algorithm

3.2.1 Basic GA

A genetic algorithm (GA) is a search meta-heuristic technique for solving optimisation problems inspired by natural evolution theory and first instigated by Holland [27, 28]. The genetic algorithm forms part of a larger class of evolutionary algorithms (EAs) . The concept was later extended and formalised by Goldberg [29].

In an optimisation problem, a population of individuals (chromosomes) representing candidate solutions is allowed to evolve towards optimum solutions. Each candidate solution possesses a set of properties (genes) that can be altered.

Table 3.1 summarises the relationship between natural genetics and GA

Table 3.1: Comparison of genetic terminology

Natural genetics	**Genetic algorithm**
phenotype	parameter set, solution alternative, decoded structure, decoded solution
genotype	structure, coded solution
chromosome	string
gene	bit, feature, character, detector
locus	string position
allele	feature value

Generally, the process of a GA begins by randomly generating a set of individuals called a population. The initial population can be seeded around an optimal solution space. Population size ranges from hundreds to thousands of possible solutions depending on the nature of the problem at hand. Each individual is characterised by a set of parameters designated as genes. Genes are joined together into a string to form a chromosome. The chromosomes signify candidate solutions for the problem at hand. Many representations do exist, with the binary presentation being the most prominent one, and where genes are encoded as either 1 or 0. Figure 3.6 below illustrates an example of a chromosome.

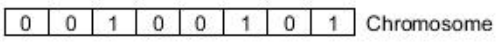


Figure 3.6: Chromosome example

Algorithm 2 : Basic GA pseudo-code

```
1:  procedure GA
2:      t = 0;
3:      Randomly initialise a population
4:      Evaluate the population
5:      while termination condition not satisfied do
6:          Select best-fit chromosomes for reproduction
7:          Breed new offspring through crossover and mutation operations
8:          Evaluate offspring
9:          Replace least-fit population with offspring
10:     end while
11:     return population
12: end procedure
```

3.2.2 Fitness function

The fitness function is the function used to evaluate the fitness of each chromosome in the population. For this dissertation, the modified Sharpe ratio, defined in Equation 2.22, is used as an evaluation function.

3.2.3 Selection

At each generation, a portion of the population is selected for breeding a new generation. The selection criterion is usually fitness-based, where the fittest individuals stand a higher chance to partake in the breeding process although other selection criteria such as random selection exist.

3.2.4 Crossover

Crossover is the most important operator in the GA. The selected parents are allowed to mate. A crossover point is chosen randomly from within the genes of each parent, and offspring is produced by exchanging genes of parents until a crossover point is reached. The offspring replace their parents in the population if they are found fitter than the parents. Figure 3.7 below

illustrates one example of crossover operation between two chromosomes.

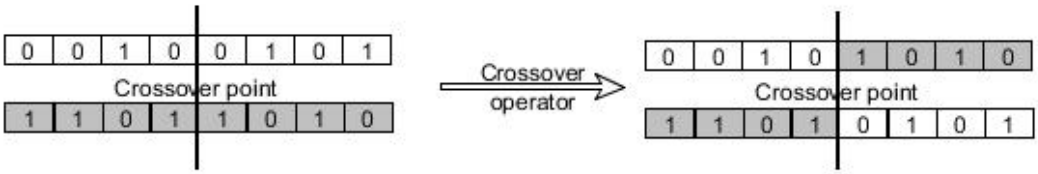


Figure 3.7: Crossover example

3.2.5 Mutation

Some offspring in the population are allowed to undergo a process of mutation with a certain probability. In mutation as demonstrated in Figure 3.8, genes of the same chromosome are selected and allowed to swap loci thereby forming an entirely new chromosome different from parent chromosome. Mutation maintains diversity within the population, and prevents premature convergence.

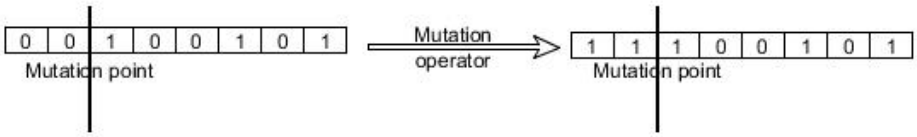


Figure 3.8: Mutation example

3.2.6 Termination

The generation process is repeated until one of the following conditions is met:

- Generation number: A user specifies the maximum number of generations to run.
- Evolution time: Once the elapsed evolution time exceeds the maximum evolution time specified by the user.
- Fitness threshold: The best fitness exceeds the maximum/minimum user-specified fitness.
- A solution has been found that satisfies criteria.
- A maximum number of generations is reached.
- The allocated time budget is reached.
- Solutions do not evolve any-more.

3.3 Brief chapter summary

In this chapter, the theoretical backgrounds of the two EAs: PSO, and GA were presented. The PSO tries to mimic the social behaviour of birds flocking or fish schooling. The algorithm randomly generates a set of particles, also known as a swarm, that represent the candidate solutions. Each particle resides at its position, and flies with a certain velocity in search of an optimum solution. The positions are updated based on the neighbourhood. Individuals in the swarm have one or more neighbours depending on the topology employed.

Similarly, the GA is a population-based and EA that is inspired by the natural evolution theory. Initially, the GA algorithm generates individuals called chromosomes. The chromosomes, like particles, correspond to the candidate solutions for the problem at hand. The GA utilises crossover and mutation as its primary operators. In the crossover, the chromosomes selected

exchange parameters producing entirely new offspring. Apart from the crossover, some chromosomes are allowed to undergo mutation with a certain probability. On the other hand, in mutation, genes of the same chromosome exchange positions, or rather, loci thereby providing yet another chromosome different from the parent chromosome. The process terminates when the optimum solution is found, or when other criteria set by the designer are met.

4

Deep reinforcement learning

The purpose of this chapter is to introduce notations used to formulate the reinforcement learning problem. Reinforcement learning is presented in terms of the Markov decision process and the underlying mathematical concepts. Moreover, dynamic programming as the natural way of solving the reinforcement learning problem is described. Furthermore, some of the algorithms used in an imperfect environment are outlined. The chapter also explains how the deep deterministic policy gradient can be used in the asset allocation problem.

4.1 Reinforcement learning

Reinforcement learning is a type of machine learning, along with supervised and unsupervised learning, concerned with learning from interaction to attain a reward [30]. The problem is classified as one of the sequential decision problems where the decision-maker, called the agent, makes successive observations of the environment, and executes a series of actions before taking the final decision. In return, the agent receives the reward from the environment based on the action taken. The goal of the agent is to determine the best strategy that yields a maximum reward. During interaction with the agent, the environment evolves stochastically. That is, the action taken may impact future reward. The study of the Markov decision process (MDP) is dated back to 1957. Markov decision process is described as a discrete-time stochastic control process that provides a mathematical framework for modelling sequential decision making under uncertainty and the influence of the decision-maker in several applications including inventory control systems and finance [31, 32, 33, 34].

The reinforcement learning problem is well represented in terms of the MDP. The RL can be schematised as in Figure 4.1: at time step t, the agent observes the current state $s_t \in \mathcal{S}$ and interacts with the environment by taking an action $a_t \in \mathcal{A}$ in the action space. In response, the environment emits an immediate reward/penalty $r_t \in \mathcal{R}$ as it transits to the next state s_{t+1} in accordance with some probability distribution dependent solely on the action selected by the agent [30].

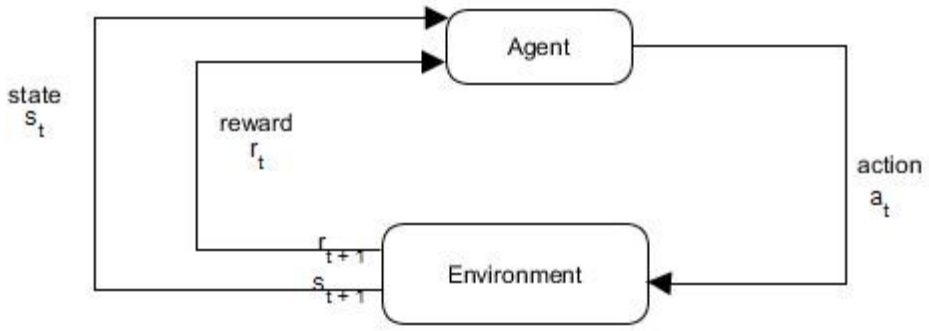


Figure 4.1: The agent-environment interaction for MDP

Generally, the MDP is well specified by a tuple $< \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma >$

where

- $\mathcal{S}$ is a finite set of states called observable state space;
- $\mathcal{A}$ is a finite set of actions called action space;
- $\mathcal{P}$ is a state transition probability kernel;
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is a reward function;
- γ is a discount factor, where $\gamma \in [0, 1]$. It determines the importance of the future rewards relative to the current state.;

The state space $\mathcal{S}$ can be either finite or continuous. The matrix $\mathcal{P}$ describes a random evolution of the system, at time *t* the system at state s, taking action *a* will transit to the next state s' irrespective of the system history. The probability of the agent being in state s' at time *t+1* by taking the action *a* is given by

$$\mathcal{P}(s, a, s') = \mathbb{P}[\mathcal{S}_{t+1} = s' \mid \mathcal{S}_t = s, \mathcal{A}_t = a] \tag{4.1}$$

Following the random transition, the reward $\mathcal{R}_{t+1}$ that the agent receives after moving to state s' from s, taking action a is expressed as

$$\mathcal{R}(s, a) = \mathbb{E}[\mathcal{R}_{t+1} \mid \mathcal{S}_t = s, \mathcal{A}_t = a] \tag{4.2}$$

Moreover, a policy denoted as π is defined as a mapping from states to the probability of selecting actions at each time step. In deterministic policy denoted $\pi(s)$, every state has explicit action to be taken whereas in stochastic policy, represented as $\pi(a|s)$, there is no well-defined action to be taken.

Furthermore, a system history, or equivalently, a trajectory, is a random sequence of the actions and states defined as:

- a state sequence which is a Markov process with tuple $< \mathcal{S}, \mathcal{P} >$;
- a state-reward sequence which is a Markov reward process with tuple $< \mathcal{S}, \mathcal{P}, \mathcal{R}, \gamma >$;

where the probability of an agent at state s to move to the next state s' following the policy π is given as:

$$\mathcal{P}_\pi(s, s') = \int_A \pi(s, a)\mathcal{P}(s, a, s')da \tag{4.3}$$

and the corresponding reward is given by

$$\mathcal{R}_\pi(s) = \int_{\mathcal{A}} \pi(s, a)\mathcal{R}(s, a)da \tag{4.4}$$

4.1.1 Discounted reward formulation

The performance of the agent is measured as a total discounted reward or as an average reward. In the discounted formulation, the reward is expressed as the expected discounted return:

$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{4.5}$$

where γ is a parameter called discounted rate and $\gamma \in [0, 1]$. If $\gamma \to 0$, the agent selects its action in a myopic manner. When $\gamma \to 1$, the agent selects its action in a far-sighted manner. Various reasons for discounting reward are:

- to circumvent infinite returns in a cyclic Markov process;
- it is mathematically sound;
- unpredictability about the future may not be represented;
- immediate rewards may not yield more interest than delayed rewards as in the case of financial problems;

It is often crucial to determine the value of the state denoted as $V_\pi(s)$, also known as the state-value function, that defines the quality of the agent at the particular state. The state-value function is measured in terms of the future rewards that can be expected starting at the particular state while following a policy π. Formally $V_\pi(s)$ is expressed as

$$V_\pi(s) = \mathbb{E}_\pi[G_t \mid \mathcal{S}_t = s] \tag{4.6}$$

The state-value function can decompose into the Bellman equation [31] as

$$V_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma V_\pi(\mathcal{S}_{t+1}) \mid \mathcal{S}_t = s] \tag{4.7}$$

Most importantly, the value to consider is the value that measures the quality of taking an action a starting at a state s under a policy π. This value is called the action-value function denoted $Q_\pi(s, a)$, and expressed formally as

$$Q_\pi(s, a) = \mathbb{E}_\pi[G_t \mid \mathcal{S}_t = s, \mathcal{A}_t = a] \tag{4.8}$$

The $Q_\pi(s, a)$ can also decompose into the Bellman equation as

$$Q_\pi(s, a) = \mathbb{E}_\pi[R_{t+1} + \gamma Q_\pi(\mathcal{S}_{t+1}, \mathcal{A}_{t+1}) \mid \mathcal{S}_t = s, \mathcal{A}_t = a] \tag{4.9}$$

The relationship between $V_\pi(s)$ and $Q_\pi(s, a)$ is expressed as:

$$V_\pi(s) = \int_{\mathcal{A}} \pi(s, a) Q_\pi(s, a) da \tag{4.10}$$

There exist an optimal value function amongst all possible functions with the highest value than any other functions for all states: the optimal state-value function $V_*(s)$ and action-value function $Q_*(s, a)$.

The optimal state-value function $V_*(s)$ is the maximum achievable value of all state-value functions starting from state s over all policies

$$V_*(s) = \max_\pi V_\pi(s) \tag{4.11}$$

The optimal-action function $Q_*(s, a)$ is the maximum achievable action-value functions starting from a state s taking an action a over all policies:

$$Q_*(s, a) = \max_\pi Q_\pi(s, a) \tag{4.12}$$

The relationship between optimal state-value function and action-value function is rooted in that the state-value function is the maximum expected total return starting from a state s which is equivalent to the maximum of the action-value function $Q_*(s, a)$ over all possible actions, hence,

$$V^*(s) = \max_a Q^*(s, a) \tag{4.13}$$

Once $Q^*(s, a)$ has been obtained, the agent can act greedily by choosing $Q^*(s, a)$

$$\pi^*(s) = \arg\max_a Q^*(s, a) \tag{4.14}$$

Moreover, the optimal value functions are also recursively related by the Bellman optimality equations as

$$V_*(s) = \max_a \mathcal{R}(s, a) + \gamma \int_{\mathcal{S}} \mathcal{P}(s, a, s') V(s') ds' \tag{4.15}$$

$$Q_*(s, a) = \mathcal{R}(s, a) + \gamma \int_{\mathcal{S}} \mathcal{P}(s, a, s') \max_{a'} Q_*(s', a') ds' \tag{4.16}$$

4.1.2 Average reward formulation

Most studies in reinforcement learning focus mostly on the formulation of returns, in terms of discounted rewards, motivated by problems where rewards are interpreted in monetary terms that can earn interest, or where the probability of runs can be terminated at any time. Discounting reward tends to sacrifice bigger long-term rewards in favour of small short-term rewards. However, there are problems where discounting rewards are not applicable and an average reward formulation is proposed [35]. In the average reward formulation, the reward, designated by ρ^π, is expressed in terms of an average expected reward per step over time t, starting from

state t under a policy π

$$\rho_\pi = \lim_{T\to\infty} \frac{1}{T}\mathbb{E}\left[\sum_{t=1}^{T} R_t\right] \tag{4.17}$$

The aim of the agent is still to discover the optimal average policy π_* as

$$\pi_* = \arg\max_a \rho_\pi \tag{4.18}$$

Similar to the discounted reward formulation, the value functions are expressed respectively as

$$V_\pi(s) = \mathbb{E}_\pi\left[\sum_{k=1}^{\infty}(R_{t+k} - \rho^\pi) \mid \mathcal{S}_t = s\right] \tag{4.19}$$

and

$$Q_\pi(s,a) = \mathbb{E}_\pi\left[\sum_{k=1}^{\infty}(R_{t+k} - \rho^\pi) \mid \mathcal{S}_t = s, \mathcal{A}_t = a\right] \tag{4.20}$$

and they are reduced to the Bellman equations as

$$\begin{aligned} V_\pi(s) =& \mathbb{E}_\pi\left[(R_{k+1} - \rho^\pi) + \sum_{k=1}^{\infty}(R_{t+k+1} - \rho^\pi) \mid \mathcal{S}_t = s\right] \\ =& \mathbb{E}_\pi[(R_{k+1} - \rho^\pi) + V_\pi(S_{t+1}) \mid \mathcal{S}_t = s] \end{aligned} \tag{4.21}$$

and

$$\begin{aligned} Q_\pi(s,a) =& \mathbb{E}_\pi\left[(R_{k+1} - \rho^\pi) + \sum_{k=1}^{\infty}(R_{t+k+1} - \rho^\pi) \mid \mathcal{S}_t = s, \mathcal{A}_t = a\right] \\ =& \mathbb{E}_\pi[(R_{k+1} - \rho^\pi) + Q_\pi(S_{t+1}, A_{t+1}) \mid \mathcal{S}_t = s, \mathcal{A}_t = a] \end{aligned} \tag{4.22}$$

The relationship between the state-value function $V_\pi(s)$ and the action-value function $Q_\pi(s,a)$ under the average reward formulation is still expressed as in equation 4.10

4.2 Dynamic programming

Dynamic programming (DP) is a natural way of solving the standard MDP, where both the states and actions are finite and discrete, and where the transitions, the reward, and the value functions are assumed to store values for states and actions separately. Dynamic programming is ideal for discovering the optimal policy in a perfect model hence it is regarded as a model-based technique. The DP solves the MDP through a policy iteration or value iteration [36].

