Deep Reinforcement Learning for Financial Portfolio Optimisation

Nigel Cuschieri

Supervised by Dr Vince Vella

Co-supervised by Dr Josef Bajada

Department of Artificial Intelligence

Faculty of Information & Communication Technology

University of Malta

January, 2022

A dissertation submitted in partial fulfilment of the requirements for the degree of M.Sc. in Artificial Intelligence.

L-Università
ta' Malta

WWW.UM.EDU.MT

First edition, Saturday 8th January, 2022

To All My Friends, My Family And My Love

For their undying patience and support.

Acknowledgements

I would like to express my sincere gratitude to my supervisor Dr. Vincent Vella, and co-supervisor Dr. Josef Bajada for their guidance throughout the writing of this dissertation. Their readiness to share their wealth of knowlegde, constant availability and constructive feedback have been essential contributions to its completion. Working with them both has been a pleasure.

Abstract

Portfolio Selection (PS) is a perennial financial engineering problem that requires determining a strategy for dynamically allocating wealth among a set of portfolio assets to maximise the long-term return. We investigate state-of-the-art Deep Reinforcement Learning (DRL) algorithms that have proven to be ideal for continuous action spaces, mainly Deep Deterministic Policy Gradient (DDPG) and Twin Delayed Deep Deterministic Policy Gradient (TD3), for the PS problem. Furthermore, we investigate the effect of including stock movement prediction indicators in the state representation and the potential of using an ensemble framework that combines multiple DRL models. We formulate experiments to evaluate our DRL models on real data from the American stock market, against benchmarks including state-of-the-art online portfolio selection (OLPS) approaches, using measures consisting of Average daily yield, Sharpe ratio, Sortino ratio and Maximum drawdown. Our experiments show that TD3-based models generally perform better than DDPG-based ones when used on real stock trading data. Furthermore, the introduction of additional financial indicators in the state representation was found to have a positive effect overall. Lastly, an ensemble model also showed promising results, consistently beating the baselines used, albeit not all other DRL models.

Contents

List of Figures

List of Tables

List of Abbreviations

ML Machine Learning

DL Deep Learning

MDP Markov Decision Process

PS Portfolio Selection

RL Reinforcement Learning

DRL Deep Reinforcement Learning

DQN Deep Q-Network

PG Policy Gradient

DPG Deterministic Policy Gradient

DDPG Deep Deterministic Policy Gradient

TD3 Twin Delayed Deep Deterministic Policy Gradient

APV Accumulative Portfolio Value

MDD Maximum Drawdown

ANN Artificial Neural Network

DANN Deep Artificial Neural Network

CNN Convolutional Neural Network

RNN Recurrent Neural Network

LSTM Long short-term memory

D-FFNN Deep Feedforward Neural Network

AE Autoencoder

DBN Deep Belief Networks

OLPS On-line Portfolio Selection

CRP Constant Rebalanced Portfolio

UCRP Uniform Constant Rebalanced Portfolio
BCRP Best Constant Rebalanced Portfolio
OLMAR On-Line Moving Average Reversion
RMR Robust Median Reversion
PAMR Passive Aggressive Mean Reversion
WMAMR Weighted Moving Average Mean Reversion
ONS Online Newton Step
EG Exponential Gradient
UP Universal Portfolios
ACKTR Actor-Critic using Kronecker-factored Trust Region
PPO Proximal Policy Optimisation
TRPO Trust Region Policy Optimisation
SAC Soft Actor-Critic
MSBE Mean-Squared Bellman Error
SARSA State–Action–Reward–State–Action
NYSE New York Stock Exchange
SP500 Standard and Poor Exchange
DJIA Dow Jones Industrial Average
A2C Advantage Actor-Critic
A3C Asynchronous Advantage Actor-Critic

1

Introduction

The work presented within this dissertation has been accepted to be presented at the FinPlan workshop, which is to be held in conjunction with the 31st International Conference on Automated Planning and Scheduling (ICAPS 2021).

Online portfolio selection (OLPS) is considered a principal problem in computational finance and has been studied across various research communities. Among others, these include finance, statistics, artificial intelligence, machine learning, and data mining (Li and Hoi, 2014). Some popular theories suggest that stock markets movements are simply a random walk, making it a fool's game to attempt to predict their trends (Shah and Isah, 2019). The leading cause of this challenge is the number of variables involved. Shah and Isah (2019) describe that in the short term, the market behaves like a voting machine which tallies up the popular firms, but in the longer term acts like a weighing machine which assesses the substance of companies. This description hints that there is scope for predicting the market movements over a longer time frame.

1.1 | Portfolio Selection

Portfolio Selection (PS) is a perennial financial engineering problem that requires determining a strategy of dynamically allocating wealth among a set of assets to maximise the long-term return (Li et al., 2012). The PS problem is a fundamental one in the investment industry, which has now been studied intensively for over 50 years, as it originated from the seminal paper of Markowitz (1952). As Li and Hoi (2014) indicate, there are two major schools for investigating the PS problem. These are the *Mean Variance Theory* (Markowitz, 1952) originating from the finance community, and the *Capital Growth Theory* (Cover, 1996; Kelly, 1956) originating from information theory (Li and

Hoi, 2014). The Mean Variance Theory focuses on a single-period or batch portfolio selection. It aims to trade off expected return (mean) with risk (variance), to create an optimal portfolio subject to the investor's risk-return profile (Li and Hoi, 2014). In contrast, the Capital Growth Theory focuses on multiple-period or sequential portfolio selection. Additionally, it aims to maximise the portfolio's expected growth rate, or expected mean logarithmic return (Li and Hoi, 2014). Despite the fact that these Capital Growth Theory aims follow the same concept of focusing more on return rather than risk awareness, the latter assumes the use of a commonly used function within finance, which is the log return calculation. This involves calculating the change in price within a period, dividing it by the previous price, and finally calculating the logarithm of the result (Quantivity, 2011). Both theories pose a solution to the portfolio selection task. However, only the Capital Growth Theory fits the "online" scenario (Li and Hoi, 2014) and incorporates the online machine learning perspective, consisting of multiple periods or steps. This theory of PS has set the foundation for studies over the years, all posing solutions that sequentially allocate portfolios. Many of which make use of machine learning and online learning methodologies (Li et al., 2012).

1.1.1 | Online Portfolio Selection

Online Portfolio Selection (OLPS) strategies aim to sequentially select portfolios over a group of assets in order to achieve portfolio optimisation targets (Li and Hoi, 2014). For each period or step, the portfolio weight is tuned to maximise the cumulative wealth (Li and Hoi, 2014). Due to the numerous amounts of studies aiming to improve OLPS performance, today, one can find a great variety of state-of-the-art OLPS strategies and approaches (Li and Hoi, 2014). Additionally, due to their availability and replicability (Islam et al., 2017), OLPS algorithms such as Exponential Gradient (EG) (Helmbold et al., 1996, 1998), Passive Aggressive Mean Reversion (PAMR) (Li et al., 2012), and Online Moving Average Reversion (OLMAR) (Li and Hoi, 2012) are popularly used in portfolio optimisation studies as benchmarks to evaluate the performance of their portfolios (Jiang and Liang, 2018; Kanwar, 2019; Li and Peng, 2019; Patel, 2018; Ye et al., 2020).

1.2 | Reinforcement Learning

RL is an actively researched branch of machine learning in which an agent is iteratively fed new data to discover which actions yield the most reward by trying them out (Cumming, 2015; Khushi and Meng, 2019; Sutton and Barto, 1999). In contrast to a supervised approach, in an RL based approach the learner is not told which actions to take. Sutton and Barto (1999) defines RL as

> ‘learning what to do, how to map situations to actions, so as to maximise a numerical reward signal.‘

The RL algorithm taxonomy is primarily split based on the access to a model of the environment (Sutton and Barto, 1999). For example, the AlphaGo algorithm created by Silver et al. (2016) to play the Go board game is a model-based one. This is due to the Go board game being confined by a set of defined rules, allowing the agent to plan ahead and see what would happen for a range of possible choices (Kanwar, 2019). In contrast, model-free RL algorithms do not make use of such a model. This allows them to be more efficient and simpler to implement (Kanwar, 2019). Furthermore, model-free RL can be applied to a broader range of problem domains, including OLPS which has a continuous action space.

1.2.1 | Deep Reinforcement Learning Frameworks

Deep Reinforcement Learning (DRL) is the combination of RL with Deep Learning from Deep Neural Networks, extending to tasks with high-dimensional input and action spaces (Khadka et al., 2019). Deep learning (DL) has been successful in speech recognition (Noda et al., 2015) and image identification (Liu et al., 2020), and was shown to have the capability to capture complex, non-linear patterns (Liang et al., 2018). Today, a wide array of state-of-the-art RL algorithms designed for DRL exist, and many have been applied to the portfolio management and optimisation domain. Such algorithms include *Advantage Actor-Critic* (A2C) (Kang et al., 2018), *Soft Actor-Critic* (SAC) (Haarnoja et al., 2018), *Proximal Policy Optimization* (PPO) (Schulman et al., 2017), *Deep Deterministic Policy Gradient* (DDPG) (Lillicrap et al., 2016; Silver et al., 2014), and *Twin Delayed Deep Deterministic Policy Gradient* (TD3) (Fujimoto et al., 2018), amongst others.

1.3 | Artificial Intelligence for Portfolio Selection

In past studies, portfolio optimisation has been approached as a stock price prediction problem (Ravikumar and Saraf, 2020). Those interested in finding a machine learning based solution to this problem have attempted to do so using supervised machine learning algorithms (Kumar et al., 2018; Ravikumar and Saraf, 2020), and specifically artificial neural networks (ANN) (Song et al., 2018) to perform regression and classification. These studies aim to increase the predictive power further with each iteration

(Song et al., 2018). Unfortunately, the stock market is very dynamic and chaotic, making it unpredictable (Song et al., 2018). Due to this, many search for alternative strategies that do not attempt to predict price movements with the use of historical market data (Heaton et al., 2017; Niaki and Hoseinzade, 2013). Instead, many strategies make use of RL.

1.3.1 | Related Works

In recent studies, RL has shown potential as a model-free machine-learning solution to the financial portfolio management problem (Jiang and Liang, 2018). Moreover, Khushi and Meng (2019) claim that financial traders' interest in RL had been inspired by the *AlphaGo* program (Silver et al., 2016), which defeated the best Go board game player Lee Sedol in 2016. Several studies, such as the ones seen in (Jiang et al., 2017), (Jiang and Liang, 2018), and (Liang et al., 2018) have set the current benchmark for this kind of study, making use of Deep Reinforcement Learning (DRL). A number of these studies, such as (Jiang and Liang, 2018) and Hegde et al. (2019), make use of RL methods introduced by the works done by Silver et al. (2014) and Lillicrap et al. (2016), which saw the proposal and creation of the underlying DDPG framework. This was designed to operate over continuous action spaces (Lillicrap et al., 2016). The approaches seen in these papers differ from previous successful attempts, such as the ones seen in (Cumming, 2015) and (Deng et al., 2017), as the RL algorithms from the latter output discrete trading signals on singular assets. Further recent studies in this field, such as the ones done by Hegde et al. (2019), Zhang et al. (2020), Kanwar (2019), and Gran (2019) have expanded on top of their work, with more risk-averse approaches (Hegde et al., 2019; Zhang et al., 2020), focus on asset correlation (Zhang et al., 2020), genetic algorithms to perform pre-training (Gran, 2019), varieties in neural networks (Kanwar, 2019), and sentiment analysis (Gran, 2019).

1.3.2 | Motivation

The works observed do not identify a superior DRL framework for portfolio optimisation. This is partly due to the lack of a standard stock trading environment, which is required to compare portfolio optimisation model performance accurately. The agents found in these RL based approaches make use of combined features including *Open*, *Close*, *High*, *Low*, and *Volume* to make portfolio trading decisions (Hegde et al., 2019). However, more complex features could be introduced to enhance performance, such as technical indicators [1] (Huang et al., 2016; Li and Hoi, 2012). Additionally, no state format has been identified as the best option in the literature observed. Furthermore, OLPS algorithms such as EG and OLMAR observe trading strategies, including *Follow-the-Winner* and *Follow-the-Loser* with Mean Reversion (Li and Hoi, 2012). This is made possible with internal functions and calculations, which could potentially benefit a DRL framework for portfolio optimisation with their inclusion within the state observed by the agent. An additional recent related study by Yang et al. (2020) combines three trading agents with different DRL algorithms in an ensemble algorithm, which is able to adjust to different market situations. Alternative factors could be used as the differing element between the combined DRL algorithms, such as the state format, which could lead to different trading behaviours and results (Hegde et al., 2019). Additionally, ensemble DRL algorithms have been scarcely studied on the portfolio selection problem.

1.4 | Aim & Objectives

This study aims to investigate whether recent advancements in state-of-the-art DRL lead to improvements in solving the portfolio selection problem when compared to standard OLPS algorithms. To address this aim, we identify the following research objectives:

1. Investigate whether recent advances in DRL can lead to improved investment performance when compared to OLPS algorithms.

2. Investigate how OLPS features can be utilised to improve DRL market state representation, leading to increased investment performance.

3. Explore possible investment performance improvements through the use of DRL ensemble strategies.

1.4.1 | Scope

This study focuses on portfolio optimisation via daily stock trading on datasets consisting of actual close price data from stock exchanges, mainly NYSE and S&P500. Higher frequency trading rates, such as hourly and minute, are beyond the scope of this research. Throughout the study, we follow the below two assumptions:

- **Zero Slippage:** All market assets are liquid enough to make every trading at the last price immediately possible when an order is placed.

[1] https://www.tradingtechnologies.com/xtrader-help/x-study/technical-indicator-definitions/list-of-technical-indicators/

- **Zero impact on market:** In financial markets, market impact is the effect that a market participant has when buying or selling an asset (Hegde et al., 2019; Vogiatzis, 2019; Zhang et al., 2020). This is typically a concern for large investors, such as financial institutions. The investments done by our trading agents are assumed to be done in an independent or personal scope, and not by large investors. Therefore, it is assumed that the trading done by our agents does not impact the market.

1.5 | Document Structure

The current chapter introduced the research area by providing reliable contextual background information. Below are the upcoming chapters with their descriptions.

- **Chapter 2 - Background & Literature Review**
  The Literature Review chapter provides an overview of the published literature with regards to portfolio selection, applications of machine learning in portfolio management, reinforcement learning frameworks, deep learning, and applications of deep reinforcement learning in portfolio management.