4.2.1 Policy iteration

A policy iteration is a two-staged process: a policy evaluation or prediction and policy improvement. The first process, called the policy evaluation, predicts the state-value function from an arbitrary policy. The process starts by initialising the value function v_0 to an arbitrary value. The process utilises the state-value Bellman equation as an update rule to approximate the next value functions in a series until the value function converges to the desired value [37].

By replacing V_π with V_k, in the state-value Bellman equation 4.7, V_{k+1} becomes

$$V_{k+1}(s) = \mathbb{E}_\pi[R_{t+1} + \gamma V_k(\mathcal{S}_{t+1}) \mid \mathcal{S}_t = s] \tag{4.23}$$

The question that always arises following the policy evaluation, is whether to apply the selected policy to the end or to enhance the policy at each state. A way to address this problem is through a selection of action a while at state s under the new greedy policy π' and compute

action-value function $Q_\pi(s,a)$. The greedy policy π' is defined as

$$\pi'(s) = \arg\max_a Q_\pi(s,a) \tag{4.24}$$

Once the policy π has been evaluated and improved to optimal policy π' using state-value function V_π , the successive policy π' can also be improved to π'' using state-value function $V_{\pi'}$, and so forth.

The whole policy iteration algorithm is shown in Algorithm 3 below.

Algorithm 3 : Policy iteration pseudo-code

```
1:  procedure Policy_iteration
2:      V_0(s) ← 0    ∀s ∈ S
3:      V_1(s) ← 0    ∀s ∈ S    ∀a ∈ A
4:      t ← 1
5:      while ∃s ∈ S such that |V_t(s) − V_{t−1}(s)| > ε do
6:          for all s ∈ S do
7:              for all a ∈ A do
8:                  Q(s,a) ← R(s,a) + γ Σ_{s'∈S} P(s,a,s') V_{t−1}(s')
9:              end for
10:             V_t(s) ← max_a Q(s,a)
11:             π(s) ← arg max_a Q(s,a)
12:         end for
13:         t ← t + 1
14:     end while
15: end procedure
```

4.2.2 Value iteration

One drawback of the policy iteration is that every iteration involves policy evaluation and improvement at each iteration, which may increase computational complexity. The value iteration, on the other hand, alleviates this problem by evaluating the policy once at the beginning of the iteration. The policy improvement and a single sweep of policy evaluation under value iteration are combined. Otherwise, the value function is similar to policy evaluation update rule except that it requires the maximum to take over all actions [31]. Algorithm 4 below illustrates a value

iteration Algorithm.

Algorithm 4 : Value iteration pseudo-code

```
1: procedure Value_iteration
2:     Initialise V(s) ∈ R for arbitrary ∀s ∈ S
3:     repeat
4:         for all s ∈ S do
5:             for all a ∈ A do
6:                 Q(s,a) ← R(s,a) + γ Σ_{s'∈S} P(s,a,s') V(s')
7:             end for
8:             V(s) ← max_a Q(s,a)
9:         end for
10:    until convergence
11: end procedure
```

4.3 Model-free RL techniques

The classic DP algorithms are ideal for the finite state and action spaces with complete knowledge of the environment, such as transition probability and rewards. In contrast, the majority of real-world problems exhibit continuous states and actions, and the dynamics are frequently unknown, making DP an infeasible option. As a result, the RL algorithms have been widely used to alleviate the DP shortfalls. The RL algorithms are regarded as computational techniques for solving MDPs by directly interacting with the environment, for which a model is unavailable [30].

4.3.1 Monte-Carlo

Monte-Carlo is a model-free algorithm that discovers action-value functions directly from episodes of experience. The policy under MC is evaluated by following an arbitrary policy from any state to the end of the episode. The discounted rewards are then computed for each step passed, and finally averaged at each state. The two types of MC are first visit and every visit. The first MC averages the returns only for the first time visit in the episode. Meanwhile, every time MC averages the returns at every visit. Contrary to the DP, where the policy selected is improved by acting greedily, the policy in MC is improved by applying epsilon greedy policy, that is, the

best-known policy on the experience is chosen with probability $(1 - \varepsilon)$, and other policies with probability ε.

4.3.2 Temporal-difference

Temporal difference (TD) is another model-free technique that combines the MC and DP. Just like MC, TD learns policy π directly from the raw experience without complete knowledge of the model. Like DP, the value functions are updated continuously at every time-step contrary to every episode. The TD algorithm can fall under a Q-learning or State-Action-Reward-State-Action (SARSA). The Q-learning is an off-policy that uses a greedy approach, meaning the algorithm follows a different exploration policy from the one it is estimating [38, 39]. The update rule for Q-learning is

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \tag{4.25}$$

On the other side, the SARSA is an on-policy TD control method that learns the action-value pair from a transition of state-action pair to another for the policy it is following [30, 40]. It uses the same standard policy iteration, except that the policy is evaluated using TD. The formula

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \tag{4.26}$$

is used as an update rule for all non-terminal states to produce $(s_t, a_t, r_t, s_{t+1}, a_{t+1})$. For the terminal state $Q(s_{t+1}, a_{t+1}) = 0$.

4.3.3 Actor-critic

Actor-critic algorithms learn both the policy and action-value function simultaneously, where the action-value is used for bootstrapping. The actor-critic algorithm possesses two different

memory structures. The actor structure, known as an actor, select the actions, whilst the value function, referred to as critic, evaluates the selected actions from the actor. The actor-critic is an on-policy, that is, the critic should always scrutinise the policy that is being followed by the actor. The algorithms work in that, after every action selection, the new state is evaluated and the new value is compared to the old to find out if the algorithm has improved. The difference between the current and the subsequent state value is defined:

$$\delta_t = R_{t+1} + \gamma V_t(s_{t+1}) - V(s_t) \tag{4.27}$$

Figure 4.2 shows the actor-critic architecture, where the critic estimates the value function as either the action value or state value. Conversely, the actor discovers the policy in the direction advocated by the critic.

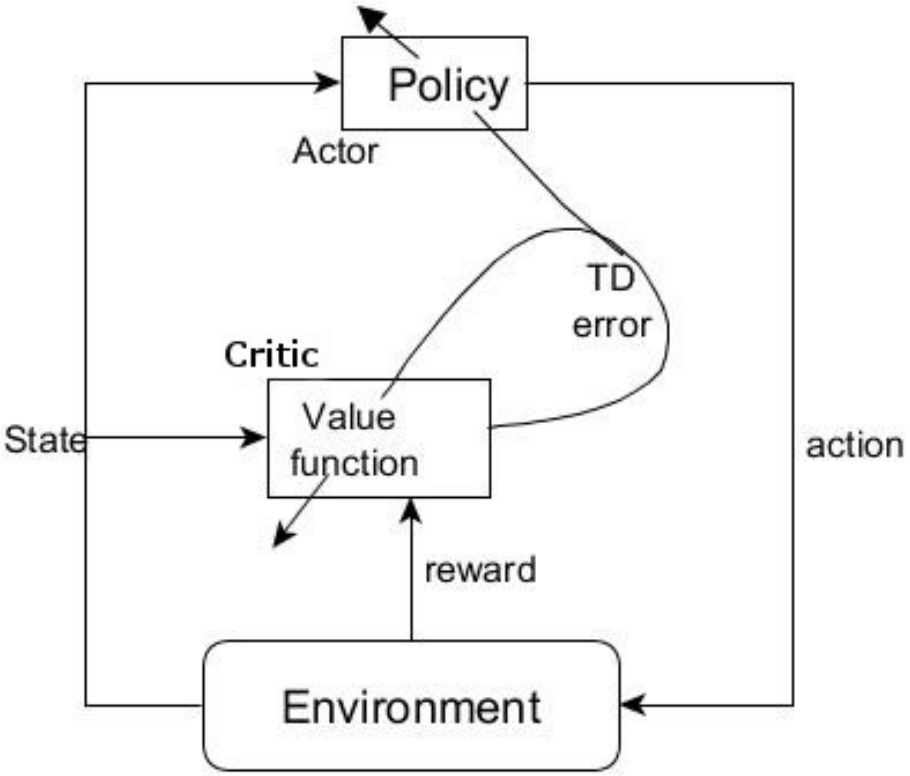


Figure 4.2: The actor-critic architecture

4.3.4 Policy gradient

The goal of a policy-based RL agent is to ascertain optimal actions to perform at each state that maximises the total rewards received from the environment in response to its actions. This technique is regarded as model-free because the agent is not required have full knowledge of the model. The policy-based RL is a three-staged process. The process of policy gradient involves defining the objective function that assesses the effectiveness of the policy, in a nutshell, the function that tells how good the results given by the policy. The aim is to adjust the parameter vector θ to maximise the objective function $J(\theta)$ as the total discounted reward. The standard

approach for optimising the $J_\theta(s)$ is to use stochastic gradient descent (SGD) whereby

$$J(\theta) \doteq V_{\pi\theta}(s_0) \tag{4.28}$$

and where $V_{\pi\theta}$ is the state-value function for policy π_θ, and s_0 is an initial state. Equation 4.28 postulates that when maximising $J(\theta)$, the $V_{\pi\theta}(s)$ is also maximised. As a result

$$\nabla J(\theta) = \nabla V_{\pi\theta}(s_0) \tag{4.29}$$

The policy gradient theorem suggests that

$$\begin{aligned} \nabla J(\theta) &\propto \sum_s \mathcal{P}(s) \sum_a Q_\pi(s,a) \nabla \pi(a|s,\theta) \\ &= \sum_s \mathcal{P}(s) \sum_a \pi(a|s,\theta) Q_\pi(s,a) \frac{\nabla \pi(a|s,\theta)}{\pi(a|s,\theta)} \\ &= \mathbb{E}_\pi[Q_\pi(s,a) \nabla \ln \pi(a|s,\theta)] \end{aligned} \tag{4.30}$$

where $\nabla \pi(a|s,\theta)$ is the gradient of the policy π on the state s and parameter vector θ and

$$\nabla_\theta \pi_\theta(s,a) = \pi_\theta(s,a) \nabla_\theta \log \pi_\theta(s,a) \tag{4.31}$$

where $\nabla_\theta \log \pi_\theta(s,a)$ is called the score function. To update parameter vector θ for optimisation to take place:

$$\nabla \theta = \alpha \nabla_\theta J(\theta) \tag{4.32}$$

Generally, there are two standard types of gradient policies used to learn parameter vector θ: softmax and Gaussian policy.

Softmax policy is usually used for discrete action space and it is made up of a softmax

function to transform the output to a probability distribution.

$$\pi_\theta(s,a) = softmax = \frac{\exp(\varphi(s,a)^{T_\theta})}{\sum_{k=1}^{N}\exp(\varphi(s,a_k)^{T_\theta})} \tag{4.33}$$

taking the log and the gradient, it follows that

$$\nabla_\theta \log \pi_\theta(s,a) = \varphi(s,a) - \mathbb{E}_{\pi\theta}[\varphi(s,\cdot)] \tag{4.34}$$

where $\varphi(s,a)$ is a feature vector.

On the other hand, for continuous actions, the Gaussian policy is used as

$$\pi_\theta(s,a) = \frac{1}{\sqrt{2\pi\sigma}}\exp\left(-\frac{(a-\mu)^2}{2\sigma^2}\right) \tag{4.35}$$

which then, by taking log and gradient, reduces to

$$\nabla_\theta \log \pi_\theta(s,a) = \frac{(a-\mu(s))\varphi(s)}{\sigma} \tag{4.36}$$

where σ is the standard deviation, $\varphi(s)$ is the feature vector, $\mu(s)$ is the mean.

Reinforce

Reinforce (MC policy gradient) estimates returns based on MC methods through episode samples to update the policy parameter θ. The algorithm assumes sample gradient expectation to be equal to the actual gradient, and because $Q_\pi(S_t,A_t) = \mathbb{E}_\pi[G_t|S_t,A_t]$. By replacing a and s with the sample A_t, and S_t it follows:

$$\begin{aligned}\nabla J(\theta) =& \mathbb{E}_\pi[Q_\pi(s,a)\nabla \ln \pi(a|s,\theta)] \\ =& \mathbb{E}_\pi[G_t \nabla \ln \pi(A_t|S_t,\theta)]\end{aligned} \tag{4.37}$$

In this formulation G_t is measured right from the sample trajectories of which the policy gradient is updated. The general pseudo-code of the MC policy gradient is given in Algorithm 5 below.

Algorithm 5 : MC policy gradient pseudo-code

```
1: procedure MC_policy_gradient
2:     Initialise the policy parameter θ
3:     for each episode do
4:         Generate trajectory S_0, A_0, R_1, ..., S_{T-1}, A_{T-1}, R_T following policy π
5:         for each step of episode do
6:             Estimate the return G_t
7:             Update policy parameter: θ ← θ + αγ_t G_t ∇ ln π_{θ(A_t|S_t)}
8:         end for
9:     end for
10: end procedure
```

4.3.5 Deep learning

Deep learning is a subfield of machine learning in artificial intelligence (AI) that mimic the way human brain process data in generating patterns for use in decision making. By contrast with the classical neural network which contains up to three hidden layers, deep learning uses more hidden layers.

The basis neural network as illustrated in Figure 4.3 consists of:

- input layer
- weights - represent importance of each input
- one or more hidden layers
- transfer function - combines multiple inputs into single value so that activation function can be applied,
- the activation function - transforms transfer (net input) function value into the value that

represents the inputs. Activation function can either be softmax, sigmoid function or rectified linear unit (ReLu) depending on the need of the designer

- one or more output - represents the output generated by the network from the inputs.

Deep learning algorithms are trained on large data to progressively extract features without human intervention. The five major network architectures that have been proven to be effective include, among others, recurrent neural networks (RNNs), convolutional neural networks (CNNs), long short-term memory (LSTM), deep belief networks (DBNs) and deep staking networks (DSNs). Figure 4.4 demonstrates a convolutional neural network consists of one input layer, multiple hidden layers and the output layer.

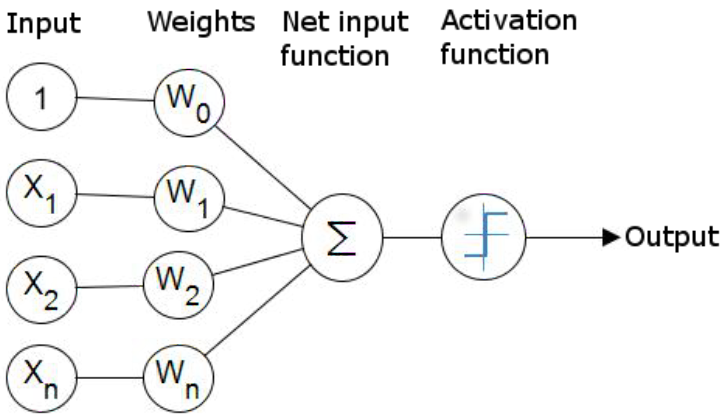


Figure 4.3: The simple neural network

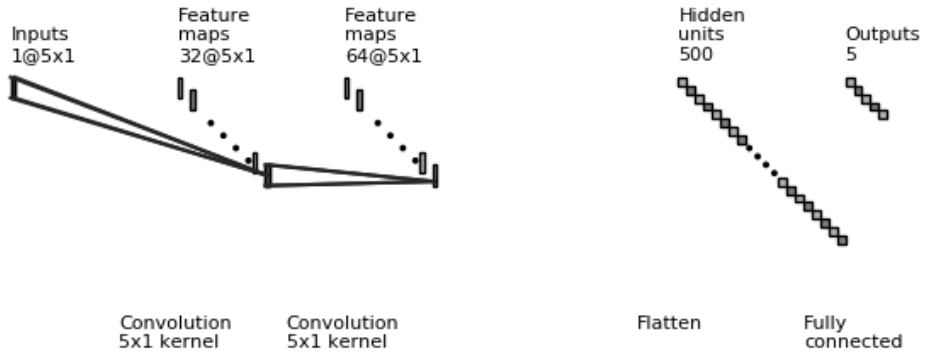


Figure 4.4: Convolutional neural networks

4.3.6 Deep deterministic policy gradient

The deep deterministic policy gradient (DDPG) is a model-free, off-policy, actor-critic algorithm that combines the principles of deterministic policy gradient (DPG) with deep Q-learning (DQN) in a continuous action space [41]. The DDPG uses four neural networks —the Q-value, deterministic policy function, target Q-value and target policy network. It is one type of actor-critic where the output of states maps directly to action instead of the probability distribution. The use of the target networks stabilise the learning. Like DQN, the DDPG employs experience replay to discover the optimal policy in the continuous action spaces rather than discrete action spaces [42]. The algorithm stores experience tuple of (state, action, reward, next state) into the finite reply buffer, and later sample mini-batch of experience from the replay buffer to update the Q-value and policy networks. The Q-value is updated similar to the DQN through the Bellman equation. However, the next Q-value is computed from the target Q-value and policy networks. The mean-square loss between the Q-value and the original value is minimised. The original value is calculated from the value networks instead of target networks. Refer to Algorithm 6

below for the DDPG pseudo-code.

$$Loss = \frac{1}{N}\sum_{i}(y_i - Q(s_i, a_i|\theta^Q))^2 \tag{4.38}$$

Algorithm 6 : DDPG pseudo-code

procedure DDPG
  Randomly initialise critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$
  Initialise target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q, \theta^\mu \leftarrow \theta^{\mu'}$
  Initialise replay buffer R
  for episode = 1,M **do**
    Initial a random process $\mathcal{N}$ for action exploration
    Receive initial observation state s_1
    for t = 1,T **do**
      select action $a_t = \mu(s_t|\theta^m u + \mathcal{N}_t)$ according to the current policy
      and exploration noise
      Store transition s_t, a_t, r_t, s_{t+1} in R
      Sample a random mini-batch of $\mathcal{N}$ transition s_t, a_t, r_t, s_{t+1} from R
      Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
      Update critic by minimising the loss: $L = \frac{1}{N}\sum_{i}(y_i - Q(s_i, a_i|\theta^Q))^2$
      Update the actor policy using the sampled gradient:
        $\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_{i} \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s_i}$
      Update the target networks:
        $\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$
        $\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$
    end for
  end for
end procedure

4.4 Brief chapter summary

Reinforcement learning is one of the machine learning algorithms concerned with learning from interaction to attain a reward. Unlike supervised learning, the RL is not equipped with output labels. The RL problem is best formulated in terms of the MDP where the decision-maker called the agent makes successive observations of the environment and executes a series of actions known as policy before taking the final decision. In return, the agent obtains the reward from the

environment as a result of its actions. The goal of the agent is to optimise the cumulative reward which is formulated as either discounted or averaged. The agent learns the policy directly or through value function from which it can act greedy to receive the optimal policy. For the perfect environment, where there is full knowledge of the environment, dynamic programming appears to be the algorithm of choice. Alternately, for imperfect environments, the model-free algorithms such as MC, temporal difference and actor-critic methods are used. In the continuous state and action spaces, policy gradient algorithms such as DPG and its extended DDPG are used. With the policy gradient algorithms instead of the agent following the policy directly, it follows the policy gradient. For this research, the DDPG is used to solve the asset allocation problem using convolutional neural network.

5

Numerical experiments

In this chapter numerical experiences of the learning algorithms is discussed with the background of chapters 3 and 4 as applied to the asset allocation problem and trading strategy. The chapter also describes the evaluation function as the modified Sharpe ratio with self-financing budget constraint used in all algorithms. Finally, the chapter highlights the performance of the three algorithms compared to one another, and then with the benchmarks.

5.0.1 Evaluation function

During the trading period for stock markets, prices are bound to go up and down. However, only four movements are regarded as significant particularly, opening, lowest, highest and closing

prices. For continuous markets, like stock markets, the closing prices of the previous period are considered to be the opening of the subsequent period. A portfolio manager is tasked with periodically adjusting portfolio weights by buying and selling relevant stocks. There are costs associated with every transaction made. The two types of transaction costs are fixed and variable costs.

Fixed costs are costs independent of the amount traded, which include commission and taxes, whereas variable costs depend entirely on the trading volumes. As in [43], transaction cost $C(w)$ incurred in trading of relevant assets is:

$$C(w) = \sum_{i=1}^{N} C_i(w_i(t)) \tag{5.1}$$

If portfolio weights $w(t-1)$ in the period t before transaction costs are deducted and portfolio weights after transaction cost is $w(t)$ in the same period t, then the total transaction costs incurred is

$$\sum_{i=1}^{N} \mu_i(|w_i(t) - w_i(t-1)|) \tag{5.2}$$

Where the transaction cost on i^{th} asset is calculated as:

$$C_i(w_i(t)) = \left\{ \begin{array}{ll} d_i^b(w_i), & \text{if} \quad w_i(t) > 0 \\ d_i^s(w_i), & \text{if} \quad w_i(t) < 0 \end{array} \right\}$$

d_i^b and d_i^s are transaction costs on buying and selling of i^{th} asset respectively. The above formulation depicts that the i^{th} asset is sold when $w_i(t) < 0$, and bought when $w_i(t) > 0$.

One other important constraint to consider is a budget constraint that ensures all funds

available are invested in the portfolio, and is formulated as:

$$\sum_{i=1}^{N} w_i(t) = 1 \tag{5.3}$$

To avoid extra cash inflow and outflow, self-financing budget constraint is introduced by combining budget and transaction costs. The sum of portfolio weights and the transaction costs must sum to a unit as:

$$\sum_{i=1}^{N} w_i(t) + \sum_{i=1}^{N} C_i(w_i(t)) = 1 \tag{5.4}$$

Therefore the final formulation employed in this dissertation is as follows:

- Maximise modified portfolio Sharpe ratio:

$$S_p = \frac{(\mu_p - r_f)}{mVaR_p} \tag{5.5}$$

Subject to

- Self-financing budget constraint:

$$\sum_{i=1}^{N} w_i(t) + \sum_{i=1}^{N} C_i(w_i(t)) = 1 \tag{5.6}$$

In this formulation, the benchmark is incorporated into the modified Sharpe ratio because the prime objective of the Sharpe ratio is to measure the excess portfolio returns over the risk-free rate related to its VaR. Normally, 90 days treasury bill rate is taken as the proxy for a risk-free rate never-the-less this dissertation uses stock index as the benchmark simple because the stock index is readily available on daily frequencies and it does not need any conversion

from interest rate to daily frequency. Ideally, any Sharpe ratio higher than 1.0 is acceptable by investors, a ratio greater than 2.0 is considered very good and the ratio higher than 3.0 is rated excellent. However, to achieve a higher Sharpe ratio, some factors contribute, among others, diversification in security selection. The focal point around security selection is to select assets from different assets classes that have a low to negative correlations so that if one moves down the other moves up in the formation of portfolios. These classes can be stocks, cash, bonds and mutual funds. The current work does not touch on the process of security selection, but it focuses mainly on finding an optimum portfolio by adjusting assets weights. Generally, stocks have slightly lower Sharpe ratio than other assets classes like bonds. This study does not attempt to provide the best Sharpe ratio but compares the performance on each learning algorithm in maximising Sharpe ratio for the given portfolios so any Sharpe ratio higher than 0 is acceptable.

5.1 Data extraction and pre-processing

For the empirical part, 8 years annualised daily closing prices for the period between 2010-01-01 and 2018-12-31 (currency ZAR and USD) from the South African and United States markets extracted from JSE and US stock exchanges respectively are considered. For the JSE, the 5 leading banks companies are considered, namely Standard Bank, First National Bank (FNB), Netbank, Capitex and Amalgamated Banks of South Africa (ABSA) and benchmarked with the FTSE/JSE Top 40 index in the testing phase. Likewise Amazon, Microsoft, Apple, Google and Select Sector fund are used for the US stock exchanges benchmarked with the S&P 500 index.

Only weekdays are included in the dataset because there is no trading happening on weekends. Alternatively, some weekdays are public holidays, in which case, no prices are available, and for this reason, the missing prices are discarded. The dataset is divided into two parts; the training phase in the range 2010-01-01 to 2015-12-31 on the other hand testing dataset is

between 2016-01-01 and 2018-12-31.

5.2 Dataset visualisation

Data visualisation is an integral aspect, especially when dealing with big data like time series. The idea is to gain valuable insight into the data one is working on. Figures 5.1 to 5.3 below show the individual price movements for the South African stocks and their movements relative to one another to have a clear understanding. It is evident from the figures that the price movements of almost all stocks tend to go up for the 8-year period.

Consequently, plotting stock returns instead of the daily prices on Figure 5.4, one can have an overview of stocks volatility. Most of the stocks have negative and positive spikes signalling their riskiness with Standard Bank being the most stable amongst all stocks.

Table 5.1 depicts the overall statistical summary for the 8 year period for the SA dataset. It is apparent from the table looking at the skewness and kurtosis that the data distribution is asymmetric. All stocks have positive skewness apart from FTSE Top 40 which has negative skewness. On the side of kurtosis, all stocks possess negative kurtosis with the exception of Stdbank with a positive kurtosis.

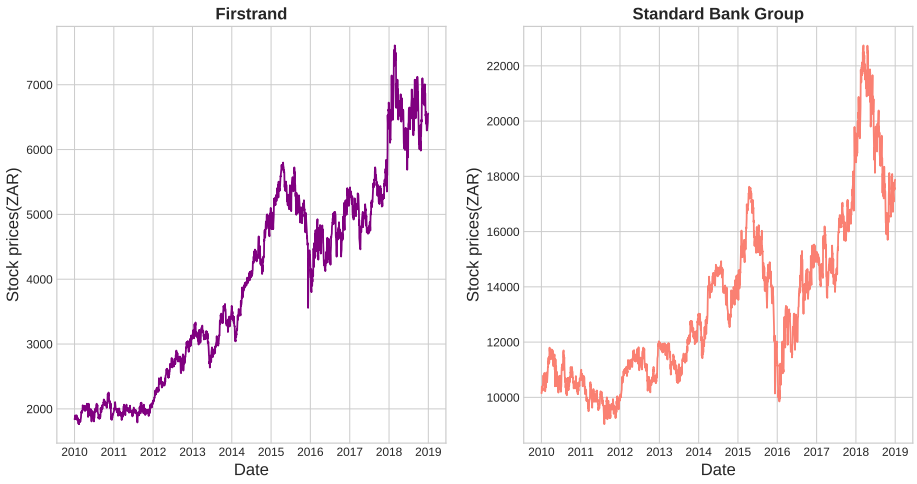


Figure 5.1: South African Stocks

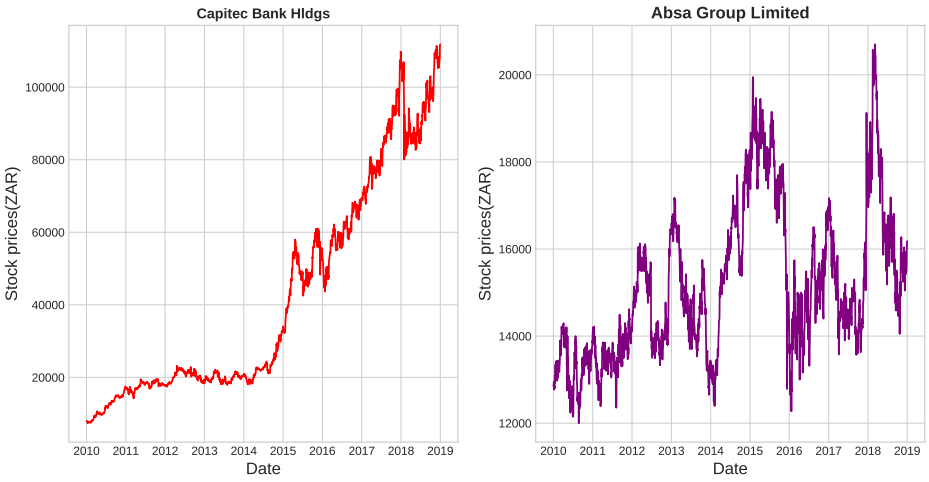


Figure 5.2: South African Stocks

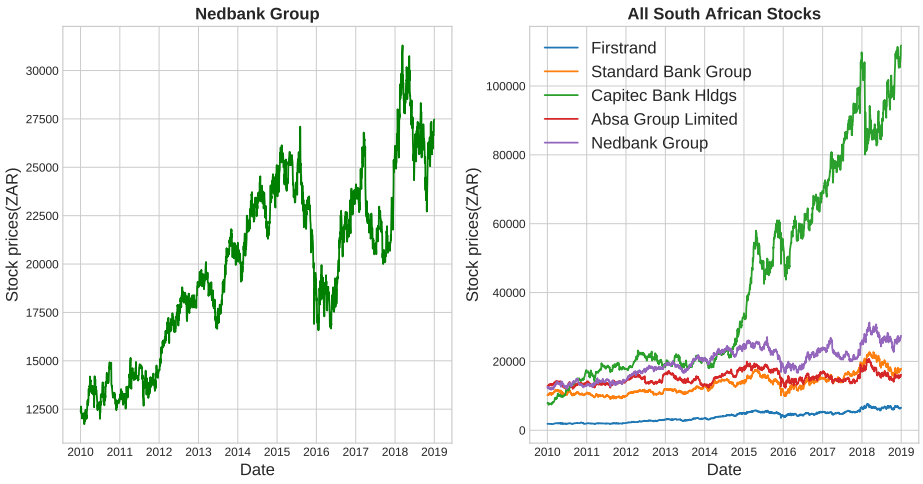


Figure 5.3: South African Stocks

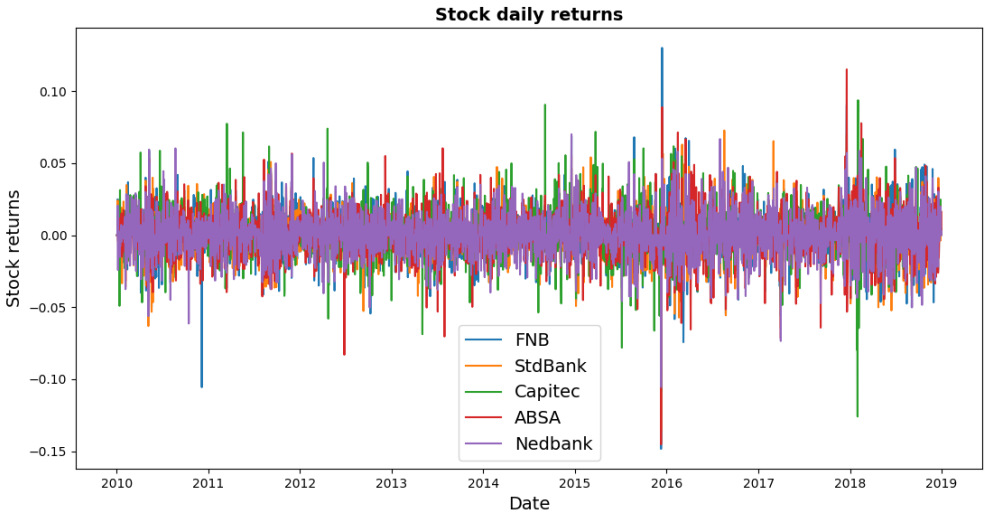


Figure 5.4: Visualisation of stock daily returns

The same procedure is followed on the FTSE/JSE Top 40 benchmark for the price movement and the return presented on Figures 5.5 and 5.11 respectively. Moreover, Table 5.2 further outlines the statistical summary of all stocks.