- **Chapter 3 - Methodology**
  The Methodology chapter presents an in-depth description of the framework proposed and the experiments within the study. The chapter starts with a detailed description of the stock data used in our study and its pre-processing prior to its use. This is followed by an in-depth description of the custom stock trading environment and the elements within, such as the state, actions, rewards, transaction costs and window lengths. Next, the internal elements of the proposed algorithms are described, including the ANN (Predictor) format and the DRL frameworks. This also includes an in-depth explanation of the internal elements of the models used in our experiments.

- **Chapter 4 - Results & Evaluation**
  The Results & Evaluation chapter describes the experiments performed in our work. For each experiment, the results are tabulated and visualised. Some experiments also include statistical tests to solidify the findings. The results are followed by discussion and evaluation.

- **Chapter 5 - Conclusion**
  The concluding chapter addresses the research questions with respect to the results obtained. Additionally, possible improvements and future work are presented.

2

Background & Literature Review

2.1 | The Stock Market, Trading and Portfolios

The stock market is a means through which individuals can invest their money in stocks. When the term “stock market“ is mentioned, one usually refers to one of the most familiar stock market indexes, such as *Standard & Poor's 500* and *Dow Jones Industrial Average*. In brief, a stock or share is a financial instrument which represents ownership in a company with a corresponding value which is proportionate to that company's assets and earnings. Therefore, through investing money on stocks, a disciplined individual may build up one's net worth. This is done by buying and selling stocks strategically. Traditionally, experienced traders may invest on a list of stocks, called a portfolio, and have the wealth invested within be dynamically allocated (Hayes et al., 2021).

2.2 | Portfolio Selection

PS is a perennial financial engineering problem that requires determining a strategy of dynamically allocating wealth among a set of assets to maximise the long-term return Li et al. (2012). A set of assets may also be called a universe of assets. Throughout these strategies, weights are set and shifted within a portfolio vector, which represents the allocation of wealth. Additionally, each strategy may shift these weights within the portfolio vector as it sees fit. In fact, whilst some strategies make use of the ability to dynamically allocate weights in batches, others opt to set static portfolio vector weights. Thus, having a single period or batch. The two major schools for investigating the PS problem are *Mean Variance Theory* (Markowitz, 1952) originating from the finance community, and *Capital Growth Theory* (Cover, 1996; Kelly, 1956) originating from information theory (Li and Hoi, 2014).

2.2.1 | Portfolio Selection Theories

The Mean Variance Theory for PS by Markowitz (1952) mathematically formulates the portfolio allocation problem on a single-period (batch) portfolio selection basis. It aims to find a portfolio vector w in a universe of M assets, which is both Greedy and Risk-Averting. The model gives the optimal portfolio vector w, which minimises volatility for a given returns level. This typically determines the optimal portfolios subject to the risk-return profile of the investor in question (Li and Hoi, 2014). The Capital Growth Theory, on the other hand, focuses on multiple-period or sequential/online portfolio selection aiming to maximise the expected growth rate or log return (Cover, 1996; Kelly, 1956).

The Mean Variance model, as described above, has proved to be useful in the past despite the drawbacks pointed out by experts such as finance practitioners (Fagiuoli et al., 2007). These drawbacks, as Fagiuoli et al. (2007) indicate, include the distributional assumptions concerning the behaviour of stock prices and the arbitrariness that the selection of a distribution class may cause. Models within the Capital Growth Theory, such as the one by Cover (1996), address these issues using OLPS algorithms. Such an approach manages to obtain portfolios based completely on the sequence of past prices, with little or no statistical processing. Further progress within this school of PS continued to enhance these approaches, and today we can find a variety of state-of-the-art PS strategies and approaches powered by machine learning algorithms (Li and Hoi, 2014). These PS approaches are classified into four tiers that are '*Follow-the-Winner*', '*Follow-the-Loser*', '*Pattern-Matching Approaches*', and '*Meta-Learning Algorithms*'.

2.3 | Online Portfolio Selection Algorithms

Throughout our study, we aim to use a number of standard benchmarks in the form of online portfolio selection (OLPS) approaches to evaluate the performance of our DRL models. This is in line with related studies such as those done by Patel (2018), Vogiatzis (2019), Li and Peng (2019), Ye et al. (2020), Kanwar (2019), and Jiang and Liang (2018). In this section, these algorithms are identified and described.

2.3.1 | Problem Setting

OLPS algorithms typically model the PS problem as a financial market with m assets and a series of n trading steps, each consisting of wealth distributed over all the assets. The *price relative vector*, consisting of m dimensions, represents the portfolio price change for each trading step. This is defined as $x_t \in \mathbb{R}^m_+, t = 1, ..., n$, where the t^{th} price relative vector consists of the ratio of t^{th} closing prices to the previous $(t-1)^{th}$. Therefore, the investment in an asset i during step t is increased (multiplied) by a factor of $x_{t,i}$ Li and Hoi (2014). Naturally, the investment for the asset would decrease if the factor is less than 1. The *portfolio vector* represents the allocation of investment wealth throughout the trading steps, and is denoted as b_t, for the t^{th} portfolio. Therefore, $b_{t,i}$ represents the ratio of wealth assigned to the i^{th} asset. No negative entries are allowed in the capital investment, as it is assumed that a portfolio is self-financed Li and Hoi (2014). A portfolio strategy for n periods can be denoted as: $b_1^n = b_1, ..., b_n$. At any period t, the capital is adjusted according to portfolio b_t at the opening time. Then the position is held until the closing time is met. Therefore the portfolio value, when excluding transaction costs, increases by a factor of:

$$b_t^T x_t = \sum_{i=1}^{m} b_{t,i} x_{t,i} \tag{2.1}$$

Due to the multiplicative increase of portfolio wealth, from period 1 to n, a portfolio strategy b_1^n increases the initial wealth, S_0, by a factor of $\prod_{t=1}^{n} b^\top x_t$. Therefore, the cumulative wealth and exponential growth rate are formulated as follows, respectively:

$$S_n(b_1^n) = S_0 \prod_{t=1}^{n} b^\top x_t = S_0 \prod_{t=1}^{n} \sum_{i=1}^{m} b_{t,i} x_{t,i} \tag{2.2}$$

$$W_n(b_1^n) = \frac{1}{n} \log S_n(b_1^n) = \frac{1}{n} \sum_{t=1}^{n} \log b_t \dot{x}_t \tag{2.3}$$

The portfolio managers' goal is to produce a portfolio strategy b_1^n, that is computed in a sequential fashion in order to achieve certain targets, such as maximising the portfolio cumulative wealth S_n (Li and Hoi, 2014).

2.3.2 | OLPS used as baselines in Recent Literature

To evaluate the performance of their DRL portfolio optimisation models, a number of recent related studies implement OLPS algorithms and execute them on their portfolio management environment. This allowed them to compare their results on a number of criteria. Notably, Patel (2018) makes use of *Online Moving Average Reversion (OLMAR)*, and *Online Newton Step (ONS)*. Vogiatzis (2019) makes use of an equal-weighted portfolio, reminiscent of *Uniform Constant Rebalanced Portfolios (UCRP)*. Kanwar (2019) also makes use of *UCRP*. Li and Peng (2019) make use of *Passive Aggressive Mean Reversion (PAMR)*, *OLMAR*, and *Weighted Moving Average Mean Reversion (WMAMR)*. Ye et al. (2020) make use of *OLMAR* and *WMAMR*. Jiang and Liang (2018) make use of long list of baselines, including *UCRP*, *OLMAR*, *PAMR*, *WMAMR*, *Robust Median Reversion (RMR)*, *ONS*, *Universal Portfolios (UP)*, and *Exponential Gradient (EG)*. All of these OLPS baselines are discussed in the following sections in this chapter. Additionally, we also include the *Best Constant Rebalanced Portfolio (BCRP)* which is a constant rebalanced portfolio achieving the highest wealth in hindsight.

2.3.3 | Online Portfolio Selection Baselines

The first pair of OLPS algorithms discussed are *Uniform Constant Rebalanced Portfolio* (UCRP) and *Best Constant Rebalanced Portfolio* (BCRP).

2.3.3.1 | Uniform Constant Rebalanced Portfolios (UCRP)

Constant Rebalanced Portfolios (CRP) is a strategy that utilises fixed proportions on asset weights and rebalances the portfolio wealth at the beginning of every period based on that proportion. As Li and Hoi (2014) indicate, the portfolio strategy can be represented as $b_1^n = \{b, b, ...\}$. The cumulative portfolio wealth achieved by a CRP strategy after n periods is therefore defined as:

$$S_n(CRP(b)) = \prod_{t=1}^{n} b^\top x_t \tag{2.4}$$

A CRP strategy with fixed proportions where $b = (\frac{1}{m}, ..., \frac{1}{m})$ is called *Uniform Constant Rebalanced Portfolios* (UCRP).

2.3.3.2 | Best Constant Rebalanced Portfolio (BCRP)

Utilising a hindsight strategy, it is possible to calculate an optimal CRP. This is what is known as the *Best Constant Rebalanced Portfolio* (BCRP) (Li and Hoi, 2014).

$$b* = \underset{b^n \in \triangle_m}{\arg\max} \log S_n(CRP(b)) = \underset{b \in \triangle_m}{\arg\max} \sum_{t=1}^{n} \log(b^\top x_t) \tag{2.5}$$

The final cumulative portfolio wealth and corresponding exponential growth rate achieved by BCRP is defined as:

$$S_n(BCRP) = \max_{b \in \triangle_m} S_n(CRP(b)) = S_n(CRP(b^*)) \tag{2.6}$$

$$W_n(BCRP) = \frac{1}{n}\log S_n(BCRP) = \frac{1}{n}\log S_n(CRP(b^*)) \tag{2.7}$$

2.3.4 | Follow the winner

The *'Follow-the-Winner'* PS approach aims to asymptotically achieve the same growth rate as that of an optimal strategy, often based on the Capital Growth Theory. This is characterised by increasing the relative weights of more successful assets at set periods (Li and Hoi, 2014).

2.3.4.1 | Exponential Gradient (EG)

One strategy considered in this field is *Exponential Gradient* (EG), proposed by Helmbold et al. (1996, 1998) based on the algorithm previously proposed for mixture estimation problem (Helmbold et al., 1997). This strategy focuses on the following optimisation problem:

$$b_{t+1} = \underset{b\in\triangle_m}{\arg\max} \quad \eta \log b \cdot x_t - R(b, b_t) \tag{2.8}$$

where $R(b, b_t)$ denotes a regularisation term and $\eta > 0$ is the learning rate. One straightforward interpretation of the optimisation is to track the stock with the best performance in last period but keep the new portfolio close to the previous portfolio. This is obtained using the regularisation term $R(b, b_t)$. The one key parameter for EG, the learning rate, has to be small to achieve the desired performance. However, as $\eta \longrightarrow 0$, its update approaches uniform portfolio. Due to this the EG reduces to UCRP (Li and Hoi, 2014).

2.3.4.2 | Universal Portfolios (UP)

Another strategy belonging to this field is *Universal Portfolios* (UP) proposed by Cover (1996). The strategy is coined the name *μ-Weighted Universal Portfolio*, μ denoting the distribution on the space of valid portfolio $\triangle_m$. As Li et al. (2012) describe, this strategy can be interpreted as a historical performance weighted average of all valid constant rebalanced portfolios.

$$b_{t+1} = \frac{\int_{\triangle_m} bS_t(b)d\mu(b)}{\int_{\triangle_m} S_t(b)d\mu(b)} \tag{2.9}$$

At the beginning of period $(t+1)$, one CRP manager's portfolio value is equal to $S_t(b)d\mu(b)$. The final portfolio value is the weighted average of CRP managers' wealth (Cover, 1996).

$$S_n(UP) = \int_{\triangle_m} S_n(b)s\mu(b) \tag{2.10}$$

2.3.5 | Follow the loser

'Follow-the-Loser' approaches are associated with the use of a mean-reversion strategy, in which wealth is transferred from winning assets to losers. Although contradictory, oftentimes this approach achieves significantly better performance than the former (Li and Hoi, 2014). Some strategies that follow this principle include *Passive Aggressive Mean Reversion* (PAMR) (Li et al., 2012), *Online Moving Average Reversion* (OLMAR) (Li and Hoi, 2012), *Weighted Moving Average Mean Reversion* (WMAMR) (Gao and Zhang, 2013), and *Robust Median Reversion* (RMR) (Huang et al., 2016).

2.3.5.1 | Passive Aggressive Mean Reversion (PAMR)

The main concept behind PAMR is the design of a loss function which reflects the mean reversion property (Li et al., 2012). Therefore, the loss will linearly increase if the expected return based on last price relative is larger than a threshold. Otherwise, the loss is zero. In particular, the ε-insensitive loss function for the t^{th} period is defined as,

$$\ell_\varepsilon(b; x_t) = \begin{cases} 0 & b \cdot x_t \leq \varepsilon \\ b \cdot x_t - \varepsilon & otherwise \end{cases} \tag{2.11}$$

where $0 \leq \varepsilon \leq 1$ is a sensitivity parameter to control the mean reversion threshold (Li et al., 2012). The next portfolio weights are obtained via the following optimisation problem, which aims to reduce the loss.

$$b_{t+1} = \arg\min_{b \in \triangle_m} \frac{1}{2} ||b - b_t||^2 \quad s.t. \quad \ell_\varepsilon(b; x_t) = 0 \tag{2.12}$$

The base idea of the strategy described is to exploit the single-period mean reversion. Due to this, PAMR suffers from drawbacks in risk management stemming from the possible absence of single period mean reversion (Li and Hoi, 2012).

2.3.5.2 | Online Moving Average Reversion (OLMAR)

Whilst PAMR implicitly assumes single-period mean reversion, OLMAR makes use of a multiple-period mean reversion in the form of Moving Average Reversion (Li and Hoi, 2012). The single period prediction method could possibly cause failure in certain cases (Li et al., 2012). The inclusion of simple moving average proved to solve this issue (Li and Hoi, 2012). The corresponding next price relative is therefore calculated as follows,

$$\hat{x}_{t+1}(w) = \frac{MA_t(w)}{p_t} = \frac{1}{w}\left(1 + \frac{1}{x_t} + ... + \frac{1}{\odot_{i=0}^{w-2} x_{t-i}}\right) \tag{2.13}$$

where p is the price vector that corresponds to the related x, w is the window size and $\odot$ denotes element-wise product (Li and Hoi, 2012).

2.3.5.3 | Weighted Moving Average Mean Reversion (WMAMR)

WMAMR is an OLPS approach created by Gao and Zhang (2013), that exploits the property of mean reversion during the recent window of time instead of single period mean reversion seen in PAMR. Due to this, WMAMR is very similar to OLMAR conceptually, but WMAMR also aims to reduce computational complexity. Gao and Zhang (2013) do this by making use of the weighted arithmetic average of stock price relative. This is defined as:

$$\tilde{x}_{t+1} = \sum_{i=1}^{w} \omega_i x_{t-i+1} \tag{2.14}$$

where $(\omega_1, ..., \omega_w)$ is a weighted vector. The optimisation problem proposed in this approach is an approximate solution of Equation 2.12 using standard techniques from convex analysis:

$$b_{t+1} = b_t - \tau_t(\tilde{x}_t - \bar{\tilde{x}}_t \cdot 1) \tag{2.15}$$

where $\bar{\tilde{x}}_t = \frac{\tilde{x}_t \cdot 1}{m}$ is the mean of the t^{th} stock price relative and denotes the market return, and τ_t is computed as:

$$\tau_t = \max\left\{0, \frac{\ell_{1,\varepsilon}}{||\tilde{x}_t - \bar{\tilde{x}}_t \cdot 1||^2}\right\} \tag{2.16}$$

2.3.5.4 | Robust Median Reversion (RMR)

RMR was created by Huang et al. (2016) to reduce estimation errors found in alternate mean reversion strategies, caused by noise in the data. RMR explicitly estimates next price vector via robust L_1-median estimator at the end of t^{th} period, that is, $\hat{p}_{t+1} = L_1 med_{t+1}(w) = \mu_{t+1}$, where w is a window size, and μ is calculated by solving *Fermat-Weber problem* (Weber, 1929):

$$\mu_{t+1} = \underset{\mu}{\arg\min} \sum_{i=0}^{w-1} ||p_{t-i} - \mu|| \tag{2.17}$$

L_1-median is the point with minimal sum Euclidean distance to k given price vectors. Therefore, the expected price relative with the estimator described becomes:

$$\hat{x}_{t+1}(w) = \frac{L_1 med_{t+1}(w)}{p_t} = \frac{\mu_{t+1}}{p_t} \tag{2.18}$$

Then RMR follows the similar portfolio optimisation method as OLMAR to learn an optimal portfolio (Li et al., 2012).

2.3.6 | Meta-Learning

Whilst the previously discussed approaches focus on a single strategy, *Meta-Learning Algorithms* use a combinatorial approach, where multiple strategies are amalgamated (Li and Hoi, 2014).

2.3.6.1 | Online Newton Step (ONS)

One example of a Meta-Learning approach is *Online Newton Step* (ONS), created by Agarwal et al. (2006). The algorithm takes three parameters, β and η for theoretical analysis, and δ as a heuristic tuning parameter. A uniform portfolio $p_1 = \frac{1}{n}1$ is utilised on the first period. For the rest of the periods, a Newton-based strategy is put to play:

$$p_t = \prod_{S_n}^{A_{t-1}} (\delta A_{t-1}^{-1} b_{t-1}) \tag{2.19}$$

where:

$$b_{t-1} = \left(1 + \frac{1}{\beta}\right) \sum_{r=1}^{t-1} \bigtriangledown [\log_\tau (p_\tau \cdot r_\tau)], \tag{2.20}$$

$$A_{t-1} = \sum_{r=1}^{t-1} - \bigtriangledown^2 [\log(p_\tau \cdot r_\tau)] + I_n, \tag{2.21}$$

and $\prod_{S_n}^{A_{t-1}}$ is the projection in the norm induced by A_{t-1}. n is the number of stocks. t is the trading period ($t = 1, ..., T$). r_t is the *price relative vector* for a trading period. p_t is the portfolio on day t, taking the form of a distribution on the n stocks. This is a point in the n-dimensional simplex S_n.

2.4 | Reinforcement Learning

> 'RL is the learning what to do - how to map situations to actions - so as to maximise a numerical reward signal' (Sutton and Barto, 1999)

Reinforcement learning makes use of *evaluative* feedback, which tells one *how well* they achieved your goal (Sutton and Barto, 1999). In contrast, the feedback received in supervised learning is *instructive* telling one *how* to achieve their goal (Sutton and Barto, 1999). The two most important distinguishing features of RL are:

- *Trial-and-error:* The learner must discover which actions generate the greatest rewards by trying them.

- *Delayed reward:* Actions completed by the learner, in some cases, might not necessarily have an immediate reward but a delayed one, achieved in future steps (Sutton and Barto, 1999).

RL aims to maximise a reward signal instead of finding hidden structures in unlabelled data (Sutton and Barto, 1999). RL algorithms have been successfully applied in a variety of applications, such as video games (Mnih et al., 2015, 2016), board games (Lillicrap et al., 2016), and robotics (Henderson et al., 2018).

2.4.1 | Markov Decision Process

Reinforcement Learning is typically modelled as a Markov Decision Process (MDP). An MDP is a directed graph with *state* nodes, S, *action* nodes A, and directed edges connecting states to actions and actions to states (Greaves, 2017). To understand the role of the state and action nodes, we borrow a figure by Greaves (2017) providing a basic example of an MDP (Figure 2.1).

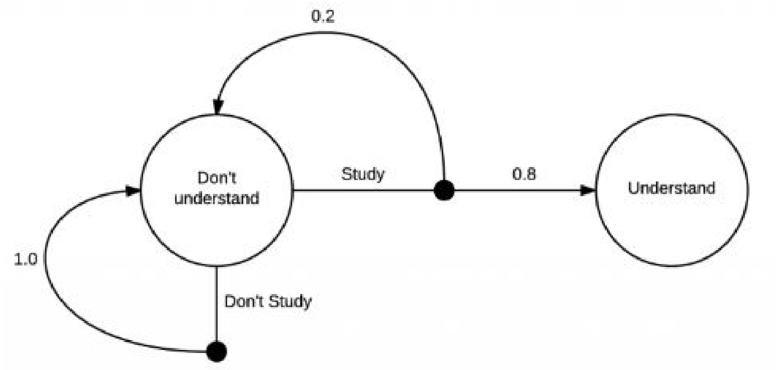


Figure 2.1: A simple MDP (Greaves, 2017).

Starting from *Do not understand* state, one has two possible actions. These are *Study* or *Don't Study*. The *Don't Study* option, when chosen, has a 100% chance to send one back to the initial state. On the other hand, the *study* state, has only a 20% chance of leading one back to the initial state, and an 80% chance leading towards the *Understand* state. The goal of RL is to learn how to spend more and more time within the most valuable states. For valuable states, rewards are included in state transitions MDP (Figure 2.2) (Greaves, 2017).

In Figure 2.2, negative rewards are received when transitioning to the *Hungry* state, and a considerable negative reward is received when transitioning to the *Starving* state. Transitioning into the *Full* state, however, grants a positive reward.

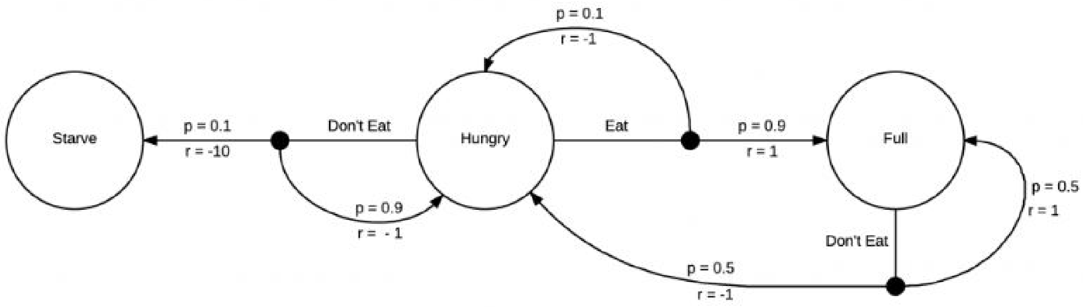


Figure 2.2: A simple MDP with rewards (Greaves, 2017).

2.4.2 | Bellman Equation

The *Bellman equation* is a central element in RL algorithms, allowing for optimal policies to be found and the value function to be measured Sutton and Barto (1999). The *value function* measures how good it is for the agent to be in the current state s. According to the Bellman equation, the value function is composed of the immediate reward (R_{t+1}) and discounted value of successor states (Bellman, 1954). The Bellman equation for value function is therefore:

$$v(s) = \mathbb{E}\left[R_{t+1} + \gamma v(S_{t+1}) | S_t = s\right] \tag{2.22}$$

where S_t is the current state, S_{t+1} is the state the agent is moving to, and γ is the discount value ($0 \leq \gamma \leq 1$). Given that multiple possible state transitions could be present, the probability p is included in the equation. Therefore, $v(s)$ is the value of the current state, which is equal to the summation of the immediate reward, and the value of the next state $v(s')$ which is discounted by γ and multiplied by its transition probability p.

A *policy* is a function which defines the probability distribution over *actions* for each *state*. Therefore, $\pi(a|s)$ is the probability of the agent taking action a, given policy π. Furthermore, the *state-action value function* or *Q-Function*, measures how good it is for an action a to be taken on the current state s, given a policy π. This is defined as:

$$q_\pi(s,a) = \mathbb{E}\left[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \middle| S_t = a, A_t = a\right] \tag{2.23}$$

From the above equation for the state-action value function, we can see that it is composed of the immediate reward given on performing a certain action from state s to go to state s', added with the discounted value of the state-action value of the s' with respect to the action a taken from that state on-wards.

2.4.3 | Key concepts and terminology

Whereas MDP assumes that a model of the environment is known, RL allows for learning by experience without the need of knowing the model. The main characteristics of RL are the *environment* and the *agent* that interacts with it. At each step of interaction, the learning agent senses the *state* of its environment and must be able to take *actions* that affect the state (Kanwar, 2019). In addition, the agent perceives a numerical *reward* signal from the environment that represents how good or bad the current environment state is based on explicit goals (Sutton and Barto, 1999). Figure 2.3 depicts this agent-environment interaction. At some time step t, the agent is in state s_t and takes an action a_t. The environment iterates one step forward and replies, giving a new state s_{t+1} and a reward r_{t+1} (Sutton and Barto, 1999).

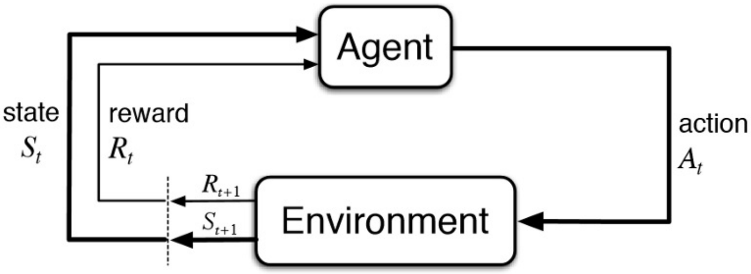


Figure 2.3: RL Setting (Sutton and Barto, 1999).

2.4.3.1 | State

A *state* in RL conveys the current situation of the environment to the agent, allowing it to choose an action based on it. States can take various forms, consisting from low-level readings to high-level abstract ones (Sutton and Barto, 1999). The *state space*, on the other hand, is the set of all possible states that can be presented to the agent.

2.4.3.2 | Reward and return

RL agents learn to maximise *cumulative future reward*, which is known as Return R (Greaves, 2017). R_t is the return from the current time step, defined as:

$$R_t = r_{t+1} + r_{t+2} + r_{t+3} + r_{t+4} + ... = \sum_{k=0}^{\infty} r_{t+k+1} \tag{2.24}$$

However, if we had an infinite series with the equation above, we would end up with an infinite return. Therefore, this equation would only be considered if we expect the series to be *episodic*, always terminating (Greaves, 2017). Due to this, *future cumulative discounted reward* (Equation 2.25) is more commonly used than *future cumulative reward*.

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + ... = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{2.25}$$

where γ is the discount value ($0 \leq \gamma \leq 1$). According to Greaves (2017), two benefits of defining return this way are that:

- the return is defined with an asymptotic upper bound, and
- imminent rewards are given greater weight than those received further in the future. This is adjusted with the γ value.

2.4.3.3 | Action spaces

Different problems have different environments, and in turn, these allow for various forms of actions (Kanwar, 2019). The *action space* is the set of all valid actions in a given environment. For example, a board game such as Go has a finite set of actions based on the game rules. This is what is known as a discrete action space. In contrast, other environments, such as those used for OLPS, have continuous action spaces.

2.4.3.4 | Policies

The policy π determines the behaviour of the RL agent and the action it takes at any point in time (Sutton and Barto, 1999). Formulated as a function, it takes a state s as input and returns an action a. Therefore, $\pi(s) \rightarrow a$. Our goal in reinforcement learning is to learn an optimal policy, π^*, which tells us how to maximise return in every state (Sutton and Barto, 1999).

2.4.3.5 | Exploration and exploitation trade-off

One unique challenge not present in other kinds of machine learning is the trade-off between exploration and exploitation when taking this action. The dilemma is caused because pursuing exploration or exploitation exclusively would surely fail at the task. The agent must explore and try various actions whilst progressively favour those that appear to provide the greatest reward and value (Sutton and Barto, 1999).

2.4.4 | A Taxonomy of Reinforcement Learning Algorithms

RL algorithms are mainly classified by the concept of providing the agent with access to a model of the environment. These are called model-free and model-based RL (Liang

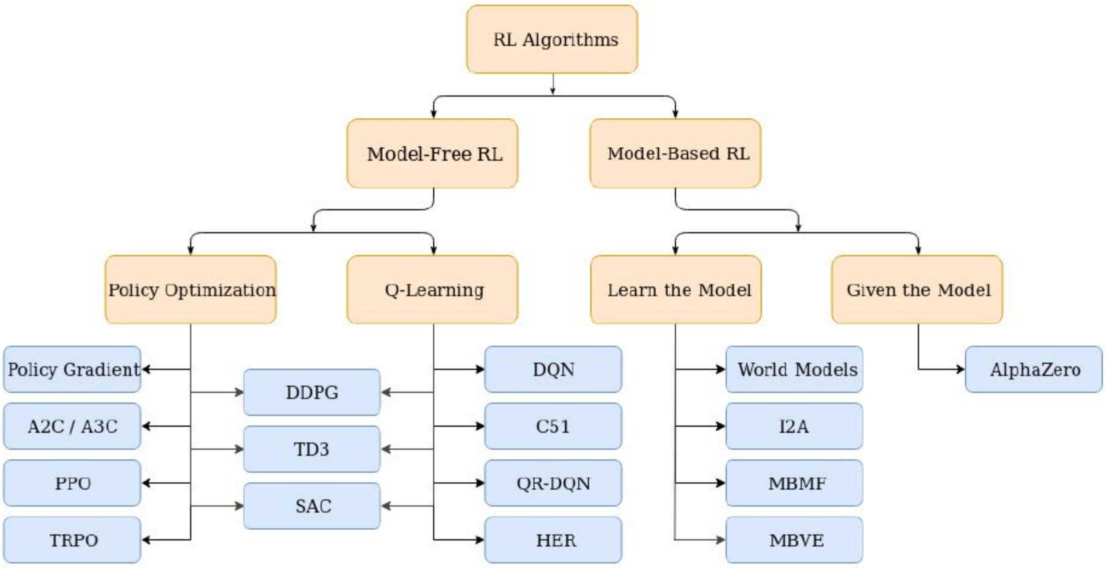


Figure 2.4: Taxonomy of RL Algorithms Kanwar (2019)

et al., 2018). This can be seen in the first branch in Figure 2.4. In this work, we are mainly concerned with model-free RL due to the dynamic nature of our domain. The behaviour of the agent is defined by its policy. Two main forms of algorithms are described as *On-Policy* and *Off-Policy*, creating two varieties of model-free RL. *Policy Optimisation*, and *Q-learning* are two approaches that represent and train agents with model-free RL, and are almost always done on-policy and off-policy, respectively (Kanwar, 2019).