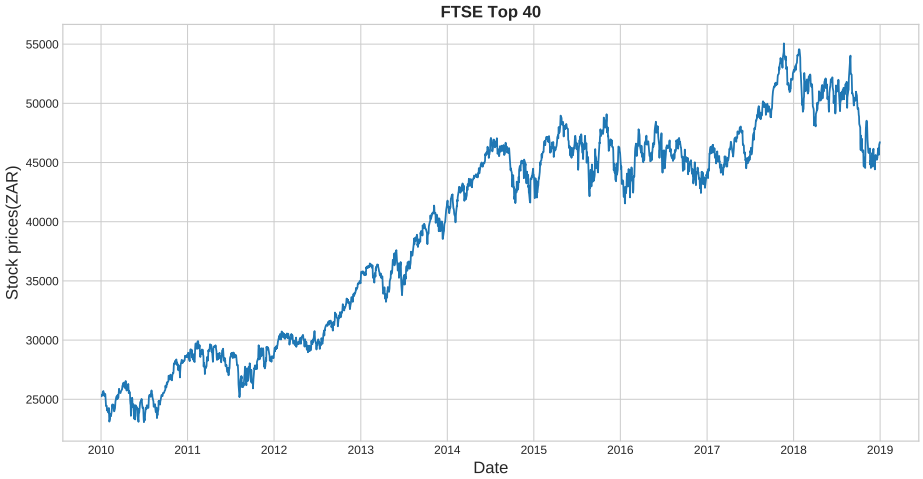


Figure 5.5: FTSE/JSE Top 40 to benchmark South African stock

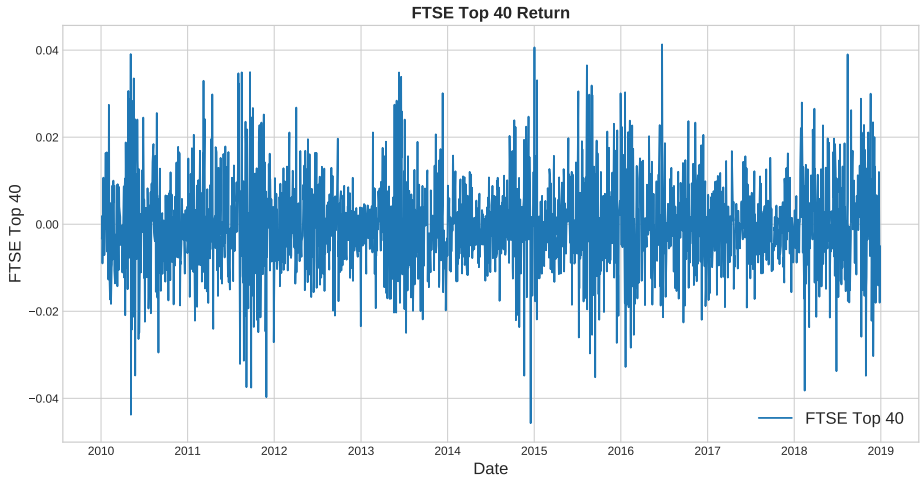


Figure 5.6: Visualisation of FTSE Top 40 daily returns

Table 5.1: The overall statistical summary of the 8 year period for the JSE dataset with the FTSE/JSE Top 40 benchmark

	FTSE Top 40	FNB	StdBank	Capitec	Absa	Nedbank
Count	2248.0	2251.0	2251.0	2251.0	2251.0	2251.0
Mean	39495.6	3933.2	13330.0	42001.9	15168.3	19886.7
Std	8904.9	1553.3	3016.3	29635.2	1739.7	4552.8
Min	23066.7	1764.0	9029.0	7369.0	12000.0	11725.0
25%	29852.87	2403.0	10900.0	18698.5	13793.5	16590.5
50%	43217.95	4030.0	12516.0	22452.0	14838.0	20230.0
75%	46358.1	5154.5	15039.0	63762.0	16250.0	23366.0
Max	55065.4	7607.0	22741.0	111800.0	20700.0	31300.0
Skewness	-0.326213	0.220056	0.945921	0.742331	0.718362	0.019618
Kurtosis	-1.328439	-1.105186	0.316576	-0.866260	-0.146453	-0.92558844

Alternately, Figures 5.7 through 5.10 show stock movements for the US dataset used in this dissertation namely, Select Sector fund (XLK), Microsoft (MSFT), Apple (AAPL), Google (GOOG), Amazon (AMZN) and S&P 500 index as the benchmark.

Table 5.2 shows the overall statistical summary for the 8 years for the US dataset. It is also clear from the table looking at the skewness and kurtosis that the data distribution is asymmetric. All stocks have positive skewness and all stocks possess negative and positive kurtosises.

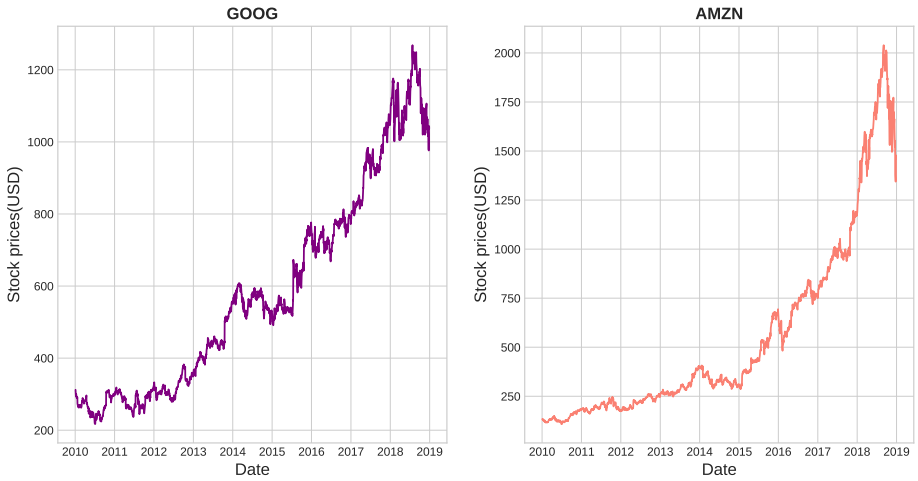


Figure 5.7: US Stocks

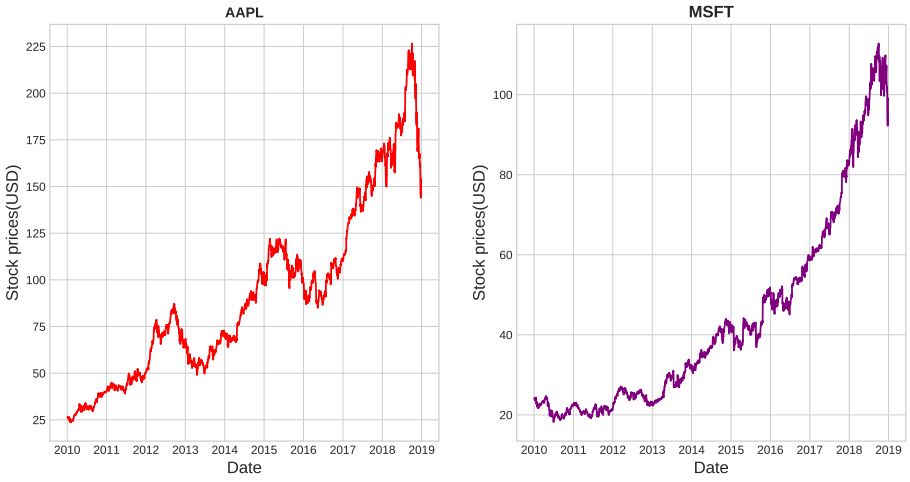


Figure 5.8: US Stocks

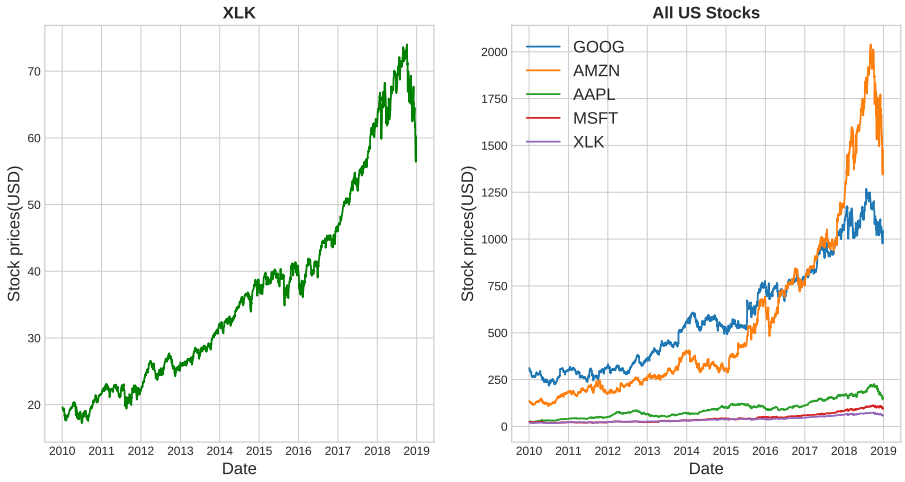


Figure 5.9: US Stocks

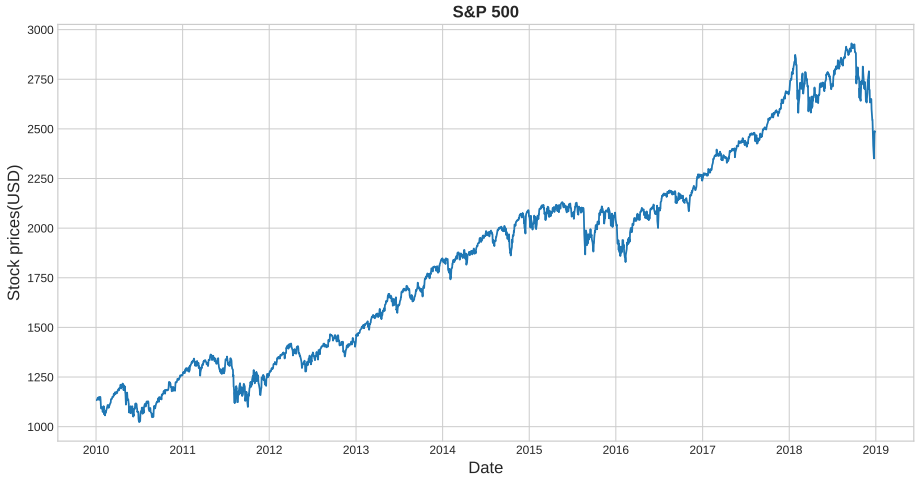


Figure 5.10: S&P 500 to benchmark US stock

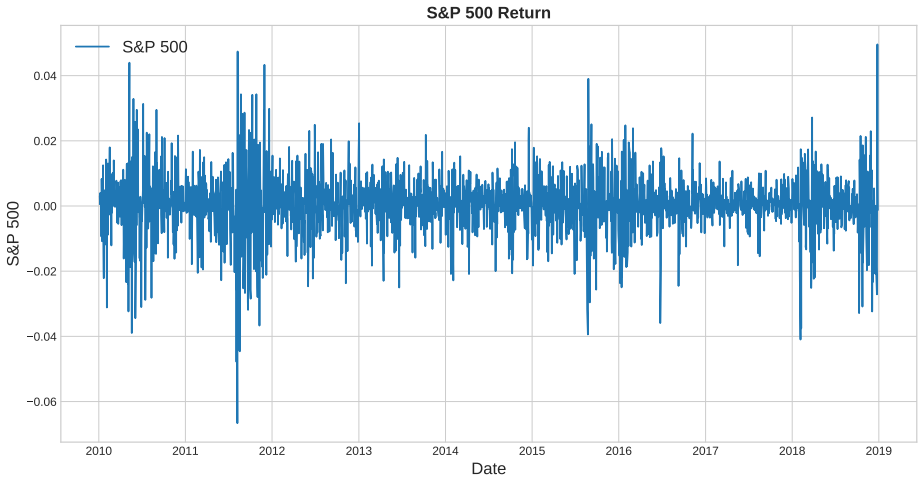


Figure 5.11: Visualisation of S&P 500 daily returns

Table 5.2: The overall statistical summary of the 8 year period for the US dataset with the S&P 500 benchmark

	S&P 500	GOOG	AMZN	AAPL	MSFT	XLK
count	2263	2263	2263	2263	2263	2263
mean	1856.5	583.12	551.90	92.11	43.64	36.86
std	520.57	283.58	468.72	47.09	24.67	15.08
min	1022.580017	217.22	108.61	23.75	18.18	17.22
25%	1352.204956	307.97	217.74	54.18	23.7	24.64
50%	1904.010010	537.36	332.85	85.79	36.78	34.66
75%	2168.38	772.36	766.9	115.84	54.02	44.62
max	2930.75	1268.33	2039.51	226.69	112.81	74.02
Skewness	0.217317	0.602594	1.429576	0.744082	1.172238	0.786398
Kurtosis	-1.043448	-0.778561	1.136871	-0.157029	0.403215	-0.39078111

5.3 Training and Experiment

The training dataset consists of the first 5 years from 2010-01-01 to 2015-12-31 and the remaining 3 years dataset from 2016-01-01 to 2018-12-31 is devoted to testing. The learning algorithms are trained and tested on the historical stock price series of the risky assets obtained from JSE and US stock exchanges with a benchmark of S&P 500 and FTSE/JSE Top 40 for the South African and US market respectively.

The stock prices fluctuate daily as a result, the stock closing prices are normalised to daily stock returns before they are exposed to the learning algorithms in order to measure the magnitude of the change. The positive change signifies growth on the value of the stock while the negative returns means the stock has lost value. The daily return measures the change in stock's price as a percentage of the closing price of the previous day.

The model consists of two instances: 3 stocks and 5 stocks portfolio. The 5 stock portfolio takes all stocks in the dataset while the 3 stocks portfolio considers only 3 stocks from the dataset which are applied on all algorithms. The 3 stocks instances for South Africa consist of FNB, Std Bank and Capitec while GOOG, AMZN and AAPL are selected for the 3 stocks instances for the US market. Initially, portfolio weights are randomly generated and exposed to

learning algorithms for evolution over 500 epochs/runs for all algorithms. All learning algorithms are run 5 times, and the results are averaged. The experiments are run on Ubuntu 20.04 LTS (focal), 12G RAM, Intel(R) Core(TM) i5-7200U CPU 2.50GHz Processor and Python 3.7.3.

5.3.1 Implementation of PSO and GA

This dissertation applies *gbest* PSO. The algorithm generates a swarm of particles with random positions and velocities. For the problem at hand, the positions represent candidate portfolios. The algorithm iteratively improves candidate portfolio with regard to the modified Sharpe ratio s_p defined in Equation 5.5 and classical Sharpe ratio for comparison purposes. Each particle is influenced by its best position, and the best position attained by the entire swarm. The termination criteria in this case are the maximum number of iterations. Parameter settings and the pseudo-code of the PSO are summarised in Table 5.3 and Algorithm 7 below respectively.