2.4.4.1 | Policy Optimisation

Methods in this family represent a policy explicitly as $\pi_\theta(a|s)$. They optimise the parameters θ either directly by gradient ascent on the performance objective $J(\pi_\theta)$, or indirectly, by maximising local approximations of $J(\pi_\theta)$ (Kanwar, 2019). An example of a performance objective is to maximise the expected return, $J(\pi_\theta) = \underset{r\sim\pi_\theta}{E}[R(r)]$. Since these are *on-policy*, each update only uses data collected while acting according to the most recent version of the policy. In most cases, approximator $V_\phi(s)$ for the on policy value function $V^\pi(s)$ is learned to discover how to update the policy Kanwar (2019). Algorithms in this model-free, on-policy branch of RL include *Actor Critic Methods* (A2C/A3C) (Kang et al., 2018), which performs gradient ascent to directly maximise performance, and *Proximal Policy optimisation* (Schulman et al., 2017), whose updates indirectly maximise performance, by instead maximising a surrogate objective function that gives a conservative estimate for how much $J(\pi_\theta)$ will change as a result of the update (Kanwar, 2019).

2.4.4.2 | Q-learning

In Q-learning, on the other hand, an agent learns optimal policy with the help of a greedy policy and behaves using the policies of other agents. The updated policy is different from the behaviour policy, making Q-Learning off-policy (Sutton and Barto, 1999). An approximator $Q_\theta(s,a)$ for the optimal action-value function, $Q^*(s,a)$, is learned using an objective function based on the Bellman equation (Bellman, 1954). An example of a Q-Learning method is Deep Q-Network.

2.4.4.3 | Interpolating Between Policy Optimisation and Q-Learning

These are standard approaches discussed in papers, but alternate and combinatorial methods exist. These algorithms attempt to balance the trade-off between the strengths and weaknesses of both branches (Kanwar, 2019). Two such algorithms are DDPG (Silver et al., 2014) and TD3 (Fujimoto et al., 2018).

2.5 | Deep Reinforcement Learning

DRL is the combination of RL with Deep Learning from Deep Artificial Neural Networks (DANNs), extending to tasks with high-dimensional input and action spaces (Khadka et al., 2019). Tasks with these qualities would consist of a large state space, making approaches such as Q-learning not viable. This is resolved with a function approximator by using DANNs. These are discussed in Section 2.6. DANNs are able to handle more complex environments due to the inclusion of additional hidden layers within the ANN, and hence when applied to RL, this allows it to be applied to larger problems (Henderson et al., 2018). This is visualised in Figure 2.5. Deep learning (DL) has witnessed its rapid progress in speech recognition (Noda et al., 2015) and image identification (Liu et al., 2020), and has shown its capability to capture complex, non-linear patterns (Liang et al., 2018). Despite these being very powerful algorithms, they are not without their challenges. Khadka et al. (2019) describe three major challenges found when applying such algorithms to a real-world problem. First, in many real-life scenarios, rewards are an uncommon occurrence. This is often referred to as the temporal credit assignment problem (Sutton and Barto, 1999). Also, a lack of diverse exploration done with a DRL algorithm may cause premature converging, and finally, such methods tend to be sensitive to the choice of their hyperparameters (Henderson et al., 2018).

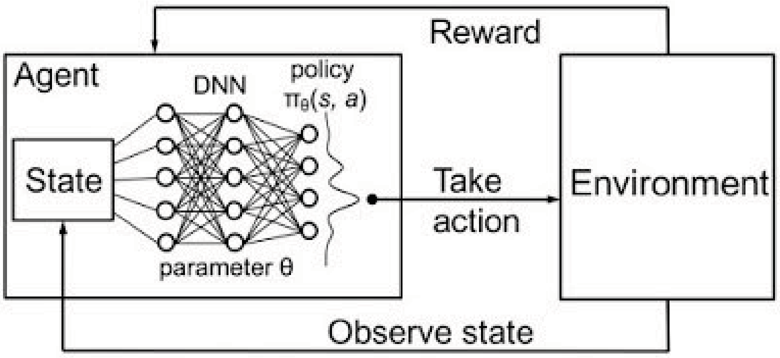


Figure 2.5: DRL Setting from Hu and Lin (2019).

2.5.1 | Deep Deterministic Policy Gradient (DDPG)

DDPG and its variants have been frequently used for this problem domain in recent literature (Gran, 2019; Hegde et al., 2019; Kanwar, 2019; Zhang et al., 2020). DDPG is an algorithm devised by Google DeepMind (Lillicrap et al., 2016; Silver et al., 2014) to tackle the continuous action space problem. It is a policy gradient based algorithm which makes use of a stochastic behaviour policy to aid exploration but estimates a deterministic target policy. This, as Lillicrap et al. (2016) describe, facilitates the learning process. The algorithm concurrently learns a Q-function and a policy, as off-policy data and the Bellman equation are used to learn the Q-function, and in turn, the Q-function is used to learn the policy. In their second experiment concerned with continuous reinforcement learning, Silver et al. (2014) compare the deterministic approach against its stochastic counterpart using standard reinforcement learning benchmark environments of the time (mountain car, pendulum and 2D puddle world). Likewise, in (Lillicrap et al., 2016), their DDPG approach is implemented on standard reinforcement learning benchmark environments to be adequately compared to other approaches. In continuous spaces, actions are real-valued vectors (Kanwar, 2019). Implementing Q-learning for such an action space is intractable when the said action space is large, due to the *"curse of dimensionality"*, which implies that with a fixed number of training samples, predictive power first increases steadily along with the number of dimensions or features used. However, beyond a certain number of dimensions, predictive power instead starts deteriorating (Liang et al., 2018; Trunk, 1979). Overestimation bias is a property of Q-learning in which the maximisation of a noisy value estimate induces a consistent overestimation (Thrun and Schwartz, 1993).

The creation of DDPG was inspired by the advancements in Deep Q-network (DQN) methodologies, and thus is closely connected to DQN. If the optimal action-value function $Q^*(s,a)$ is known, with any given state the best action $a^*(s)$ can be found.

$$a^*(s) = arg \max_a Q^*(s,a) \tag{2.26}$$

To compute the max over actions $\max_a Q^*(s,a)$ in a continuous action space, the function $Q^*(s,a)$ is presumed to be differentiable with respect to he action argument. This way an efficient, gradient-based learning rule for a policy $\mu(s)$ can be set up making use of this approximation (Kanwar, 2019).

$$\max_a Q(s,a) \approx Q(s,\mu(s)) \tag{2.27}$$

The Bellman equation for the optimal action-value function $Q^*(s,a)$ is defined as:

$$Q^*(s,a) = \underset{s'\sim P}{E}[r(s,a) + \gamma \max_{a'} Q^*(s',a')] \tag{2.28}$$

where $s' \sim P$ is the next state s' sampled from the environment from a distribution $P(.|s,a)$. Equation (2.28) is the starting point for learning an approximator $Q^*(s,a)$. Let us consider a scenario with a Predictor in the form of a neural network $Q_\phi(s,a)$, where ϕ are its parameters and a collected set of transitions D consisting of (s,a,r,s',d). Here, d is an indicator on whether s' is terminal with a Boolean, 0 or 1 signal. A mean-squared Bellman error (MSBE) function tells us roughly how closely Q_ϕ comes to satisfying the Bellman equation. The aim is to minimise this MSBE during training.

$$L(\phi, D) = \underset{(s,a,r,s',d)\sim D}{E}\left[\left(Q_\phi(s,a) - (r + \gamma(1-d)\max_{a'} Q_\phi(s',a'))\right)^2\right] \tag{2.29}$$

2.5.1.1 | Experience replay

As in DQN, DDPG makes use of a replay buffer. These models train using mini-batches to sample experience to update neural network parameters and use a Replay Buffer. The frameworks cannot optimise a sequential decision process in an on-policy way. Instead, data has to be independently distributed. The replay buffer is a finite-sized cache on which tuples, consisting of this state, action, reward and the next state, are stored (Lillicrap et al., 2016). When full, the oldest samples are discarded. The actor and critic are updated at each timestep by sampling a mini-batch uniformly from the buffer.

2.5.1.2 | Target networks

The aim of minimising the MSBE loss, is to make the Q-function be more like the target.

$$r + \gamma(1-d)\max_{a'} Q_\phi(s',a') \tag{2.30}$$

The target depends on the same parameters we are trying to train, ϕ, which makes MSBE minimisation unstable. A solution to this is to utilise a set of parameters. These come close to ϕ but with a time delay via a target network. The parameters of the target network are denoted as ϕ_{targ}. In DDPG, the target network is updated once per main network update by Polyak averaging (Polyak and Juditsky, 1992).

$$\phi_{targ} \leftarrow p\phi_{targ} + (1-p)\phi \tag{2.31}$$

where p is a hyperparameter between 0 and 1, but is usually set close to 1. Therefore, the formula for calculating MSBE loss $L(\phi, D)$ in Figure (2.29) can be adjusted:

$$\underset{(s,a,r,s',d)\sim D}{E}\left[\left(Q_\phi(s,a) - \left(r + \gamma(1-d)Q_{\phi_{targ}}(s', \mu_{\theta_{targ}}(s'))\right)\right)^2\right] \tag{2.32}$$

where $\mu_{\theta_{targ}}$ is the target policy.

2.5.1.3 | Exploration vs. Exploitation

From the policy learning side, DDPG aims to learn a deterministic policy $\mu_\theta(s)$ which gives the action that maximises $Q_\phi(s,a)$. Due to the action space being continuous and the Q-function is differentiable with respect to action, one can simply perform gradient ascent.

$$\max_\theta \underset{s\sim D}{E}\left[Q_\phi(s, \mu_\theta(s))\right] \tag{2.33}$$

where the Q-function parameters are treated as constants.

Noise is added to the actions at training time to enhance the DDPG policy exploration. The Ornstein-Uhlenbeck method (Uhlenbeck and Ornstein, 1930) is generally the chosen one to generate noise in literature (Lillicrap et al., 2016). The pseudocode for the DDPG framework is defined in Algorithm 1.

2.5.2 | Overestimation Bias

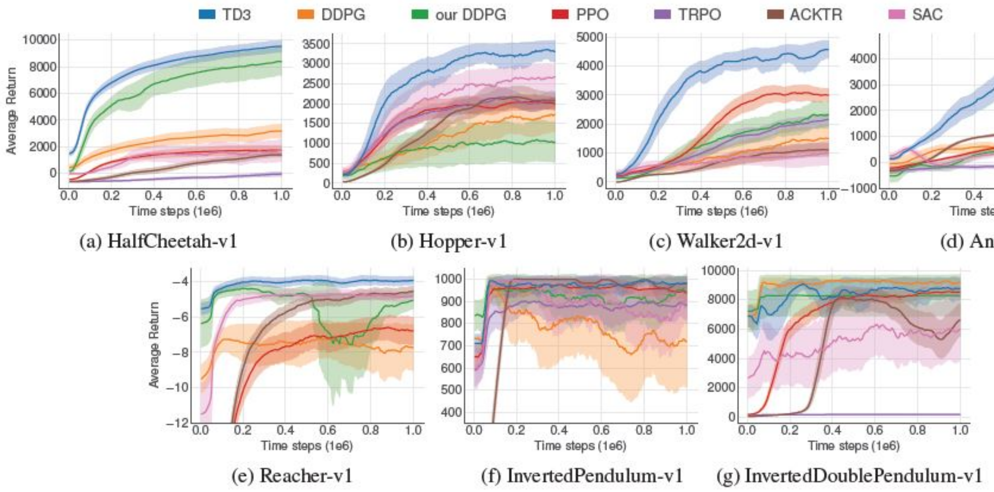
If the target being estimated (Equation 2.30) is susceptible to error ϵ, the maximum over the value along with its error will be of a greater value than the true maximum (Thrun and Schwartz, 1993). Due to this, even initially, value updates resulting in consistent overestimation bias can be caused by a zero-mean error. This would then be propagated onward through the Bellman equation. This is problematic as errors induced by function approximation are unavoidable. Despite the minimal nature that the overestimation may have, two concerns are raised by the presence of error (Fujimoto et al., 2018). These are the potential of overestimation developing into more significant bias as training episodes go by, and poor policy updates due to inaccurate value estimates.

Algorithm 1 DDPG (Lillicrap et al., 2016; Silver et al., 2014)

```
Randomly initialise critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights $\theta^Q$ and $\theta^\mu$.
Initialise target network $Q'$ and $\mu'$ with weights $\theta^{Q'} \leftarrow \theta^Q, \theta^{\mu'} \leftarrow \theta^\mu$
Initialise replay buffer $R$
for episode = 1, M do
    Initialise a random process $N$ for action exploration
    Receive initial observation state $s_1$
    for t=1, T do
        Select action $a_t = \mu(s_t|\theta^\mu) + N_t$ according to the current policy and exploration noise
        Execute action $a_t$ and observe reward $r_t$ and observe new state $s_{t+1}$
        Store transition $(s_t, a_t, r_t, s_{t+1})$ in $R$
        Sample a random minibatch of $N$ transitions $(s_i, a_i, r_i, s_{i+1})$ 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 policy gradient:
        $\nabla_{\theta^\mu} J \sim \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$
        Update the target networks:
        $\theta^{Q'} \leftarrow r\theta^Q + (1-r)\theta^{Q'}$
        $\theta^{\mu'} \leftarrow r\theta^\mu + (1-r)\theta^{\mu'}$
    end for
end for
```

2.5.3 | Twin Delayed Deep Deterministic Policy Gradient (TD3)

TD3 (Fujimoto et al., 2018) is a DRL model which combines Policy gradient, Actor-Critics, and continuous Double Deep Q-Learning (Dankwa and Zheng, 2019). TD3 builds on top of Double Q-Learning (as seen in (Patel, 2018)) to reduce overestimation bias in Deep Q-Learning. Fujimoto et al. (2018) perform an experiment to prove that the theoretical overestimation occurs in practice when training DDPG models, and that although Double Q-Learning is more effective than conventional Q-Learning, it still does not eliminate overestimation bias. In their approach, they overcome this by making use of a novel method, *Clipped Double Q-learning* Van Hasselt (2010), to replace the critic in the actor-critic framework. This method is found to decrease significantly the overestimation done by the critic (Fujimoto et al., 2018). Fujimoto et al. (2018) introduce a new DRL framework, TD3, utilising this alternate method described, along with *Delayed Policy Updates* and *Target Policy Smoothing Regularisation*. The proposed TD3 framework is compared with a variety of alternate frameworks, including DDPG, on a number of OpenAI gym environments. The results acquired can be seen in Figure 2.6 and Table 2.1.