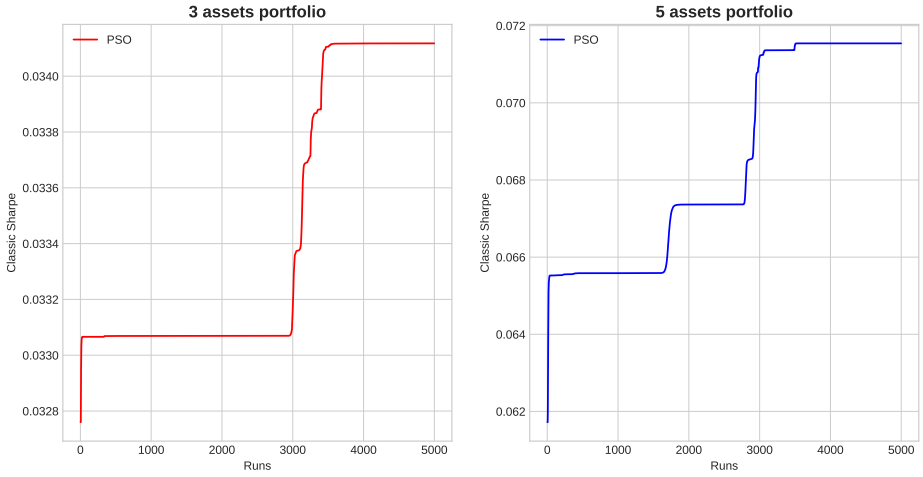
Table 5.3: Parameters for the particle swarm optimisation

Parameter	**Value**
Population size	100
c1	2.05
c2	2.05
w	0.729844
No of iterations	5000

Algorithm 7 : PSO pseudo-code implementation based on [44]

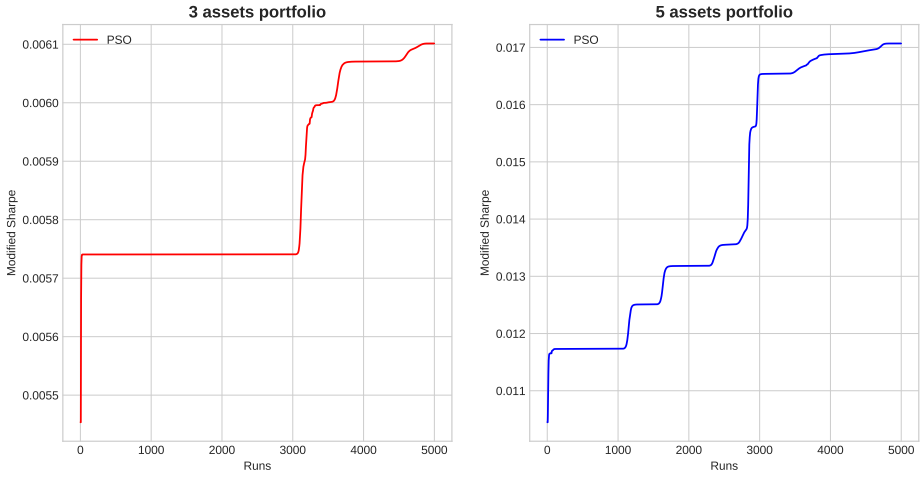
procedure PSO
  for each portfolio $i = 1, \ldots, S$ **do**
    Initialise the portfolio's initial position with a uniformly distributed random vector: $\mathrm{x}_i \sim U(\mathrm{b}_{\mathrm{lo}}, \mathrm{b}_{\mathrm{up}})$
    Initialise the portfolio's best known position to its current position: $\mathrm{p}_i \leftarrow \mathrm{x}_i$
    if sharpe(p_i) < sharpe(g) **then**
      update the swarm's best known position: $\mathrm{g} \leftarrow \mathrm{p}_i$
    end if
    Initialise the portfolio's velocity: v_i
  end for
  while termination condition not satisfied **do**
    for each portfolio **do**
      for each dimension d = 1, …,n **do**
        pick random numbers: r_p, r_g $simU(0, 1)$
        update the portfolio's velocity: $\mathrm{v}_{\mathrm{i,d}} \leftarrow \gamma \mathrm{v}_{i,d} + \varphi r_p(\mathrm{p}_{i,d)} - \mathrm{x}_{i,d}) + \varphi r_g(\mathrm{g}_{i,d)} - \mathrm{x}_{i,d})$
      end for
      update the portfolio's position: $\mathrm{x}_i \leftarrow \mathrm{x}_i + \mathrm{v}_i$
      initialise the portfolio's best known position to its current position: $\mathrm{p}_i \leftarrow \mathrm{x}_i$
      if sharpe(x_i) < sharpe(p_i) **then**
        update the swarm's best known position: $\mathrm{g} \leftarrow \mathrm{p}_i$
        if sharpe(p_i) < sharpe(x_i) **then**
          update the swarm's best known position: $\mathrm{g} \leftarrow \mathrm{p}_i$
        end if
      end if
    end for
  end while
  return g
end procedure

The results for PSO on the two datasets are given below. Figures 5.12a to 5.15b show the best performance for both the modified Sharpe ratio and the classic Sharpe ratio for each dataset. The figures are followed by Tables 5.4 to summarise the best outcome and the computation time for each dataset and scenario.



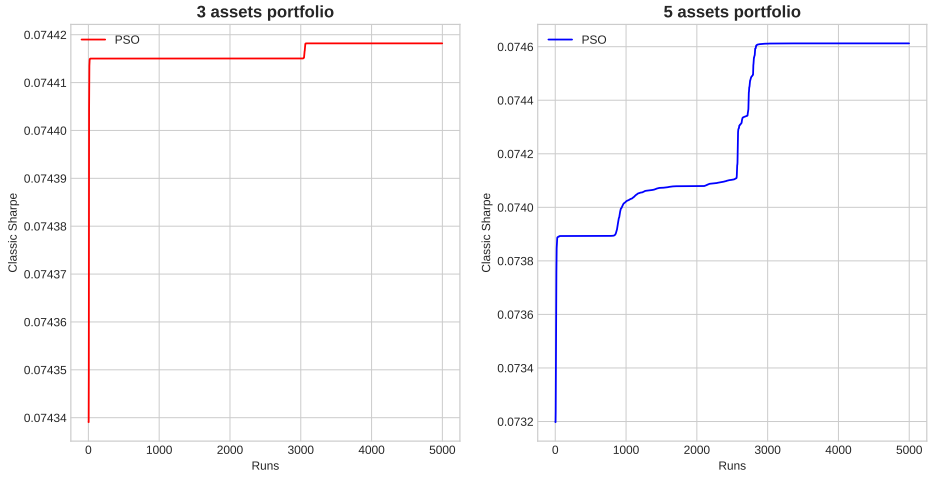
(a) PSO results for 3 stocks portfolio (b) PSO results for 5 stocks portfolio

Figure 5.12: Results of training PSO agent on 3 and 5 stocks portfolios on the South African dataset using the classic Sharpe ratio as evaluation function



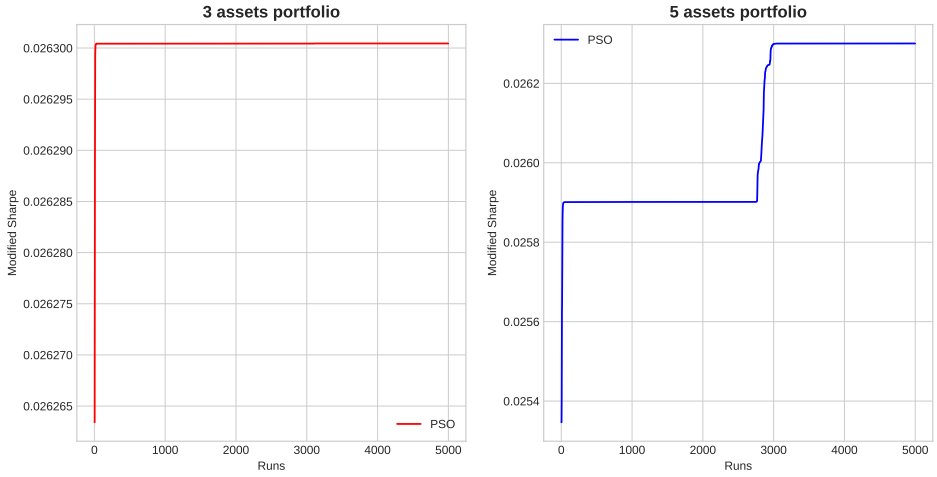
(a) PSO results for 3 stocks portfolio (b) PSO results for 5 stocks portfolio

Figure 5.13: Results of training PSO agent on 3 and 5 stocks portfolios on the South African dataset using the modified Sharpe ratio as evaluation function



(a) PSO results for 3 stocks portfolio (b) PSO results for 5 stocks portfolio

Figure 5.14: Results of training PSO agent on 3 and 5 stocks portfolios on the US dataset using the classic Sharpe ratio as evaluation function



(a) PSO results for 3 stocks portfolio (b) PSO results for 5 stocks portfolio

Figure 5.15: Results of training PSO agent on 3 and 5 stocks portfolios on the US dataset using the modified Sharpe ratio as evaluation function

Table 5.4: Summary of the best PSO results after 5000 training runs/iterations

Dataset	Evaluation function	Average time(s)	Optimum Sharpe ratio
SA 3 stocks port	classic Sharpe	1315.42	0.034118
	modified Sharpe	1327.49	0.00610
SA 5 stocks port	classic Sharpe	1421.18	0.07154
	modified Sharpe	1415.70	0.07107
US 3 stocks port	classic Sharpe	1319.74	0.07442
	modified Sharpe	1294.59	0.02630
US 5 stocks port	classic Sharpe	1434.86	0.07461
	modified Sharpe	1482.33	0.02630

The results of training PSO tend to evolve for all datasets and scenarios. One crucial observation on both datasets from all figures concerning the classical and modified Sharpe ratios are the learning patterns which do not change vigorously. The evidence on the pattern is shown in Figures 5.12a and 5.13a on comparing the classical and modified Sharpe ratio for the 3 stocks portfolios using the South African dataset. The same evidence is provided in Figures 5.12b and 5.13b on 5 stocks portfolio. By the same token, the learning pattern is also true for the US dataset on both portfolio instances as shown in Figures 5.14a and 5.15a for the 3 stocks portfolios and Figures 5.14b and 5.15b for the 5 stocks portfolios. The classic Sharpe ratios attain higher values on all scenarios than that of the modified Sharpe ratios however, both formulations cannot be compared as they measure volatility differently.

Likewise, the GA agent generates population of portfolios which are evaluated in respect to the modified Sharpe ratio s_p defined in Equation 5.5. The algorithm iteratively selects two portfolios from the mating pool based on their quality to generate offspring. If the currently generated offspring is better than any member of the population, such member is replaced by the newly generated offspring. The offspring is mutated with a certain probability to allow diversity. Parameter settings and the pseudo-code of the GA implementation are summarised in Table 5.5 and Algorithm 8 below respectively.

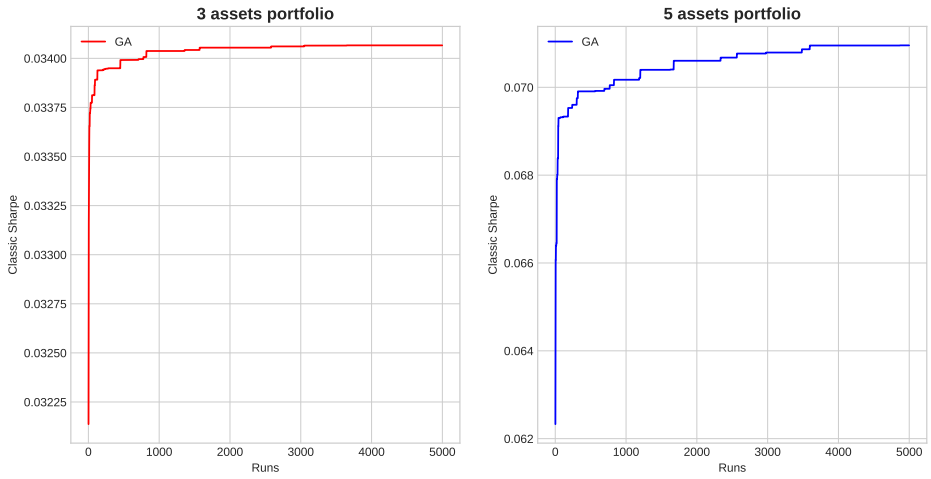
Table 5.5: Parameters for the genetic algorithm

Parameter	Value
Population size	100
Mutation rate	0.08
Crossover rate	0.78
No of iterations	5000

Algorithm 8 : GA pseudo-code implementation

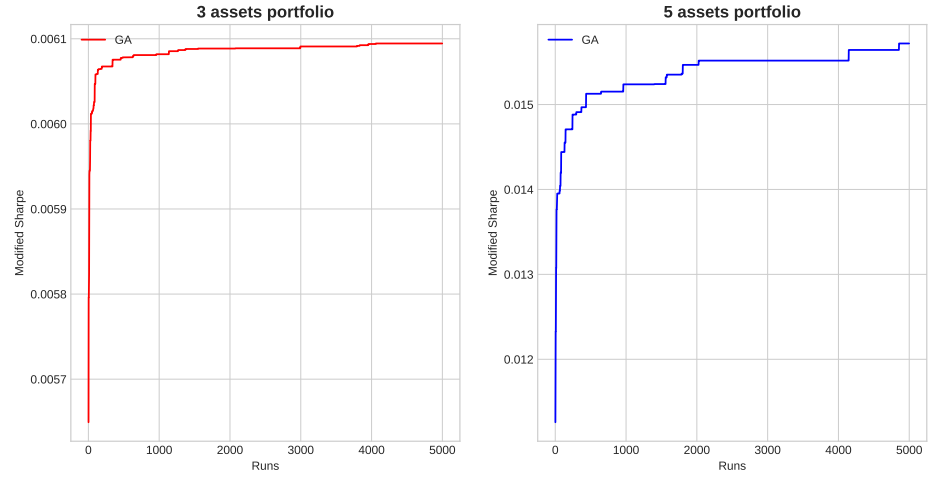
```
1:  procedure GA
2:      t = 0;
3:      Randomly initialise population of portfolios
4:      Evaluate each portfolio in the portfolio
5:      while max number of iterations do
6:          Select best-fit portfolios for reproduction
7:          Breed new offspring through crossover and mutation operations
8:          Evaluate offspring
9:          Replace least-fit population with the offspring
10:     end while
11:     return population
12: end procedure
```

The convergence results of training the GA agents on the 3 assets portfolio and 5 assets portfolio can be seen in Figures below. Like the PSO agent, the objective of the GA agent is to select the best performing portfolio in every run with respect to the modified Sharpe ratio and classic Sharpe ratio. Figures 5.16a to 5.17a present the best performance for both the modified Sharpe ratio and the classic Sharpe ratio for each dataset and scenario. The figures are succeeded by Tables 5.6 to recapitulate the best outcome and computation time for each dataset and scenario.



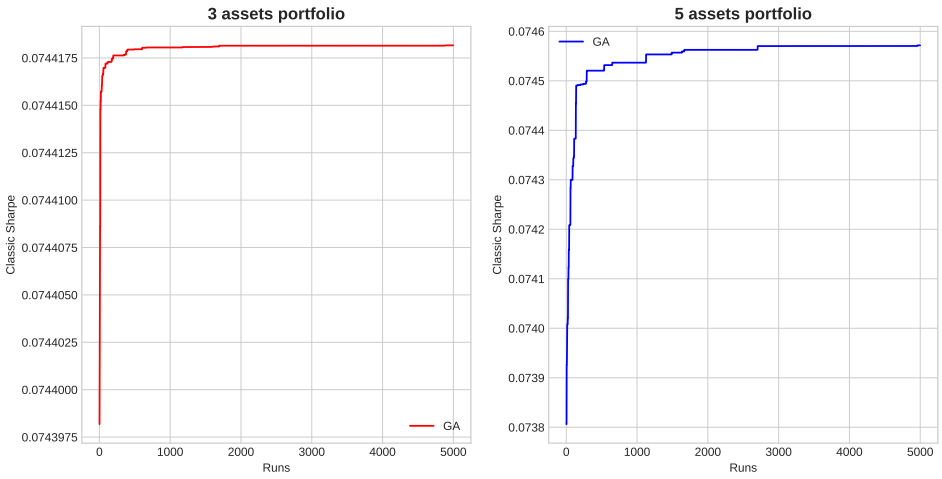
(a) GA results for 3 stocks portfolio (b) GA results for 5 stocks portfolio

Figure 5.16: Results of training GA agent on 3 and 5 stocks portfolios on the South African dataset using the classic Sharpe ratio as evaluation function



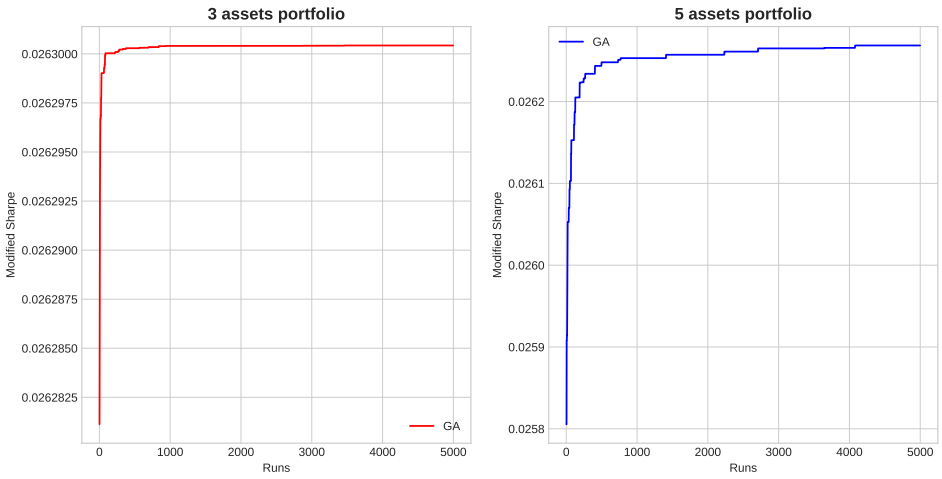
(a) GA results for 3 stocks portfolio (b) GA results for 5 stocks portfolio

Figure 5.17: Results of training GA agent on 3 and 5 stocks portfolios on the South African dataset using the modified Sharpe ratio as evaluation function



(a) GA results for 3 stocks portfolio (b) GA results for 5 stocks portfolio

Figure 5.18: Results of training GA agent on 3 and 5 stocks portfolios on the US dataset using the classic Sharpe ratio as evaluation function



(a) GA results for 3 stocks portfolio (b) GA results for 5 stocks portfolio

Figure 5.19: Results of training GA agent on 3 and 5 stocks portfolios on the US dataset using the modified Sharpe ratio as evaluation function

Table 5.6: Summary of the best GA results after 5000 training runs/iterations

Dataset	Evaluation function	Average time(s)	Optimum Sharpe ratio
SA 3 stocks port	classic Sharpe	3094.45	0.03407
	modified Sharpe	3182.74	0.00609
SA 5 stocks port	classic Sharpe	3197.03	0.07104
	modified Sharpe	3165.181	0.01594
US 3 stocks port	classic Sharpe	3073.92	0.07442
	modified Sharpe	3122.85	0.02630
US 5 stocks port	classic Sharpe	3122.37	0.07458
	modified Sharpe	3201.41	0.02630

Overall, the algorithm converges more rapidly within the first few training steps and indicates that the learning patterns look relatively similar across all portfolio instances and scenarios as demonstrated on Figures 5.16a and 5.19b. Furthermore, the findings from the table indicate that there is no significant impact on the number of stocks in a portfolio for the US dataset. On the contrary, the optimum value increases with the number of stock for the SA dataset. This evidence may suggest that choice of asset selection plays a fundamental part in the asset allocation. In terms of the average time that the algorithm takes to run, there is still no clear picture across instances (3 and 5 stocks portfolios). However, inclusion of too many variables on modified Sharpe ratio leads to the algorithm running longer than in the classic Sharpe.

5.3.2 Implementation of DDPG

The DRL employs the DDPG discussed in chapter 6 as follows:

- The portfolio vector ω_t is analogous with actions a_t.
- The set of states $s_t = (x_t, \omega_{t-1})$ is a function of two variables: the portfolio vector ω_{t-1} and the closing price mean returns x, as the only feature.

- The reward of the agent is to maximise the modified Sharpe ratio s_p defined in Equation 5.5.
- Each network consists of three hidden layers with rectified linear unit (ReLu) as activation function in between.
- There is one output layer of softmax as the activation function.
- The actor network accepts states as input and outputs the actions.
- The critic measures the quality of the generated actions from the actor network.
- The critic takes the actions from the actor and states as the input, and then outputs the action-value corresponding to modified Sharpe ratio values.

Figure 5.20 illustrates the DDPG framework applied in this dissertation.

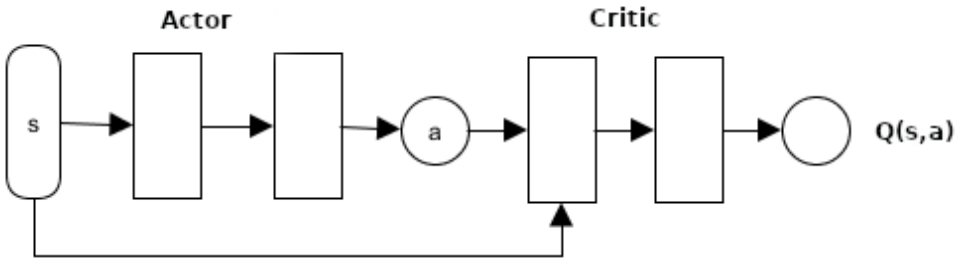
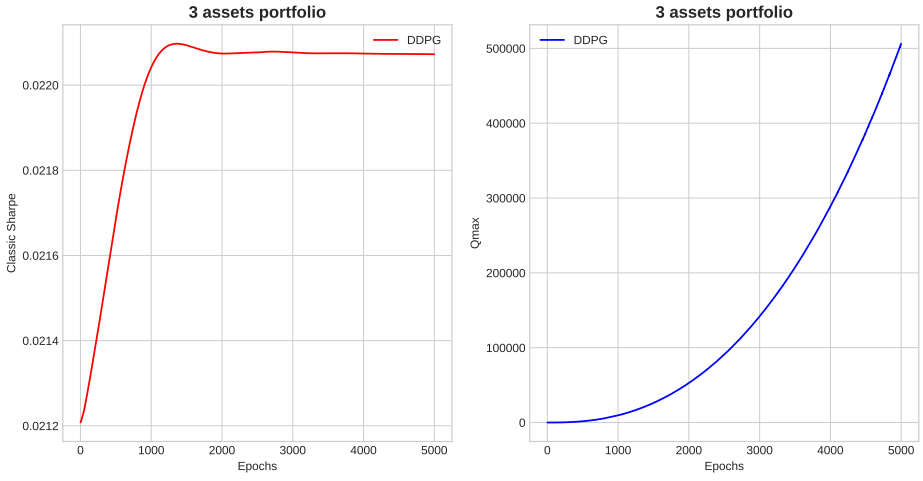


Figure 5.20: DDPG architecture

Table 5.7: Deep deterministic policy gradient parameters

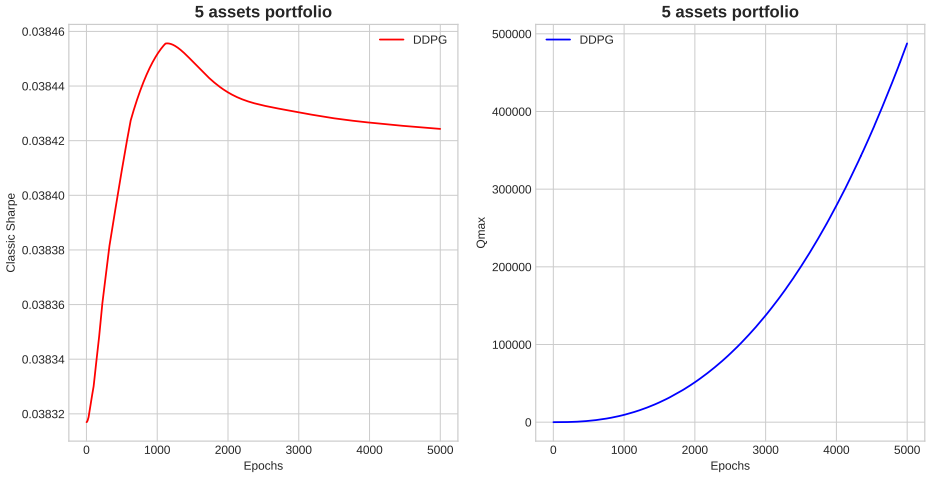
Parameter	Value
Input layer	mean returns
Actor network learning rate	0.001
Critic network learning rate	0.0001
Discount factor for critic updates	0.2
Soft target update parameter	0.001
Max size of the replay buffer	10
Size of minibatch for minibatch-SGD	3
Max num of episodes to do while training	5000
Max length of 1 episode	6

Figures 5.21 and 5.23a below depict the learning performance results of the DDPG agent on the 3 and 5 assets portfolios for both South African and US dataset respectively. The aim is to discover the action that maximise the total modified Sharpe ratio as well as the classic Sharpe as the reward. The algorithm utilises two neural networks (actor and critic). The actor predicts the action to be taken given the current state. The figures are followed by Tables 5.8 that summarises the best outcome and computation time for each dataset and scenario.



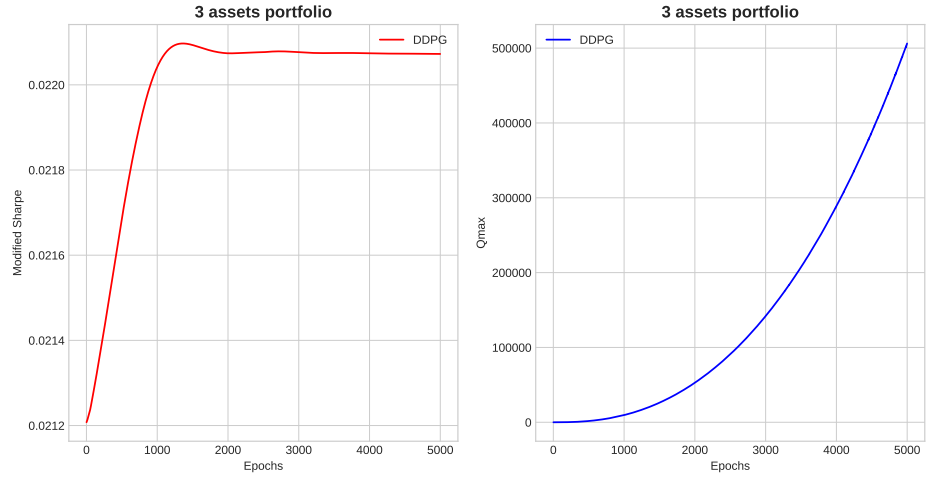
(a) Sharpe results for 3 stocks portfolio (b) Qmax results for 3 stocks portfolio

Figure 5.21: Results of training DDPG agent on 3 stocks portfolios on the South African dataset using Qmax to maximise the classic Sharpe ratio



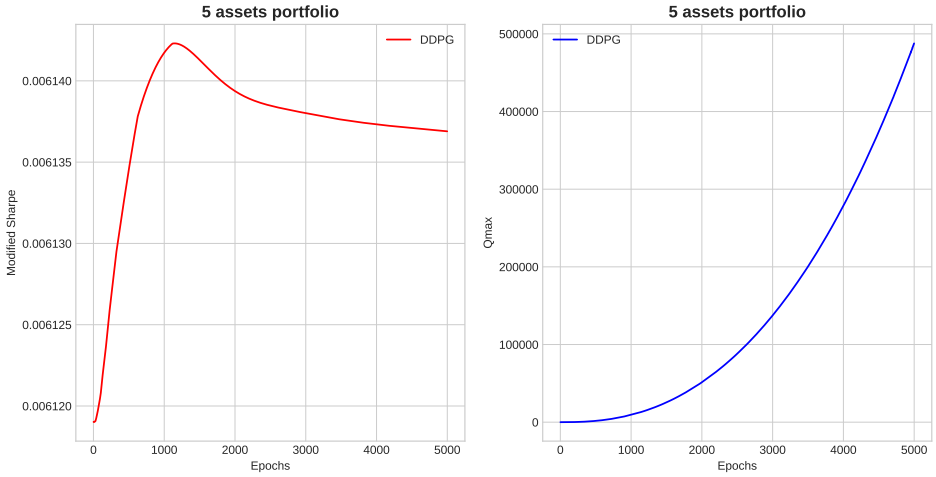
(a) Sharpe results for 5 stocks portfolio (b) Qmax results for 5 stocks portfolio

Figure 5.22: Results of training DDPG agent on 5 stocks portfolios on the South African dataset using Qmax to maximise the classic Sharpe ratio



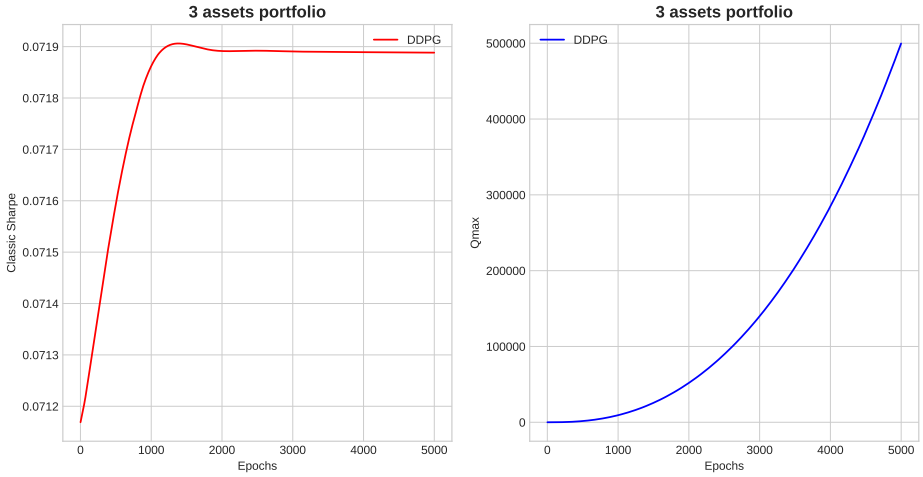
(a) Sharpe results for 3 stocks portfolio (b) Qmax results for 3 stocks portfolio

Figure 5.23: Results of training DDPG agent on 3 stocks portfolios on the South African dataset using Qmax to maximise the modified Sharpe ratio



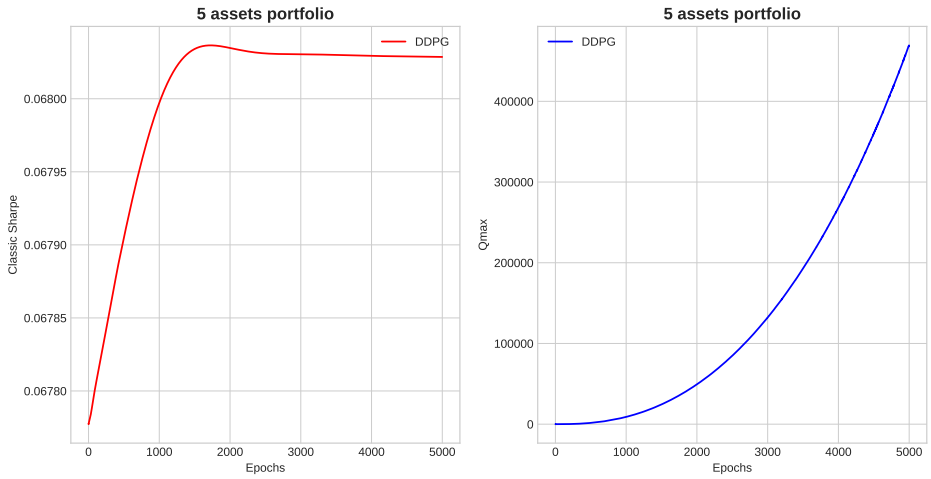
(a) Sharpe results for 5 stocks portfolio (b) Qmax results for 5 stocks portfolio

Figure 5.24: Results of training DDPG agent on 5 stocks portfolios on the South African dataset using Qmax to maximise the modified Sharpe ratio



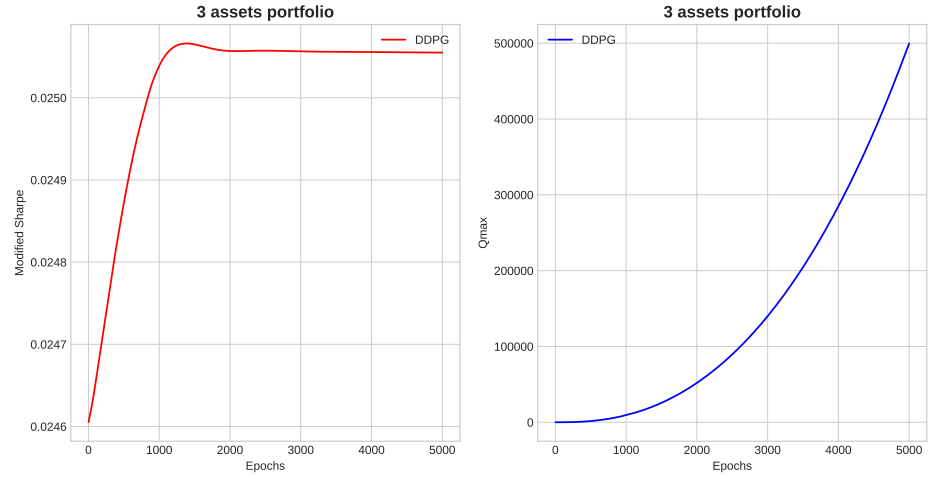
(a) Sharpe results for 3 stocks portfolio (b) Qmax results for 3 stocks portfolio