(a) HalfCheetah-v1 (b) Hopper-v1 (c) Walker2d-v1 (d) Ant-v1

(e) Reacher-v1 (f) InvertedPendulum-v1 (g) InvertedDoublePendulum-v1

Figure 2.6: Learning curves achieved by the DRL models during training on OpenAI gym continuous control tasks. The shaded regions represent half a standard deviation of the average evaluation over ten trials (Fujimoto et al., 2018). The 'our DDPG' framework is a separate implementation of DDPG by Fujimoto et al. (2018) in which the critic receives both the state and action as input to the first layer.

Environment	TD3 (Fujimoto)	DDPG	Our DDPG (Fujimoto)	PPO	TRPO	ACKTR	SAC
HalfCheetah	9636.95 ± 859.065	3305.60	8577.29	1795.43	-15.57	1450.46	2347.19
Hopper	3564.07 ± 114.74	2020.46	1860.02	2164.70	2471.30	2428.39	2996.66
Walker2d	4682.82 ± 539.64	1843.85	3098.11	3317.69	2321.47	1216.70	1283.67
Ant	4372.44 ± 1000.33	1005.30	888.77	1083.20	-75.85	1821.94	655.35
Reacher	-3.60 ± 0.56	-6.51	-4.01	-6.18	-111.43	-4.26	-4.44
InvPendulum	1000.00 ± 0.00	1000.00	1000.00	1000.00	985.40	1000.00	1000.00
InvDouble- Pendulum	9337.47 ± 14.96	9355.52	8369.95	8977.94	205.85	9081.92	8487.15

Table 2.1: TD3 results obtained by Fujimoto et al. (2018). The table consists of the maximum average return over 10 trials of 1 million time steps. ± corresponds to a single standard deviation over trials (Fujimoto et al., 2018).

Further Studies done by Dankwa and Zheng (2019) compare TD3 to DDPG, Proximal Policy Optimisation (PPO), Trust Region Policy Optimisation (TRPO), Actor-Critic using Kronecker-factored Trust Region (ACKTR) and Soft Actor-Critic (SAC) on the MuJoCo pybullet continuous control environment. The TD3 model achieved a higher Average Reward when compared to the other state-of-the-art models (Dankwa and Zheng, 2019). Many have opted for this framework over others in various other domains requiring continuous control, including Li and Yu (2020) and MacHalek et al. (2020). Li and Yu (2020) utilise TD3 in their framework to potentially improve both control performance and economy in a power grid with multiple continuous power disturbances. MacHalek et al. (2020) consider DDPG, TD3 and PPO techniques for dynamic economic optimisation of a continuously stirred tank reactor. In their results, all of the techniques mentioned above effectively optimise the system, but only TD3 demonstrated convergence to a near-optimal solution in the training curves.

2.5.3.1 | Clipped Double Q-Learning for Actor-Critic

As Fujimoto et al. (2018) indicate, several approaches have been proposed to reduce overestimation bias but are found to be ineffective in an actor-critic setting. A variant of Double Q-learning (Van Hasselt, 2010), *Clipped Double Q-learning*, is introduced by Fujimoto et al. (2018) to replace the critic in any actor-critic method. In Double Q-learning, two separate value estimates are maintained. Each of these is used to update the other. With independent value estimates, unbiased estimates of the actions selected using the opposite value estimate can be made (Fujimoto et al., 2018). Double DQN, created by Van Hasselt et al. (2016), makes use of the target network as one of the two value estimates, to then obtain a policy by greedy maximisation of the current value network instead of the target network. Therefore, in an actor-critic setting, analogous updates in the learning target make use of the current policy rather than the target policy:

$$y = r + \gamma Q_{\theta'}(s', \pi_\phi(s')) \tag{2.34}$$

where r is the reward received, s' is the new state of the environment, γ is a discount factor determining the priority of short-term rewards, π_ϕ is the optimal policy with parameters ϕ, and $Q_{\theta'}$ is the function approximator with parameters θ. In their study, Fujimoto et al. (2018) find that within their actor-critic model, the current and target networks were too similar and thus did not offer much improvement.

The original Double Q-learning formulation consists of a pair of actors (π_{ϕ_1}, π_{ϕ_2}) and critics (Q_{θ_1}, Q_{θ_2}), where π_{ϕ_1} is optimised with respect to Q_{θ_1} and π_{ϕ_2} with respect to Q_{θ_2}:

$$\begin{aligned} y_1 &= r + \gamma Q_{\theta'_2}(s', \pi_{\phi_1}(s')) \\ y_2 &= r + \gamma Q_{\theta'_1}(s', \pi_{\phi_2}(s')) \end{aligned} \tag{2.35}$$

In (Fujimoto et al., 2018), Double Q-learning is found to be more effective than Double DQN but does not eliminate the overestimation bias. Moreover, Fujimoto et al. (2018) note that the critics are not entirely independent, as the learning targets use the opposite critic and the same replay buffer. This, in turn, may cause the overestimation to be exaggerated in certain areas of the state space.

Clipped Double learning addresses this problem with an upper-bound on the less biased value estimate Q_{θ_2} by the biased estimate Q_{θ_1}. Therefore the minimum between the two estimates is chosen to give the target update:

$$y_1 = r + \gamma \min_{i=1,2} Q_{\theta'_i}(s', \pi_{\phi_1}(s')) \tag{2.36}$$

This way, the value target cannot introduce any additional overestimation over the standard Q-learning target. In turn, it may induce some element of underestimation bias, which is preferable to overestimation bias. This is due to underestimated actions not explicitly propagated through the policy update.

2.5.3.2 | Delayed Policy Updates

The work done by Fujimoto et al. (2018) shows how the growth error discussed can be reduced with a stable target. The learning behaviour with and without the target networks is examined on both the critic and the actor. The results suggest that failure may occur in the interplay of the updates done by the actor and critic. A value estimate would deviate because of overestimation with a poor policy, and in turn, the policy would become poor due to inaccurate value estimates (Fujimoto et al., 2018). Due to this, Fujimoto et al. (2018) suggests that the policy network would be updated at a lower frequency than that of the value network to first minimise error before introducing a policy update. This is done with a modification which allows the update of the policy and target networks to be done only after a fixed number of updates d to the critic, and updating the target networks slowly $\theta' \leftarrow r\theta + (1 - r)\theta'$. By delaying the policy updates, repeating updates with respect to an unchanged critic are avoided.

2.5.3.3 | Target Policy Smoothing Regularisation

Deterministic policies are known to occasionally overfit to narrow peaks in the value estimate, due to their high susceptibility to inaccuracies caused by function approximation error. Regularisation can be used to reduce the target variance induced by the approximation error. Fujimoto et al. (2018) introduces a regularisation strategy within the TD3 framework, called target policy smoothing. Target policy smoothing is very similar to the learning update from SARSA (Sutton and Barto, 1999), and enforces the idea that similar actions should have similar value. By modifying the training procedure, the relationship between actions with similarities can be created. Fujimoto et al. (2018) propose that fitting the value of a small area around the target action would have the benefit of smoothing the value estimate by bootstrapping similar state-action value estimates.

$$y = r + \mathbb{E}_{\epsilon}[Q_{\theta'}(s', \pi_{\phi'}(s') + \epsilon)] \tag{2.37}$$

This expectation over actions can be approximated in practice, with the addition of random noise to the target policy and averaging over mini-batches. The modified target update is as below:

$$\begin{aligned} &y = r + \gamma Q_{\theta'}(s', \pi_{\phi'}(s') + \epsilon), \\ &\epsilon \sim clip(N(0, \sigma), -c, c) \end{aligned} \tag{2.38}$$

where the added noise is clipped to keep the target in a small range. The full pseudocode for TD3 is available in Algorithm 2.

Algorithm 2 TD3 (Fujimoto et al., 2018)

```
Initialise critic networks $Q_{\theta_1}, Q_{\theta_2}$, and actor network $\pi_\phi$ with random parameters $\theta_1, \theta_2, \phi$
Initialise target networks $\theta'_1 \leftarrow \theta_1, \theta'_2 \leftarrow \theta_2, \phi' \leftarrow \phi$
Initialise replay buffer B
for t = 1 to T do
    Select action with exploration noise $a \sim \pi(s) + \epsilon$, $\epsilon \sim N(0, \sigma)$ and observe reward $r$ and new state $s'$
    Store transition tuple $(s, a, r, s')$ in B
    Sample mini-batch of $N$ transition $(s, a, r, s')$ from B
    $\tilde{a} \leftarrow \pi_{\phi'}(s) + \epsilon, \epsilon \sim \text{clip } (N(0, \tilde{\sigma}), -c, c)$
    $y \leftarrow r + \gamma \min_{i=1,2} Q_{\theta'_i}(s', \tilde{a}$
    Update critics $\theta_i \leftarrow \min_{\theta_i} N^{-1} \sum (y - Q_{\theta_i}(s, a))^2$
    if t mod d then
        Update $\phi$ by the deterministic policy gradient:
        $\bigtriangledown_\phi J(\phi) = N^{-1} \sum \bigtriangledown_a Q_{\theta_1}(s, a)|_{a=\pi_\phi(s)} \bigtriangledown_\phi \pi_\phi(s)$
        Update target networks:
        $\theta'_i \leftarrow r\theta_i + (1 - r)\theta'_i$
        $\phi' \leftarrow r\phi + (1 - r)\phi'$
    end if
end for
```

2.6 | Deep Learning & Function Approximators

2.6.1 | Background on Neural Networks

An ANN consists of simple processing units called neurons, and the weighted connections between them (w_{ij}, where i and j are neurons). Data is transferred between neurons through these connections, with the corresponding connecting weight being either excitatory or inhibitory (Patterson and Gibson, 2017). A neuron j receives the outputs $(o_i, ..., o_{i_n})$ of other neurons $(i_1, i_2, ..., i_n)$ which are connected to it, and are passed to the *propagation function*. This transforms them according to the connecting weights w_{ij} into the *network input net j*. This can be further processed by the *activation function*. The *weighted sum* is an example of one such propagation function:

$$net_j = \sum_{i \in I}(o_i \cdot w_{ij}) \tag{2.39}$$

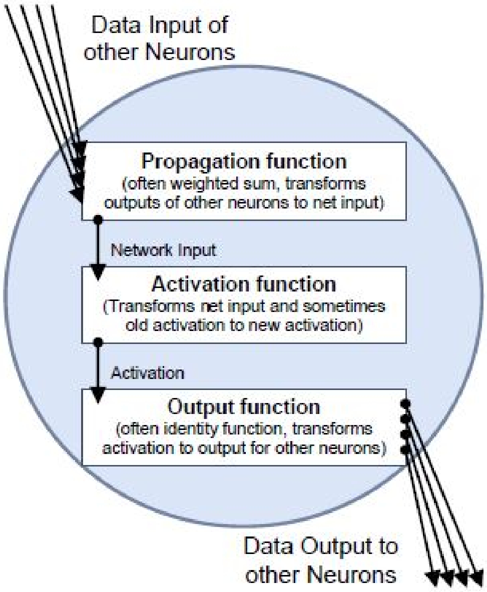

Data Input of
other Neurons
Propagation function
(often weighted sum, transforms
outputs of other neurons to net input)
Network Input
Activation function
(Transforms net input and sometimes
old activation to new activation)
Activation
Output function
(often identity function, transforms
activation to output for other neurons)
Data Output to
other Neurons

Figure 2.7: Data processing of a neuron. The activation function of a neuron implies the threshold value. (Kriesel, 2007)

The reactions of the neurons to the input values depend on the *activation state* (a_j on neuron j), which indicates the extent of the neuron's activity. This is a result of the *activation function*. A threshold value Θ_j is uniquely assigned to a neuron j that marks the position of the maximum gradient value of the activation function. The activation function is defined as:

$$a_j(t) = f_{act}(net_j(t), a_j(t-1), \Theta_j) \tag{2.40}$$

The activation function transforms the network input net_j as well as the previous activation state $a_j(t-1)$ into a new activation state $a_j(t)$ (Kriesel, 2007). Unlike the threshold values, the activation function is often defined globally for all neurons in a layer. Some examples of activation functions include *Sigmoid*, *Tanh*, *ReLU*, and *Softmax* (Kriesel, 2007). The activation state may then be processed by an *output function*. The output function of a neuron j calculates the values which are transferred to the other neurons connected to j:

$$f_{out}(a_j) = o_j \tag{2.41}$$

The output function is generally defined globally, and is often set to the *identity*, which implies that $f_{out}(a_j) = a_j$, so $o_j = a_j$.

Neural networks may be designed in various ways using the previously described neural network elements. Kriesel (2007) describes some of the usual topologies of neural networks, including Feedforward, Recurrent and Completely linked networks.

2.6.1.1 | Recurrent Networks

The process of a neuron influencing itself is called *Recurrence*. Recurrent networks do not necessarily have the input and output neurons explicitly defined. This is why the neurons are numbered in Figure 2.8 which depicts such a network. The recurrences present in the network may be *direct* by having neurons connected to themselves, or *indirect* by allowing connections directed towards the input layer. Additionally, recurrences may also be *lateral* by connecting neurons within the same layer. In such a case, each neuron generally inhibits the other neurons in the corresponding layer and strengthens itself, resulting in a *winner-takes-all scheme* in which only the strongest neuron becomes active.

2.6.2 | Deep Learning Networks

As Patterson and Gibson (2017) describe, the facets that differentiate deep learning networks from the ’canonical’ feedforward multilayer networks consist of:

- **Increased number of neurons**, to express more complex models.
- **More complex connections between neurons**, using recurrence and locally connected patches.

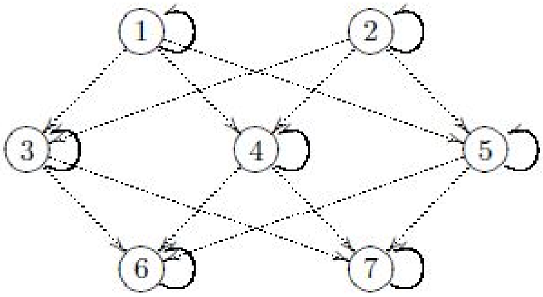


Figure 2.8: Recurring network example from (Kriesel, 2007) with direct recurrence.