Figure 5.25: Results of training DDPG agent on 3 stocks portfolios on the US dataset using Qmax to maximise the classic Sharpe ratio



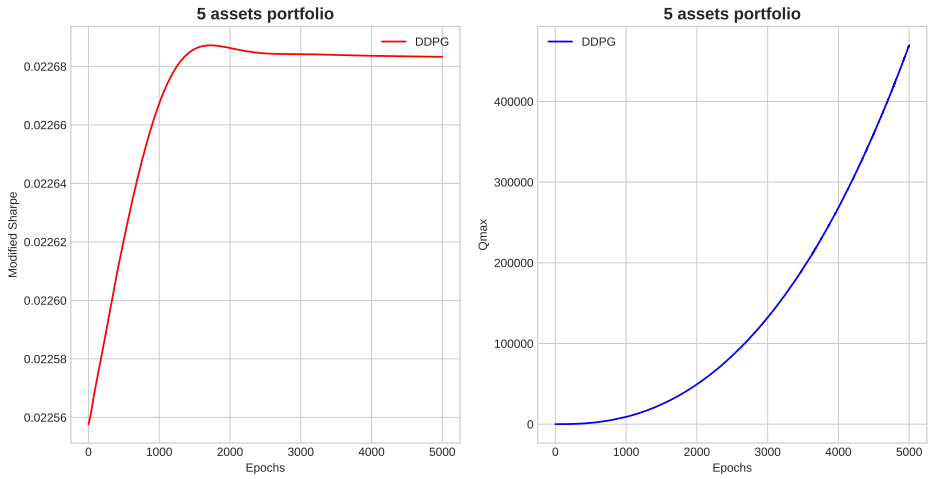
(a) Sharpe results for 5 stocks portfolio (b) Qmax results for 5 stocks portfolio

Figure 5.26: Results of training DDPG agent on 5 stocks portfolios on the US dataset using Qmax to maximise the classic Sharpe ratio



(a) Sharpe results for 3 stocks portfolio (b) Qmax results for 3 stocks portfolio

Figure 5.27: Results of training DDPG agent on 3 stocks portfolios on the US dataset using Qmax to maximise the modified Sharpe ratio



(a) Sharpe results for 5 stocks portfolio (b) Qmax results for 5 stocks portfolio

Figure 5.28: Results of training DDPG agent on 5 stocks portfolios on the US dataset using Qmax to maximise the modified Sharpe ratio

Table 5.8: Overall results of training the three learning algorithms on 3 stocks portfolio

Dataset	Evaluation function	Average time(s)	Optimum Sharpe ratio	Optimum epoch
SA 3 stocks port	classic Sharpe	14753.54	0.0221	1363
	modified Sharpe	14869.05	0.0037	1354
SA 5 stocks port	classic Sharpe	83695.39	0.0385	1144
	modified Sharpe	86252.44	0.0061	1147
US 3 stocks port	classic Sharpe	10174.29	0.0719	1384
	modified Sharpe	10375.67	0.0251	1384
US 5 stocks port	classic Sharpe	58831.47	0.0702	5000
	modified Sharpe	59706.90	0.0242	5000

Figures 5.21a to 5.28b show the positive correlation between the reward and action-value produced by the DDPG algorithm. The rewards which are, in this case, the modified and classic Sharpe ratio increase in proportion to action-value for all scenarios until they reach maximum Sharpe ratios and drop a bit thereafter. The future study will take into consideration factors which constitute this dropping, one of which might be exploration problem.

Further demonstration on Table 5.8 indicates that DDPG is inefficient as it runs up to a day in all scenarios. The inclusion of too many variables on the modified Sharpe ratio leads to the algorithm running longer than in the classic Sharpe.

5.3.3 Summary and analysis of training results

The learning algorithms are trained by running algorithms numerous times. Tables 5.9 and 5.10 and Figures 5.29 and 5.30 present overall training results and a summary of the performance comparison among the three learning algorithms. The GA and PSO agents are allowed to evolve over 5000 iterations for 5 times and 5000 episodes for DDPG, results are then averaged. Algorithms performances are measured based on three metrics, in particular accuracy, efficiency and reliability. The PSO appears to be more accurate as it obtains the highest value on classic and modified Sharpe ratio for all datasets and portfolio instances, and DDPG is trailing off at the end of all algorithms. In terms of the computational time, the PSO has the lowest computational time relative to the GA. The DDPG has the highest computational time that could be due to the difficulty of comparing iterations with episodes. The three algorithms appear to be reliable as at the end of each iteration and episode, they all tend to obtain optimum values consistently. The overall comparison is shown in Table 5.11.

In terms of weights allocations on the 3 and 5 stocks portfolios, the evidence from all algorithms shows that the allocations are not made entirely on a single variable but consider the entire Sharpe ratios. For the SA dataset, FNB obtained the highest allocation across all scenarios, even though its performance in terms of mean return is the lowest but better performed concerning standard deviation (lowest standard deviation). Similarly, for the US market, the AAPL received the highest allocation though, it performed poorly against MSFT and XLK in terms of the mean return but defeated against the AMZN and GOOG in respect of standard deviation.

Table 5.9: Overall results of training the three learning algorithms on 3 stocks portfolio

Algorithm	SA				US			
	Classic Sharpe	time(s)	Modified Sharpe	time(s)	Classic Sharpe	time(s)	Modified Sharpe	time(s)
PSO	0.034118	1315.42	0.00610	1327.49	0.07442	1319.74	0.02630	1294.59
GA	0.03407	3094.45	0.00609	3182.74	0.07442	3073.92	0.02630	3122.85
DDPG	0.0221	14753.54	0.0037	14869.05	0.0719	10174.29	0.0251	10375.67

Table 5.10: Overall results of training the three learning algorithms on 5 stocks portfolio

Algorithm	SA				US			
	Classic Sharpe	time(s)	Modified Sharpe	time(s)	Classic Sharpe	time(s)	Modified Sharpe	time(s)
PSO	0.07154	1421.18	0.07107	1415.70	0.07461	1434.86	0.02630	1482.33
GA	0.07104	3197.03	0.01594	3165.181	0.07458	3122.37	0.02630	3201.41
DDPG	0.0385	83695.39	0.0061	86252.44	0.0702	58831.47	0.0242	59706.90

Table 5.11: Overall results comparison of training the three learning algorithms

Algorithm	Accuracy	Efficiency	Reliability
PSO	1^{st} position	1^{st} position	Yes
GA	2^{nd} position	2^{nd} position	Yes
DDPG	3^{rd} position	3^{rd} position	Yes

5.3.4 Applying learning algorithms results on test dataset

Refer to Figures 5.29 and 5.30 and Tables 5.12 and 5.13. The generated portfolio weights from the training phase are further applied and tested on the test dataset. A uniform allocation, commonly regarded as a naive method is used to benchmark learning algorithms. In all scenarios, the classic Sharpe ratio produced higher values compared to the modified Sharpe ratio as expected because the modified Sharpe considers the third and fourth moments of return in its formulation. Taking a close look at the results the uniform allocation is outclassed to a great extent besides for the SA dataset where the uniform allocation outperforms the DDPG twice on 3 assets portfolios. The results obtained from the GA, in relation to Sharpe ratios exceed the

PSO results, yeilding a total of 5 scenarios showing better results than PSO. The PSO results show a better results than any other algorithms on 3 occasions. In general terms, GA always come top in the list followed by PSO and DDPG respectively and uniform allocation at the end.

Table 5.12: Overall summary of test results on SA dataset for each algorithm

Algorithm	Sharpe value
3 modified DDPG	0.010146
3 modified Uniform	0.010194
3 modified PSO	0.011066
3 modified GA	0.011076
5 modified Uniform	0.012864
5 modified DDPG	0.012923
5 modified PSO	0.013257
5 modified GA	0.013264
3 classic DDPG	0.028802
3 classic Uniform	0.028951
3 classic PSO	0.031780
3 classic GA	0.031791
5 classic Uniform	0.037269
5 classic DDPG	0.037451
5 classic GA	0.052051
5 classic PSO	0.052103

Table 5.13: Overall summary of test results on US dataset for each algorithm

Algorithm	Sharpe ratio
3 modified Uniform	0.018388
3 modified DDPG	0.018562
5 modified Uniform	0.019063
5 modified DDPG	0.019104
3 modified GA	0.019933
3 modified PSO	0.019934
5 modified PSO	0.020365
5 modified GA	0.020526
3 classic Uniform	0.062003
3 classic DDPG	0.062369
5 classic Uniform	0.065774
5 classic DDPG	0.065924
3 classic GA	0.066122
3 classic PSO	0.066130
5 classic PSO	0.068043
5 classic GA	0.068590

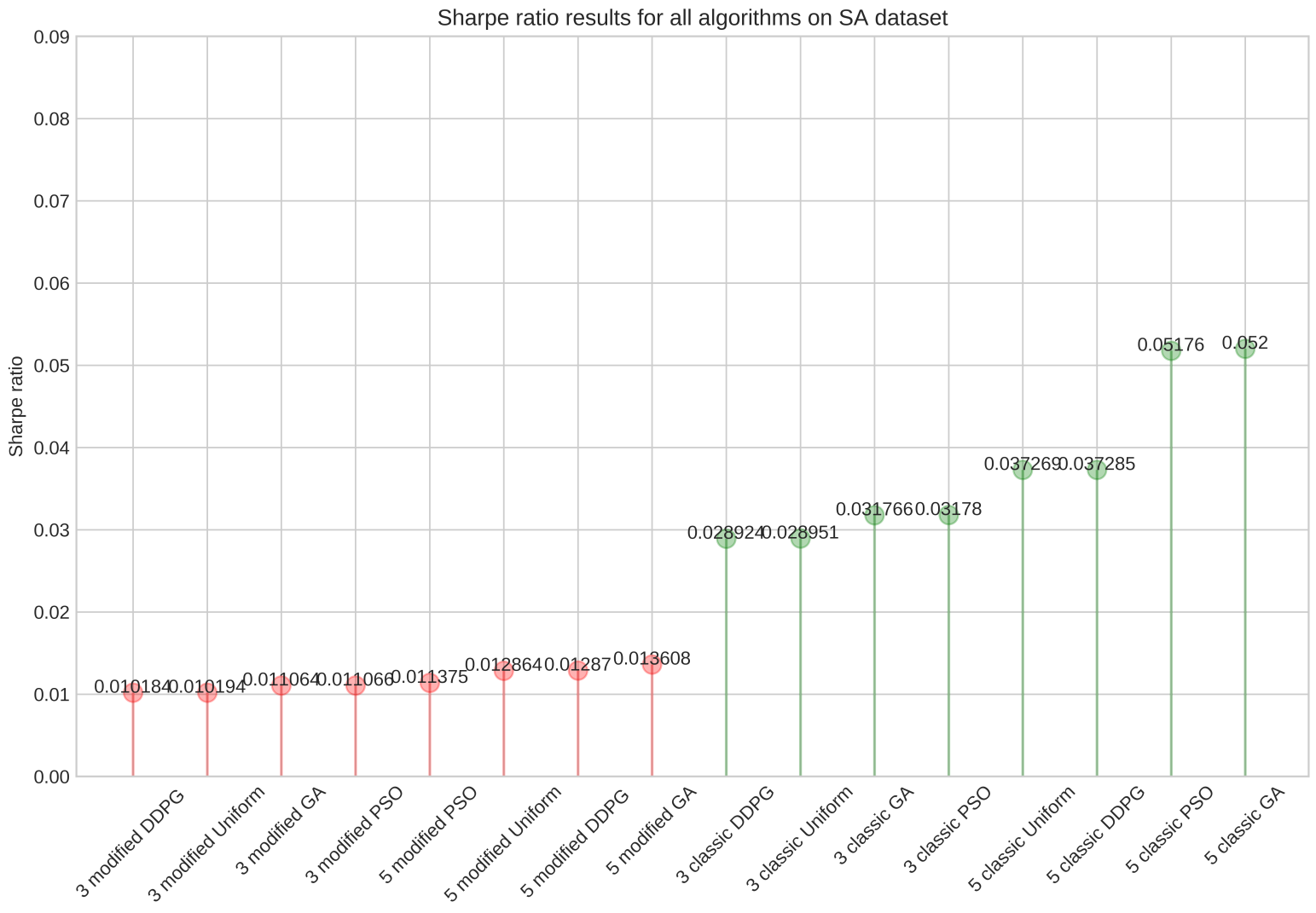


Figure 5.29: Results of training the learning algorithms using the South African test dataset

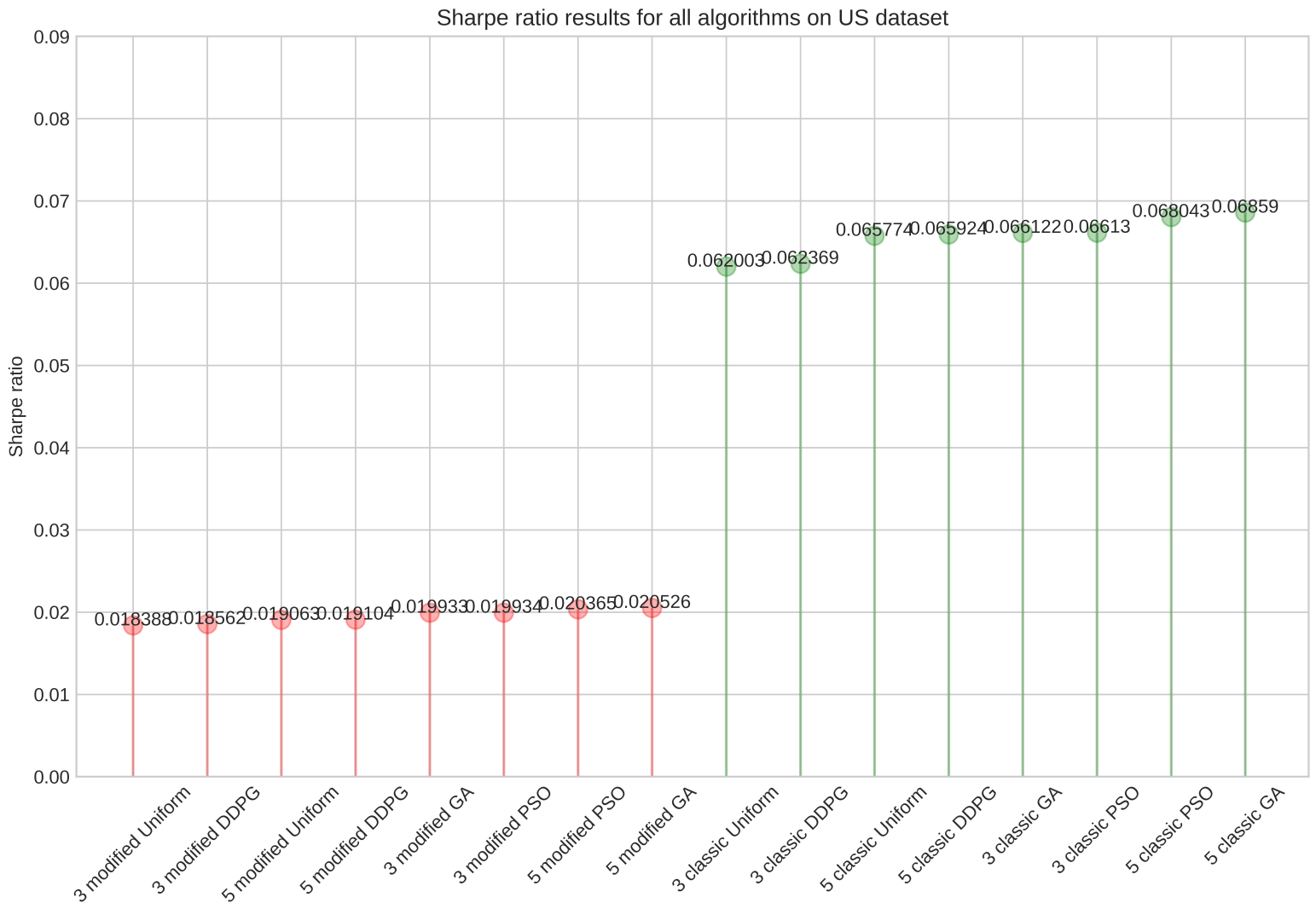


Figure 5.30: Results of training the learning algorithms using the US test dataset

5.3.5 Trading strategies

Table 5.14 shows the cumulative returns performance results on each algorithm. Asset allocation weights obtained from training various algorithms are further backtested on trading using the test dataset.

The trading process exploits buy and hold based on each algorithm with the uniform allocation portfolio as a benchmark. Ideally, the GA and PSO produce better performance in all instances and datasets than the uniform allocation and DDPG. In most cases, the DDPG outperforms the uniform allocation by an incredibly small margin except on SA dataset for the 3 portfolios where the uniform outperform the DDPG.

Table 5.14: Portfolio test results for buy-and-hold strategy each algorithm

Sharpe on	Dataset	No of stocks	Algorithm	Relative return	Log return
Classic Sharpe ratio	SA	3	Uniform	0.34	40.09 %
			GA	0.44	54.68 %
			PSO	0.44	54.73 %
			DDPG	0.34	39.83 %
Modified Sharpe ratio	SA	3	Uniform	0.34	40.09 %
			GA	0.44	54.67 %
			PSO	0.44	54.73 %
			DDPG	0.34	39.83 %
Classic	SA	5	Uniform	0.42	52.74 %
			GA	0.69	98.56 %
			PSO	0.69	99.24 %
			DDPG	0.43	53.07 %
Modified Sharpe ratio	SA	5	Uniform	0.42	52.74 %
			GA	0.68	98.09 %
			PSO	0.73	107.59 %
			DDPG	0.43	53.07 %
Classic Sharpe ratio	US	3	Uniform	0.54	71.81 %
			GA	0.62	85.55 %
			PSO	0.62	85.58 %
			DDPG	0.55	72.52 %
Modified Sharpe ratio	US	3	Uniform	0.54	71.81 %
			GA	0.60	81.37 %
			PSO	0.60	81.41 %
			DDPG	0.55	72.52 %
Classic Sharpe ratio	US	5	Uniform	0.54	72.17 %
			GA	0.63	88.11 %
			PSO	0.63	87.43 %
			DDPG	0.55	72.54 %
Modified Sharpe ratio	US	5	Uniform	0.54	72.17 %
			GA	0.61	84.53 %
			PSO	0.60	81.41 %
			DDPG	0.55	72.54 %

Moreover, results obtained in Table 5.14 suggest that no improvement made by the Modified

Sharpe ratio to address the question of whether the modified Sharpe ratio improves the classic Sharpe ratio. Figures 5.31a and 5.31b show the relationship between the two Sharpe formulations with relative returns. The results from applying linear regression illustrate a weak positive correlation between the Sharpe ratios and the relative returns for all algorithms and datasets. It is also difficult to prove the relationship between the two formulations by plotting the two on the same graph.

The results from the SA dataset yielded R^2 = 0.096 and a Pearson correlation 0.31. Consequently, the US dataset achieved R^2 = 0.015 (US) and a Pearson correlation 0.12.

Looking closely at the dataset at hand as shown in Tables 5.1 and 5.2, non of the stocks possess a zero skewness and kurtosis confirming that the dataset is asymmetric. It is therefore imperative to pick the modified Sharpe ratio over classic Sharpe ratio to cater for asymmetric distribution nature of the stock market dataset even though the former neither improves nor impairs the latter.

(a) Relationship between Sharpe ratio and relative return on US dataset

(b) Relationship between Sharpe ratio and relative return on US dataset

Figure 5.31: Relationship between Sharpe ratio and the relative returns for each dataset

5.4 Summary of results

As expected, all learning algorithms are capable of solving the portfolio allocation problem. The PSO and GA provide relatively better results than the DDPG and uniform allocation on numer-

ous instances although, the GA has a higher computational time relative to the PSO. On the other hand, the DDPG has the highest computational time as a result of employing episodes as opposed to iterations.

For the testing purposes, the portfolio weights generated from the training are tested on the test dataset and then applied on trading using buy and hold strategy. The uniform allocation portfolio recorded the lowest Sharpe ratio of all the algorithms slightly below that on DDPG. On the side of trading results, PSO and GA produce better results than the DDPG and uniform allocation. The DDPG could not improve much on the benchmark.

In general, the GA and PSO have comparable performance in terms of maximising the modified Sharpe ratio. It is also imperative to pick the modified Sharpe ratio over classic Sharpe ratio to cater for asymmetric distribution nature of the stock market dataset even though the former neither improves nor impairs the latter.

6

Conclusion

This study is concerned with the application of the GA, PSO and DDPG in solving the portfolio optimisation problem. Specifically, the aim is to maximise the portfolio weights in asset allocation. Based on the results presented in training, the conclusion is that the research has been very successful because all algorithms are reliable in evolving and maximising the portfolio weights as anticipated in terms of performance with the exception of the DDPG which did not improve the benchmark. In comparison with one another, PSO showed the best training performance in computational time with the GA following close behind and DDPG at the end of the list. The result also shows that it is not easy to compare iterations with the DDPG episodes.

The training results also indicated that the PSO and GA are more efficient as they reach

the highest value of classic and modified Sharpe ratio within a few iterations. Although the GA seemed to take more time than PSO to attain its highest Sharpe value, it has recorded the highest Sharpe ratio among all the three algorithms. The demonstration also shows that the three algorithms outperforms the uniform allocation in acquiring the highest classic and modified Sharpe ratio. The GA obtains the highest Sharpe ratio followed by the PSO, DDPG and the uniform allocation at the bottom. The results from the buy-and-hold strategy suggest that the three algorithms outclass the uniform allocation with GA at the top on the list followed by PSO.

The study also shows applicability of the third and the fourth moment of returns into Sharpe ratio. Further research into the application of the DDPG is required before concluding that the algorithm is suitable for solving portfolio optimisation problems. Moreover, further research will involve the hybrid of GA or PSO with DDPG to determine if there are any improvements in the results, and it must involve the larger portfolio scenario.

References

[1] H. Markowitz, “Portfolio selection,” *Journal of Finance*, vol. **7**, pp. 77–91, 1952.

[2] B. Naqvi, N. Mirza, W. A. Naqvi, and S. K. A. Rizvi, “Portfolio optimisation with higher moments of risk at the pakistan stock exchange,” *Economic Research-Ekonomska Istraživanja*, vol. 30, no. 1, pp. 1594–1610, 2017. [Online]. Available: https://doi.org/10.1080/1331677X.2017.1340182

[3] S. Yourdkhani, “Portfolio management by using value at risk (var) (a comparison between particle swarm optimization and genetic algorithms),” *Life Science Journal*, vol. 11, no. 1 SPECL. ISSUE, pp. 15–26, 2014, cited By 1. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-84892380064&partnerID=40&md5=1070436b4cdf71e0b29d902471d2e946

[4] H. Zhu, Y. Wang, K. Wang, and Y. Chen, “Particle swarm optimization (pso) for the constrained portfolio optimization problem,” *Expert Systems with Applications*, vol. 38, no. 8, pp. 10 161–10 169, 2011, cited By 90. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-79953714837&doi=10.1016%2fj.eswa.2011.02.075&partnerID=40&md5=82666ca8cd409594b57629dbb4d0866c

[5] J.-F. Chang and P. Shi, “Using investment satisfaction capability index based particle swarm optimization to construct a stock portfolio,” *Information Sciences*, vol. 181, no. 14, pp. 2989–2999, 2011, cited By 39. [Online].

Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-77956620256&doi=10.1016%2fj.ins.2010.05.008&partnerID=40&md5=922f289ee4aa7dfbccf13b268d8fc771

[6] M. Corazza, G. Fasano, and R. Gusso, "Particle swarm optimization with non-smooth penalty reformulation, for a complex portfolio selection problem," *Applied Mathematics and Computation*, vol. 224, pp. 611–624, 2013, cited By 20. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-84885107711&doi=10.1016%2fj.amc.2013.07.091&partnerID=40&md5=1ab52dc68903b5bbaca468e002bfbb1f

[7] V. Ranković, M. Drenovak, B. Urosevic, and R. Jelic, "Mean-univariate garch var portfolio optimization: Actual portfolio approach," *Computers and Operations Research*, vol. 72, pp. 83–92, 2016, cited By 4. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-84959514497&doi=10.1016%2fj.cor.2016.01.014&partnerID=40&md5=bb2e23c651f489765d994f482d19cd9a

[8] D. Cheong, Y. Kim, H. Byun, K. Oh, and T. Kim, "Using genetic algorithm to support clustering-based portfolio optimization by investor information," *Applied Soft Computing Journal*, vol. 61, pp. 593–602, 2017, cited By 1. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85028936428&doi=10.1016%2fj.asoc.2017.08.042&partnerID=40&md5=f4b30301889286f4aa0f642fdb6a2e19

[9] J. Moody, L. Wu, Y. Liao, and M. Saffell, "Performance functions and reinforcement learning for trading systems and portfolios," *Journal of Forecasting*, vol. 17, no. 5-6, pp. 441–470, 1998, cited By 65. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-0001164059&partnerID=40&md5=8885a42b2b9763ed20ee7aef65b3ae97

[10] D. Maringer and T. Ramtohul, "Regime-switching recurrent reinforcement learning for investment decision making," *Computational Management Sci-*

ence, vol. 9, no. 1, pp. 89–107, 2012, cited By 8. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-84855224037&doi=10.1007%2fs10287-011-0131-1&partnerID=40&md5=3c9fc2694f9d18198a368d773da59c37

[11] P. Pendharkar and P. Cusatis, "Trading financial indices with reinforcement learning agents," *Expert Systems with Applications*, vol. 103, pp. 1–13, 2018, cited By 0. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85043385087&doi=10.1016%2fj.eswa.2018.02.032&partnerID=40&md5=2fd6c8ceb9934f691c18f8f4a04b1f8f

[12] H. Valian, M. Jafari, and D. Golmohammadi, "Resource allocation with stochastic optimal control approach," *Annals of Operations Research*, vol. 239, no. 2, pp. 625–641, 2016, cited By 0. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-84902709566&doi=10.1007%2fs10479-014-1653-z&partnerID=40&md5=a4045360d6f69febcc133276127a1506

[13] J. P. Morgan, "Introduction to risk metrics," New York: Morgan Guaranty Trust Company, Tech. Rep. 4th ed, 1995.

[14] F. Sharpe, "Mutual fund performance," *The journal of business*, vol. 39, no. 1, pp. 119–138, 1966.

[15] S. F.W., "Adjusting for risk in portfolio performance meansurement," *The journal of portfolio management*, vol. 1, no. 2, pp. 29–34, 1975.

[16] ——, "The sharpe ratio," *The journal of portfolio management*, vol. 21, no. 1, pp. 49–58, 1994.

[17] J. Kenney and E. Keeping, *Mathematics of statistics*, ser. Mathematics of Statistics. Van Nostrand, 1947, no. pt. 2. [Online]. Available: https://books.google.co.ls/books?id=UdlLAAAAMAAJ

[18] G. N. Gregoriou and J.-P. Gueyie, “Risk-adjusted performance of funds of hedge funds using a modified sharpe ratio,” *The Journal of Wealth Management*, vol. 6, no. 3, pp. 77–83, 2003. [Online]. Available: https://jwm.pm-research.com/content/6/3/77

[19] L. Favre and J.-A. Galeano, “Mean-modified value-at-risk optimization with hedge funds,” *The Journal of Alternative Investments*, vol. 5, pp. 21–25, 01 2002.

[20] V. Kachitvichyanukul, “Comparison of three evolutionary algorithms: Ga, pso, and de,” *Industrial Engineering and Management Systems*, vol. 12, pp. 215–223, 09 2012.

[21] J. Kennedy and R. C. Eberhart, “Particle swarm optimization,” in *Proceedings of the IEEE International Conference on Neural Networks*, 1995, pp. 1942–1948.

[22] J. Kennedy and R. Mendes, “Neighborhood topologies in fully informed and best-of-neighborhood particle swarms,” *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 36, no. 4, pp. 515–519, July 2006.

[23] D. Bratton and J. Kennedy, “Defining a standard for particle swarm optimization,” 2007, pp. 120–127, cited By 847. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-34548720461&doi=10.1109%2fSIS.2007.368035&partnerID=40&md5=9abf5389dba788b22d8819eef47d7119

[24] J. Kennedy, “Small worlds and mega-minds: Effects of neighborhood topology on particle swarm performance,” vol. 3, 1999, pp. 1931–1938, cited By 793. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-84899047533&doi=10.1109%2fCEC.1999.785509&partnerID=40&md5=62438339bfb3f8ead2447d3693c07dce

[25] A. J. R. Medina, G. T. Pulido, and G. Ramírez-Torres, “A comparative study of neighborhood topologies for particle swarm optimizers,” in *IJCCI*, 2009.

[26] P. Gonçalo, “Particle swarm optimization,” *INESCID and Intituto Superior Tecnico, Porto Salvo*, vol. 15, May 2011.

[27] J. Holland, “Genetic algorithm,” *Scientic American*, vol. **7**, pp. 66–72, 1992.

[28] Darwin, C. R., *On the origin of species*. London: John Murray, 1859.

[29] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, ser. Artificial Intelligence. Addison-Wesley Publishing Company, 1989. [Online]. Available: https://books.google.co.za/books?id=2IIJAAAACAAJ

[30] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. The MIT Press, 2018. [Online]. Available: http://incompleteideas.net/book/the-book-2nd.html

[31] R. Bellman, “A markovian decision process,” *Indiana Univ. Math. J.*, vol. 6, pp. 679–684, 1957.

[32] N. Bauerle and U. Rieder, *Markov Decision Process with Applications in Finance*. Spring, Berlin, Heidelbery, 2011.

[33] M. L. Puterman, *Markov Decision Processes*. Wiley, 1994.

[34] L. N. Steimle, D. L. Kaufman, and B. T. Denton, “Multi-model markov decision processes,” *Optimization online*, 2019.

[35] P. Tadepalli and D. Ok, “Model-based average reward reinforcement learning,” *Artif. Intell.*, vol. 100, no. 1-2, pp. 177–224, Apr. 1998. [Online]. Available: https://doi.org/10.1016/S0004-3702(98)00002-2

[36] M. V. Otterlo and R. May, “Markov decision processes: Concepts and algorithm,” 2009.

[37] H. A. Ronald, *Dynamic Programming and Markov Processes*. The M.I.T. Press, 1960.

[38] C. J. C. H. Watkins, “Learning from delayed rewards,” Ph.D. dissertation, King’s College, Cambridge, UK, May 1989.

[39] C. J. C. H. Watkins and P. Dayan, “Q-learning,” *Machine learning*, vol. 8, no. 3–4, pp. 279–292, 1992.

[40] G. Rummery and M. Niranjan, “On-line q-learning using connectionist systems,” *Technical Report CUED/F-INFENG/TR 166*, 11 1994.

[41] T. Lillicrap, J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, “Continuous control with deep reinforcement learning,” 2016, cited By 217. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85063867176&partnerID=40&md5=cd0e33e3be1157477566e63b63b3c8f6

[42] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, “Deterministic policy gradient algorithms,” in *ICML*, Beijing, China, Jun 2014.

[43] M. Wang, C. Li, H. Xue, and F. Xu, “A new portfolio rebalancing model with transaction costs,” *Journal of Applied Mathematics*, vol. 2014, 2014, cited By 0. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-84900993080&doi=10.1155%2f2014%2f942374&partnerID=40&md5=71c518f3b0353e13f634d25c0184206e

[44] Z. Pei and M. Eisenbach, “Acceleration of the particle swarm optimization for peierls–nabarro modeling of dislocations in conventional and high-entropy alloys,” *Computer Physics Communications*, vol. 215, 02 2017.