2.6.3 | Major Architectures of Deep Networks

Emmert-Streib et al. (2020) review a number of major architectures of deep networks. These include Deep Feedforward Neural Networks (D-FFNN), Convolutional Neural Networks (CNNs), Deep Belief Networks (DBNs), Autoencoders (AEs), and Long Short-Term Memory (LSTM). The latter is a variant of a RNN (briefly described in section 2.6.1.1) created to enhance the handling of long-term dependencies (Hochreiter and Schmidhuber, 1997). This architecture has proved to be useful for sequential modelling (Patterson and Gibson, 2017).

2.6.3.1 | Deep Networks in related studies

Due to the complex and non-stationary nature of asset prices, related studies make use of Deep Network architectures such as D-FFNN (Kanwar, 2019), RNN (Jiang et al., 2017), CNN (Jiang et al., 2017; Kanwar, 2019; Vogiatzis, 2019) and LSTM (Hegde et al., 2019; Jiang et al., 2017; Patel, 2018; Zhang et al., 2020), to extract patterns. The latter two architectures have been most prevalent throughout recent studies, with LSTM having the most consistent success. On the other hand, although CNN has been successful in (Jiang et al., 2017), another more recent study by Kanwar (2019) found the performance of their implementation to be inconsistent. The success of the LSTM architecture in this domain may be attributed to their ability of sequential modelling and handling of long-term dependencies (Patterson and Gibson, 2017), and reoccurring patterns in portfolio assets Li et al. (2012).

2.6.4 | Long Short-Term Memory Neural Networks

The main appeal of RNNs is their ability to connect previous information to the present task using recurrence (Olah, 2015). LSTMs were built to enhance this ability to handle long-term dependencies, which the traditional RNN had lacked (Hochreiter, 1991).

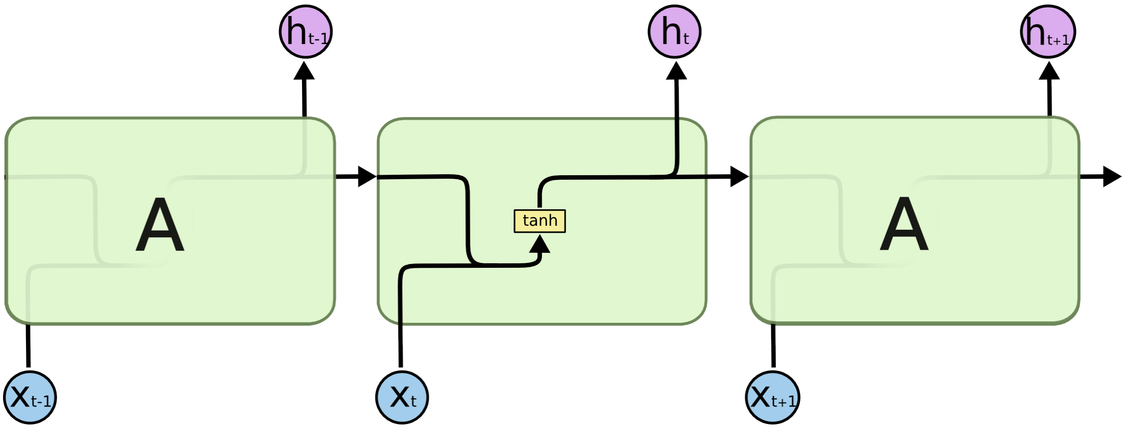


Figure 2.9: The repeating module in a standard RNN contains a single layer (Olah, 2015).

When unrolled, all RNNs take the form of a chain of repeating modules. In standard RNNs, this recurring module will have a very simple structure, such as a single *tanh* layer as depicted in Figure 2.9 (Olah, 2015). In the diagram, A represents a chunk of the neural network, the corresponding x and h are its input and output, respectively. LSTMs possess this chain-like structure, but with an enhanced repeating module that consists of four neural network layers as depicted in Figure 2.10 (Olah, 2015).
An entire vector is carried along each line, linking one node to another, from the output of the former to the input of the latter. In Figure 2.10, wherever the lines merge, the vectors are concatenated, and wherever the lines split, the vectors are being copied. The pink circles inside the middle module are pointwise operations, such as vector addition and vector multiplication. The yellow boxes, on the other hand, are learned neural network layers (Olah, 2015).

2.6.4.1 | The Cell State

The cell state acts as a transport highway that flows relative information straight down the entire chain with only a few linear interactions. This is depicted in Figure 2.11 as a horizontal line running through the top. One may think of this as the "memory" of the network, as it allows information from earlier time steps to make its way to later time steps.

2.6.4.2 | Gates

The sigmoid functions (σ) in the neural net layer return values between zero and one, making their neural net layers act as gates that allow or deny information passing into

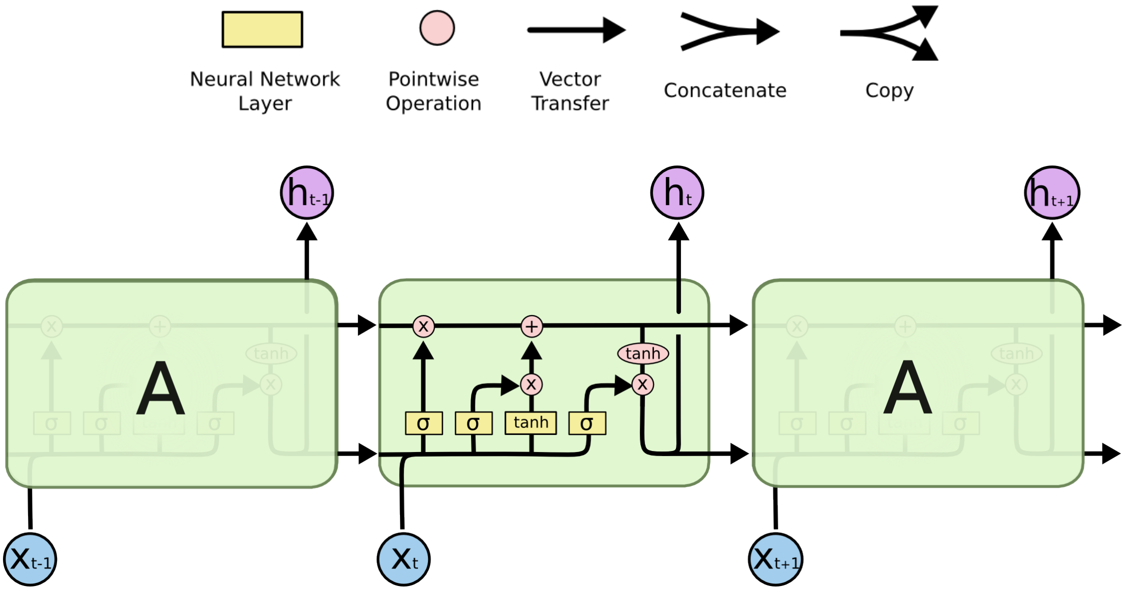


Figure 2.10: An LSTM repeating module, consisting of four layers (Olah, 2015).

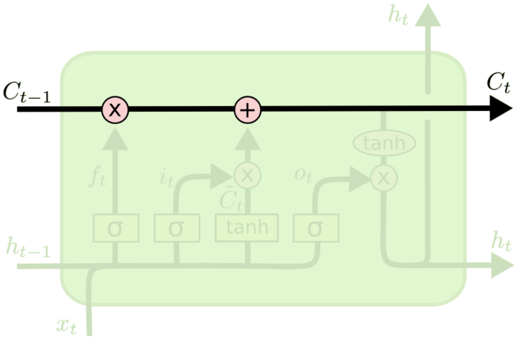


Figure 2.11: LSTM Cell State (Olah, 2015).

the cell state. Three of such gates are present in an LSTM. These gates are the *Forget gate*, *Input gate*, and *Output gate* (Olah, 2015). An example of a gate is shown in Figure 2.12.

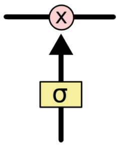


Figure 2.12: LSTM Gate (Olah, 2015).

- **Forget Gate:** Decides what data should be kept or discarded. The result of this gate is called the *forget vector*.

- **Input Gate:** Updates the cell state. This is done by splitting the previous hidden state and the current input to send one copy to a *sigmoid* function, which decides what data is to be updated, and send the second copy to a *tanh* function, which normalises the values $(-1 <= x <= 1)$ to regulate the network. The output of each is multiplied so that the sigmoid function filters the output of the *tanh* function. The *new cell state* is calculated by being multiplied with the forget vector and then being added with the result from the input gate.

- **Output Gate:** Sets up what the next hidden state will be. This is done by passing the previous hidden state and the current input into a *sigmoid* function. The new cell state is also passed through a *tanh* function. The results of the two are multiplied to calculate the new hidden state. The cell state and the new hidden state are then passed on to the next time step.

2.7 | Ensemble Reinforcement Learning

An ensemble strategy combines a set of models that aim to solve the same original task to obtain an optimised model with more accurate and robust estimates or decisions than can be obtained from a singular model (Rokach, 2006). The idea of building such a model is not a new one. Ensemble methods could be traced back to as early as 1977 with *Tukeys Twicing*, which consisted of an ensemble of two linear regression models (Buhlmann and Yu, 2003).

Combined with RL, ensemble models may be utilised to combine function approximators and create a singular policy. Alternatively, different RL algorithms that learn separate value functions may have their derived policies combined in a final policy for the agent (Wiering and van Hasselt, 2008). In their paper, Wiering and van Hasselt (2008) identify a number of ensemble methods that combine multiple RL algorithms in a single agent, including Majority Voting, Rank Voting, Boltzmann Multiplication, and Boltzmann Addition. *Majority Voting* is one of the simplest ensemble models as it combines the best action of each algorithm and bases its final decision on the number of times each algorithm prefers an action. On the other hand, *Rank Voting* lets each algorithm rank the different actions and combines these rankings to select a final action. Finally, in *Boltzmann Multiplication*, Boltzmann exploration (Wiering and van Hasselt,

2008) is used for each algorithm, and the Boltzmann probabilities for each action computed by the algorithms are multiplied. Similarly, *Boltzmann Addition* uses Boltzmann exploration, but the probabilities calculated are added instead of being multiplied.

2.8 | Machine Learning for Portfolio Management

The traditional PS approaches and strategies previously discussed make use of handcrafted features, such as moving averages and other technical indicators (Li and Hoi, 2012), which may perform unsatisfactorily due to poor representation abilities (Deng et al., 2017). Additionally, they assume no transaction cost. This introduces biases into the estimation of accumulative returns (Ormos and Urbán, 2013). In recent years, DANNs have shown strong representation abilities in modelling sequence data (Sutskever et al., 2014). Although extracting sequential price patterns and asset correlations using such techniques is non-trivial (Zhang et al., 2020), many studies have applied them to the finance sector such as (Zhang et al., 2020),(Jiang et al., 2017), and (Hegde et al., 2019).

There are a variety of deep machine-learning approaches to financial market trading, with their goals set to predicting price movements or trends using historical market data as seen in (Heaton et al., 2017) and (Niaki and Hoseinzade, 2013). With historic asset close prices as its input, an ANN can output a vector consisting of a prediction of the next period asset prices. The trading agent is to then act upon this prediction. Therefore, the problem is a supervised learning or regression one. The unstationary nature of the financial market makes market prices challenging to predict, impacting the accuracy of such systems (Jiang et al., 2017). An example of neural network models in related projects can be seen in (Song et al., 2018), where the five different neural network models implemented prove to be successful at extracting meaningful information from past prices. The five neural network models consisted of a backpropagation network, radial basis function network and a general regression network.

In recent work, portfolio optimisation and stock trading have been attempted with various ML strategies. These include Deep Learning (Cao and Cao, 2020), DRL (Hegde et al., 2019; Jiang and Liang, 2018; Li and Peng, 2019), Evolutionary Algorithms (Estalayo et al., 2019) and their variations (Hu and Lin, 2019; Ye et al., 2020). Recent studies such as (Zhang et al., 2020),(Jiang et al., 2017), and (Hegde et al., 2019), utilising DRL in financial applications have moved away from looking at a discrete action space and on to a continuous one. Many of these studies make use of DRL frameworks such as DDPG which provides a model-free ML based approach allowing for continuous states and action spaces (Gran, 2019).

2.8.1 | Deep Reinforcement Learning in Portfolio Optimisation

Sato (2019) examines existing model-free RL approaches applied to the portfolio optimisation problem in literature. Sato (2019) also identifies the *"Bellman's curse of dimensionality "* as a disadvantage present in value-based RL methods such as Q-Learning, when the state and action spaces are large. This disadvantage causes the exploration done by the agent to be inefficient. This issue is not present in policy-based RL methods, and thus they can be applied directly to large continuous domains. However, approximating the optimal policy with an ANN is difficult and could lead to sub-optimal solutions mainly due to its instability, sample inefficiency, and sensitivity on the selection of hyperparameter values (Sato, 2019).

In the work done by (Silver et al., 2014) and (Lillicrap et al., 2016) DPG and DDPG are found to significantly outperform their counterparts in high-dimensional action spaces. These were tested on reinforcement learning benchmark environments, made available by OpenAI gym MuJoCo (Lillicrap et al., 2016). Some examples of these environments include, *Cartpole*, *Cheetah* and *Hopper*. In a similar fashion to these examples, the OLPS problem has a continuous action space. According to Hegde et al. (2019), alternative DRL methods suffer from challenges of stability and do not lend themselves well to such a continuous action space. Jiang and Liang (2018) make use of Deterministic Policy Gradient (DPG) inspired by the work of Silver et al. (2014). In their implementation, Jiang and Liang (2018) avoid the Q-function estimation and instead use a direct reward function due to the fact that training two neural networks may be difficult and sometimes even unstable (Jiang and Liang, 2018). Further recent studies in this field, such as the ones done by Hegde et al. (2019), Zhang et al. (2020), Kanwar (2019), and Gran (2019) have expanded on top of their work, with more risk-averse approaches (Hegde et al., 2019; Zhang et al., 2020), focus on asset correlation (Zhang et al., 2020), genetic algorithms to perform pre-training (Gran, 2019), varieties in neural networks (Kanwar, 2019), and sentiment analysis (Gran, 2019).

Recent related work has been inspired by the research done by Jiang et al. (2017) which provided a detailed paper and open source code [1]. Jiang et al. (2017) presents a deep reinforcement learning solution to the portfolio management problem. Three different model varieties of the framework are evaluated. These models vary by the ANN/Predictor format, which included CNN, RNN and LSTM. These models are evaluated along with various online portfolio selection strategies on cryptocurrency market data with an intra-day trading period of 30 minutes. In all experiments, the profitability surpasses all those from the traditional portfolio-selection models.

[1] https://github.com/ZhengyaoJiang/PGPortfolio

Hegde et al. (2019), inspired by the work of Jiang and Liang (2018) and Lillicrap et al. (2016), make use of full actor-critic DDPG on S&P500 stocks instead of cryptocurrency assets. Hegde et al. (2019) make use of this framework to construct a risk-aware portfolio with higher returns and lower risk compared to a baseline Markowitz approach or to buy and hold strategies (Hegde et al., 2019). This was mainly achieved by introducing a cost-sensitive reward in the form of Sortino Ratio (Equation 2.42).

$$Sortino_{Ratio} = \frac{R_p - r_f}{\alpha_d} \tag{2.42}$$

where R_p is the portfolio return, r_f is the risk free rate, and α_d is the downside standard deviation.

In more recent work, Zhang et al. (2020) introduce a DRL model consisting of a risk-sensitive reward to constrain risk costs. The risk-sensitive reward by defining the empirical variance of log-return on sampled portfolio data ($\sigma^2(\hat{r}_t)$ where $\hat{r}_2$ is the log return on the t-th period) as the risk penalty. The reward function is defined as follows:

$$R = \frac{1}{T}\sum_{t=1}^{T} \hat{r}_t - \lambda\sigma^2(\hat{r}_t) \tag{2.43}$$

where $\lambda \geq 0$ is the trade-off hyperparameter, $\hat{r}_2$ is the log return on the t-th period, and T is the total number of sampled portfolio data. Furthermore, Zhang et al. (2020) include an asset correlation component in their solution by splitting sequential price pattern and asset correlation into two different streams in the architecture. Whilst initially considering using DDPG, Zhang et al. (2020) opt out of using the framework in question due to the state-action values estimated via Q-network. Zhang et al. (2020) claim that this is often hard to learn and might fail to converge. Due to this, Policy Gradient framework is used. Zhang et al. (2020) evaluate their Cost-Sensitive DRL-based PS models, against a variety of OLPS on custom cryptocurrency datasets. The proposed framework is seen beating all of the benchmarks used in the study in terms of

- *Accumulated Portfolio Value* (APV) (Equation 2.44)

  $$APV = \frac{p_t}{p_0} \tag{2.44}$$

  where p_t is the final portfolio value and p_0 is the initial investment.

- *Sharpe Ratio* (Equation 2.45)

  $$Sharpe_{Ratio} = \frac{R_p - r_f}{\alpha} \tag{2.45}$$

  where R_p is the portfolio return, r_f is the risk free rate, and α is the standard deviation.

- *Calmar Ratio* (Equation 2.46)

$$Calmar_{Ratio} = \frac{S_n}{MDD} \tag{2.46}$$

where S_n is the accumulated profit, and MDD is the biggest loss from a peak to a through, known as *Maximum Drawdown*

$$MDD = \frac{ThroughValue - PeakValue}{PeakValue} \tag{2.47}$$

To enhance the learning process of their proposed model, Gran (2019) include a pre-training element using a genetic algorithm which periodically performs selection and mutation during the training process. Initially, the individuals' respective positions are random. Then, for each quarter in the DDPG model training data set, the genetic base decides what portfolio to hold and perform reproduction with genetic mutation. The fitness function utilised in the genetic algorithm in question does not comply with the original objective function but instead aims to help the model obtain a good balance between exploration and exploitation (Gran, 2019). Their proposed solution also includes sentiment analysis via Google search data acquired via Google Trends (Gran, 2019). The solution is evaluated against a Buy and Hold strategy on various datasets acquired from real stock trading data.

2.8.1.1 | Drawbacks

One of the general issues when implementing RL algorithms is to handle the explore-exploit dilemma (Sutton and Barto, 1999). Policy Gradient (PG) approaches are known to have overfitting and converging properties. To overcome overfitting and convergence to local optimum policies, in (Hegde et al., 2019) the agents' weights were randomly initialised at the beginning of each training iteration/epoch. In some studies such as the one by Kanwar (2019), where PG and DDPG frameworks are compared using D-FFNN and CNN, the two frameworks were found to be unstable and sensitive to hyperparameters. For example, the learning rate chosen made a great difference in the result and determined the optimisation. This could also be related to the 'vanishing gradient problem' encountered when training the two ANNs described with gradient-based methods (Kanwar, 2019). Patel (2018) extends DDPG with Double Deep Q-Networks in the critic function to reduce the overestimation of action-values, and in the future work section of his paper, explains how his work may be expanded in various ways, including using *Dueling Networks* to obtain more accurate critic estimates. This is reminiscent of one of the enhancements introduced in TD3 (Fujimoto et al., 2018).

2.8.2 | Ensemble Deep Reinforcement Learning in Portfolio Optimisation

The study done by Yang et al. (2020) employs an ensemble strategy for automated stock trading with three different trading agents. Each of these trading agents consists of a distinct actor-critic based algorithm. These are Proximal Policy Optimisation (PPO), Advantage Actor-Critic (A2C), and DDPG. The ensemble strategy aims to inherit and integrate the best features of the three algorithms, thereby robustly adjusting to different market situations (Yang et al., 2020). The proposed ensemble strategy by Yang et al. (2020) first trains the three agents, then validates all agents via Sharpe Ratio (Equation 2.45) to find the best model, and finally, the best agent is used for trading.

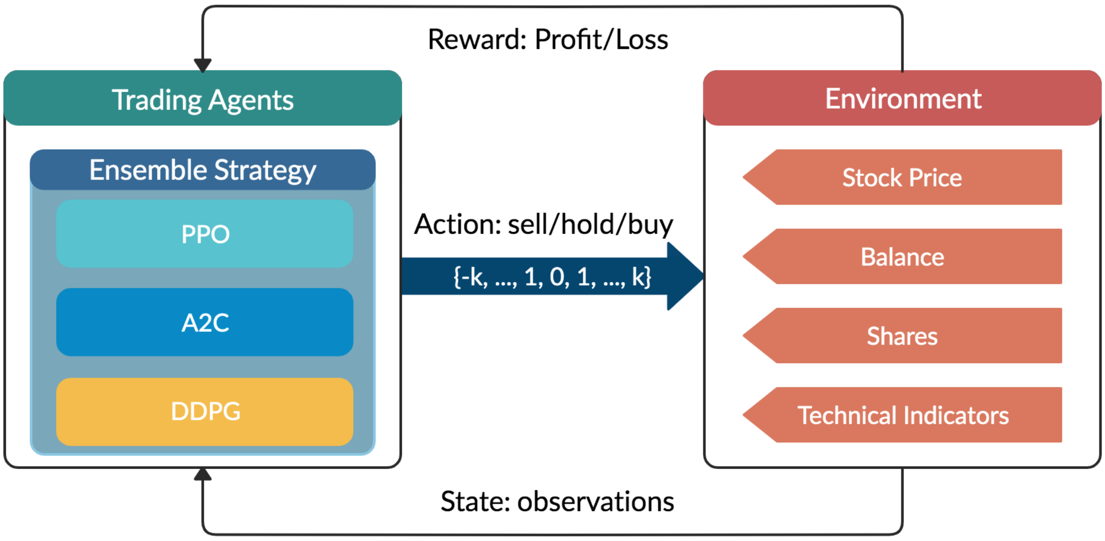


Figure 2.13: Overview of reinforcement learning-based stock trading strategy by Yang et al. (2020).

2.9 | Conclusion

The literature reviewed in the initial phase of our study would suggest that applying the DDPG framework on our problem domain could be an ideal starting point (Jiang et al., 2017). Many recent studies, such as the one done by Gran (2019) have applied such an approach whilst including new innovative elements. All the studies discussed apply this framework in DRL models on custom stock trading environments using real stock trading data. They do so using a variety of DANN architectures, with LSTM showing the most promise. Findings in research done by Fujimoto et al. (2018) show that the DDPG framework has a successor, TD3, which has been scarcely applied on the PS domain. The reduction of overestimation bias provided by the framework could allow for more optimised policies that grant greater long term return (Fujimoto et al., 2018). Additionally, the agents found in recent related studies make use of combined features within the state, including *Open*, *Close*, *High*, *Low*, and *Volume* (Hegde et al., 2019). However, more complex features could be introduced to enhance performance, such as stock movement prediction indicators used in OLPS algorithms (Huang et al., 2016; Li and Hoi, 2012). Ensemble strategies on RL have been studied (Wiering and van Hasselt, 2008), and also applied on the PS domain (Yang et al., 2020), albeit very scarcely. The aim behind the ensemble algorithm proposed by Yang et al. (2020) is to integrate a number of DRL frameworks to adjust to different market situations. However, alternative factors could be used to create an ensemble model that reaches this goal, such as differing state formats.

Therefore, we deduce that it is worth investigating TD3, complex features within the RL state format, and an ensemble RL approach consisting of models with differing state formats to solve the portfolio selection problem. In line with related studies such as those done by Patel (2018), Vogiatzis (2019), Li and Peng (2019), Ye et al. (2020), Kanwar (2019), and Jiang and Liang (2018) we evaluate the performance of the DRL models created in our experiments against those of standard OLPS algorithms.

3 Methodology

In this chapter, we present our implementations of DDPG and TD3 DRL frameworks applied to the portfolio selection domain. Our implementations of these DRL frameworks are based on the corresponding seminal papers, including (Lillicrap et al., 2016) and (Fujimoto et al., 2018). The DDPG and TD3 models proposed in these studies are reapplied to interact with a portfolio selection environment, and with an LSTM ANN predictor. The two frameworks are implemented with similar internal functionality and parameters, providing a level playing field to allow various experiments. We base the implementation of our DRL frameworks on the original papers (Fujimoto et al., 2018; Lillicrap et al., 2016), rather than more recent implementations where they are applied on the portfolio selection (Hegde et al., 2019; Jiang and Liang, 2018), due to difficulties in DRL framework replication. Additionally, no related studies using the TD3 framework were observed during our research. Islam et al. (2017) note the difficulty in the replication of DRL frameworks, and thus the difficulty of benchmarking against previous studies. This is due to the fact that performance is highly dependent on the choice of hyperparameters and the stochasticity of the environment (Islam et al., 2017). Unfortunately, recent studies such as (Zhang et al., 2020) and (Hu and Lin, 2019) do not make the hyperparameters used in their study publicly available. Due to this, multiple papers, such as (Lillicrap et al., 2016) and (Fujimoto et al., 2018) opt-out of replicating the work from previous papers and instead utilise standard OpenAI gym environments [1] allowing for comparison of results. Throughout our research, no standard stock trading or portfolio selection environment has been identified. Due to this, we implement a custom portfolio selection environment, in line with other work Hegde et al. (2019); Jiang et al. (2017). With a custom portfolio selection environment, the results obtained cannot be directly compared against recent related studies. Although this made it difficult to

[1] https://gym.openai.com/

compare results with studies such as (Zhang et al., 2020) and (Hu and Lin, 2019), we aimed to use standard models described in Section 2.3 as benchmarks to validate our results. All the standard models are tabulated along with their references in Table 3.3. In this chapter, we discuss and describe all the elements leading to the implementation of the custom environment to allow for its reproduction.

As discussed in Section 2.3.2, various observed studies related to our problem domain, such as (Patel, 2018), (Vogiatzis, 2019), (Li and Peng, 2019), (Ye et al., 2020), (Kanwar, 2019), and (Jiang and Liang, 2018) make use of standard state-of-the-art OLPS algorithms as benchmarks for their solutions. OLPS algorithms are less sensitive to parameters making them good options as benchmarks which could be applied to custom portfolio management environments.

To address the aim of our work, we identify the following research objectives with their corresponding experiments:

1. Investigate whether recent advances in DRL can lead to improved investment performance when compared to OLPS algorithms.
   Experiment 1: Evaluation of DDPG and TD3 based models.

2. Investigate how OLPS features can be utilised to improve DRL market state representation, leading to increased investment performance.
   Experiment 2: Evaluation of enhancements in State format.

3. Explore possible investment performance improvements through the use of DRL ensemble strategies.
   Experiment 3: Evaluation of Ensemble strategy.

In this chapter, the data used in the experiments, the experiment environment, the DRL frameworks and the experiments are discussed in detail.

3.1 | Stock Trading Data used in Experiments

OLPS algorithms are typically evaluated on a variety of standard stock trading data, such as *NYSE(O), NYSE(N), SP500 and DJIA* (Cover, 1996; Fagiuoli et al., 2007; Li et al., 2012). These consist of actual daily closing prices of a variety of stocks in a specific period. These datasets are publicly available (Li and Hoi, 2014) [2].

[2] http://www.mysmu.edu.sg/faculty/chhoi/olps/datasets.html

In this work, we make use of the newest standard NYSE-based dataset *NYSE(N)*, as it consists of enough steps to allow for training and to test our models. Alternatively, for the SP500 market, a custom dataset is created similarly to other related work such as (Nazir, 2019), with more recent data gathered from yahoo finance [3]. This allows us to see how our models and the benchmarks perform in the current market and curb potential dataset selection and data-snooping biases introduced in OLPS algorithms (Nazir, 2019). The number of assets or stocks within the *NYSE(N)* dataset is 23, whilst the *SP500* consists of 25. The assets chosen for the SP500 dataset were selected according to market capitalisation and liquidity. All the datasets used in this work are described in Table 3.1. Whilst the OLPS algorithms observed do not require an initial training phase, the DRL models implemented require training to learn policies. Therefore, datasets are split in a 6:1 ratio for training and testing, respectively. Throughout the observed literature, no ratio is identified as a standard or the best. Our selected ratio allows for a great number of training steps, whilst still leaving an adequate number of testing steps. The number of steps kept for testing on both datasets is greater than those found in other work such as (Hegde et al., 2019) and (Vogiatzis, 2019), but less than others such as (Zhang et al., 2020) and (Kanwar, 2019). The portfolio values of the OLPS algorithms discussed in Section 2.1 on both datasets are visualised in Figures 3.1, and 3.2.

Datasets	#Asset	Training Data		Testing Data	
		Data Range	Steps	Data Range	Steps
NYSE(N)	23	07/01/1985 to 02/11/2006	5507	03/11/2006 to 29/06/2010	917
S&P500	25	06/01/1995 to 25/01/2017	5552	26/01/2017 to 29/09/2020	925

Table 3.1: Dataset Statistics.

Due to the standard datasets discussed consisting of solely close prices, we format the input dataset to include the previous close price beside the current one. This is done to facilitate the generation of the state features and the calculation of returns.

3.1.1 | Dataset Stock Composition

- As defined in the benchmarks dataset web-page [4], the NYSE(N) dataset consists of: ’ahp’, ’alcoa’, ’amer_brands’, ’coke’, ’comm_metals’, ’dow_chem’, ’Dupont’, ’ford’, ’ge’, ’gm’, ’hp’, ’ibm’, ’ingersoll’, ’jnj’, ’kimb-clark’, ’kin_ark’, ’Kodak’, ’merck’, ’mmm’, ’morris’, ’p_and_g’, ’schlum’, and ’sher_will’.

[3] https://finance.yahoo.com/

[4] http://www.mysmu.edu.sg/faculty/chhoi/olps/datasets.html

- The SP500 dataset consits of: 'AAPL', 'MSFT', 'JNJ', 'JPM', 'PG', 'UNH', 'HD', 'DIS', 'VZ', 'CMCSA', 'ADBE', 'PFE', 'BAC', 'INTC', 'T', 'WMT', 'MRK', 'KO', 'PEP', 'ABT', 'TMO', 'CSCO', 'CVX', 'NKE', and 'XOM'.

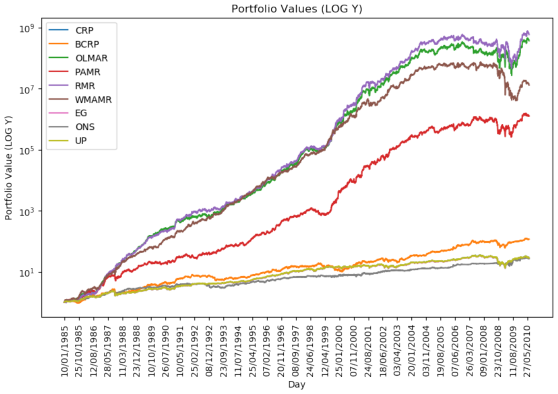


Figure 3.1: NYSE(N) OLPS portfolios.

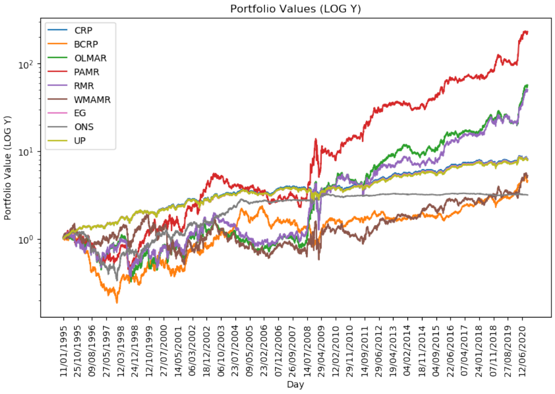


Figure 3.2: SP500 OLPS portfolios.

3.2 | Experiment Environment

A portfolio consists of a number of assets, and throughout our study we make use of solely their close prices. This is due to the absence of open, high and low prices in standard OLPS datasets. Our models are given the opportunity to change portfolio weights after each market open day t. This may be called a step, time step or period according to context. The close prices formed into a *price vector* for day t are denoted as v_t. The *price relative vector* is calculated using the price vector provided at day t (v_t), and the one from the previous day (v_{t-1}):

$$y_t = \left(1, \frac{v_{1,t}}{v_{1,t-1}}, \frac{v_{2,t}}{v_{2,t-1}}, ..., \frac{v_{n,t}}{v_{n,t-1}}\right) \tag{3.1}$$

where $v_{1,t}$ is the closing price of the first asset in the vector for day t, and n is the number of assets. The first element in the price vector $v_{0,t}$ and price relative vector and $y_{0,t}$ are kept constantly as 1 as to provide an option not to trade any asset whilst keeping the trading currency. This is in line with other work such as (Jiang et al., 2017). The price relative vector can be used to calculate change in *total portfolio value* (p) in a period. For example, given that p_{t-1} is the portfolio value at the beginning of period t, without taking transaction cost into consideration, p_t is calculated as follows:

$$p_t = p_{t-1} y_t \bullet w_{t-1} \tag{3.2}$$

where w_{t-1} is the portfolio weight vector at the beginning of period t. Therefore, $w_{t,i}$ is the weight of the ith asset at time t. The initial portfolio weight vector w_0 is set to $(1, 0, ..., 0)$, and the elements in the portfolio weight vector at any period w_t, always sum up to one (Equation 3.3).

$$\sum_i w_{t,i} = 1, \forall t \tag{3.3}$$

Therefore, the *rate of return* for period t is

$$\rho_t := \frac{p_t}{p_{t-1}} - 1 = y_t \bullet w_{t-1} - 1, \tag{3.4}$$

and the corresponding *logarithmic rate of return* is

$$r_t := ln\frac{p_t}{p_{t-1}} = lny_t \bullet w_{t-1}, \tag{3.5}$$

Hence, assuming no transaction cost, the final portfolio value is

$$p_f = p_0 exp\left(\sum_{t=1}^{t_f+1} r_t\right) = p_0 \prod_{t=1}^{t_f+1} y_t \bullet w_{t-1}, \tag{3.6}$$

where p_0 is the initial investment amount. This is set to 1 throughout all our experiments.

3.2.1 | State and Action Representation

Open, Close, High, Low, Volume are some of the features that may be combined in the state (Hegde et al., 2019; Zhang et al., 2020). Our models initially make use of only close prices, and at any period, a state is generated using the dataset with a defined *window length*. The window length is a modular parameter denoting the number of past time steps considered relevant at each period and thus affects the size of each generated state. Our preliminary experiments aim to train and evaluate our models utilising various window lengths. These include 3, 7, 11 and 14, in line with Hegde et al. (2019).

Each value in the state corresponds to the normalised *log return*, $R_{t,i}$ of an asset, i, on a specific day, t. $R_{t,i}$ is defined as:

$$R_{t,i} = \log\left(\frac{Close_{t,i}}{Close_{t-1,i}}\right) \tag{3.7}$$

where $Close_{t,i}$ is the asset's close price for day t, and $Close_{t-1,i}$ is its close price of the previous day, $t-1$. The log return data within the state is normalised using z-score normalisation $\frac{value-\mu}{\sigma}$ across all assets, where *value* is the log return of an asset on day

t. μ and σ are the mean and standard deviation of the log return of all assets on day t, respectively.

In most related studies, such as (Hegde et al., 2019), (Zhang et al., 2020), (Kanwar, 2019), (Patel, 2018), (Gran, 2019), and (Vogiatzis, 2019) an action takes the form of a new weight vector prediction for allocation of capital. We also formulate the actions within our solution is this manner.

$$Action = \alpha_t = w_t = (w_1, ..., w_n) \tag{3.8}$$

3.2.2 | Transaction Costs

Transaction costs are expenses incurred when buying or selling a good or service. These are applied whenever the weights allocated to the assets change. The portfolio vector at the beginning of period t is w_{t-1}. Due to price movements, at the end of the period the portfolio weight vector evolves into

$$w'_t = \frac{y_t \odot w_{t-1}}{y_t \bullet w_{t-1}}, \tag{3.9}$$

where $\odot$ is the element-wise multiplication. Before the next period starts, w'_t is to be transformed into w_t by reallocating the portfolio weights. The *transaction remainder factor* is the factor by which the portfolio value shrinks due to this reallocation procedure $\mu_t.\mu_t \in (0, 1]$. In our work we borrow the equation used to calculate μ_t from (Jiang and Liang, 2018)

$$\mu_t = c \sum_{i=1}^{m} |w'_{t,i} - w_{t,i}|, \tag{3.10}$$

where c is the transaction cost rate. Throughout all our experiments, c is set as 0.25%, which is a commonly used transaction cost rate found in work done by Jiang and Liang (2018), Kanwar (2019), Vogiatzis (2019), and Hegde et al. (2019).

3.2.3 | Exploration and the Reward Function

The aim of the agent is to maximise the final portfolio value p_f at the end of the $t_f + 1$ period. Due to the agent not having control over the choice of the initial portfolio weights p_0, and the number of total time steps t_f, this job is equivalent to maximising the average logarithmic accumulated return R (Jiang and Liang, 2018):

$$R(s_1, a_1, ..., s_{t_f}, a_{t_f}, s_{t_f+1}) := \frac{1}{t_f} ln \frac{p_f}{p_0} = \frac{1}{t_f} \sum_{t=1}^{t_f+1} ln(\mu_t y_t \cdot w_{t-1}) = \frac{1}{t_f} \sum_{t=1}^{t_f+1} r_t \tag{3.11}$$

In Equation (3.11), weight w_{t-1} is given by action a_{t-1} (Equation 3.8), y_t is the *price relative vector* on period t (Equation 3.1), s_t is the state variable, μ_t is the *transaction remainder factor* (Equation 3.10), and r_t is the immediate reward.

3.3 | Deep Reinforcement Learning Frameworks

In this work we build our implementations of the DDPG (Lillicrap et al., 2016; Silver et al., 2014) and TD3 (Fujimoto et al., 2018) DRL frameworks, following their relevant literature closely, along with use of published code [5]. Due to the latter being based on the former, the frameworks have many commonalities which include *Experience Replay*, *Target Networks* and *Exploration vs. Exploitation*. Due to this, the enhancements found in TD3 presented by Fujimoto et al. (2018) were able to be applied on top of a primarily implemented DDPG framework.

3.3.1 | Deep Neural Network Topology

Extracting patterns from a portfolio series is non-trivial due to the non-stationary property of asset prices. Due to this, we propose a Predictor with a Long short-term memory (LSTM) framework in line with related work done by Hegde et al. (2019), Zhang et al. (2020), and Patel (2018). Both the actor and critic networks, for both our DDPG and TD3 frameworks, use the same configuration.

3.3.1.1 | Actor Network

For the actor network, the input layer is defined by the dimensions of the observation of State s, and therefore the size of the window and the number of assets in our portfolio. Additional features may be added within the state for each asset increasing the window size dimension. This is reshaped for input into an LSTM layer, which is followed by a hidden layer. The actors' output layer is a *Softmax* layer with dimensions corresponding to the number of assets, so as to hold the portfolio weight vector, which is used as an action by the agent. The actual output is then bound to have all the values within the action array sum up to 1.

3.3.1.2 | Critic Network

The critic network consists of identical construction, except for the input and output layers. The critic network's input layer consists of the input taken from the observation and

[5] `https://github.com/sfujim/TD3`

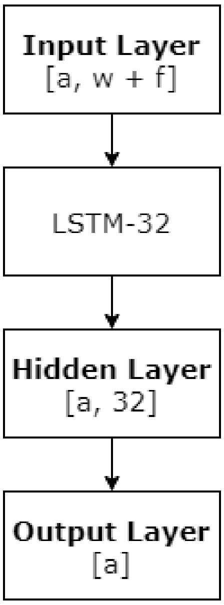


Figure 3.3: Actor network architecture.

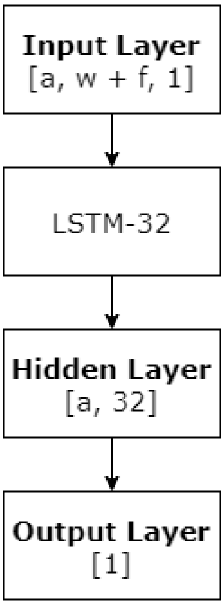


Figure 3.4: Critic network architecture.

the action from the actor network. The action is shown in Figure 3.4 as a 1, as the actions throughout our study consist of a single step. On the other hand, the output layer for the critic network predicts a single output which would be the predicted reward. Due to the *Clipped Double Q-Learning* feature in TD3, two critics are created, each following this exact structure. Figures 3.3 and 3.4 visualise the two LSTM networks, where a is the number of assets, w is the window length, and f is the number of technical indicator values or features.

3.3.1.3 | Actor and Critic learning rates

Even though the actor and critic learning rates are modular in our implementation as parameters, these are kept constant throughout our experiments. These are set to a learning rate of 10^{-4} and 10^{-3} for the actor and critic respectively, based on (Lillicrap et al., 2016). Additionally, from the critic side, the *Discount factor* (γ) is set to 0.99, and the *Target network update ratio* (τ) is set to 0.001. This is also in line with (Lillicrap et al., 2016). The full list of parameters is available in Table 3.2.

3.3.2 | Noise Generation Functions

One distinct difference in implementation between the two frameworks is the noise added to the actions to aid exploration. Whilst DDPG makes use of Ornstein-Uhlenbeck (Lillicrap et al., 2016; Uhlenbeck and Ornstein, 1930), TD3 opts for uncorrelated noise for exploration, as Fujimoto et al. (2018) claim that the former method offered no performance benefits. The Ornstein-Uhlenbeck method generates noise correlated with the previously generated noise to prevent the noise from cancelling out or 'freezing' the overall dynamics (Uhlenbeck and Ornstein, 1930). Some studies, such as (Kanwar, 2019) do divert away from using the Ornstein-Uhlenbeck on the DDPG framework, but the results obtained do not clearly show that this modification has positive or negative effects.

Our implementation of the DDPG framework encompasses the Ornstein-Uhlenbeck method to adhere to the original literature (Lillicrap et al., 2016). Even though the two frameworks use two different noise generation methods, the parameters chosen for each were chosen so as not to cause any intentional advantage or disadvantage to either framework. The parameters chosen for each method is also based on the implementations in the original literature (Fujimoto et al., 2018; Lillicrap et al., 2016). These parameters can be seen in Table 3.2.

3.3.3 | Training Process

The datasets are split in a 6:1 ratio for training and testing, respectively. Training is done over 400 episodes or epochs, each consisting of 1000 steps. At the start of each episode, the agent is placed at a random point within the training subset. Any starting point chosen must allow for the training steps to be completed. ANN training is done with a mini-batch of 64, sampled uniformly from a replay buffer consisting of the agents' history. This is a common procedure in both the original DDPG and TD3 implementation, as it uses hardware and resources more efficiently at the computer-architecture level (Patterson and Gibson, 2017). The mini-batch size parameter used throughout our experiments is in line with (Lillicrap et al., 2016).

3.3.4 | Value Function Threshold

The greater our models are trained and optimised, the greater their chance of overfitting to the training data. If so, the learned policy would perform poorly in the testing phase due to possible noise or random fluctuations being picked up during training. To address this issue, we include a threshold reward value parameter that would allow the training phase to end before the defined number of episodes. Therefore training ends when 10 episode rewards exceed the threshold amount in succession, or the total number of episodes defined (400) is reached. This threshold value is coined as the *value function threshold*. Whilst the number of successive and excessive rewards is set to 10 in our implementation, the value function threshold is implemented as an adjustable parameter to allow for optimisation (Table 3.2).

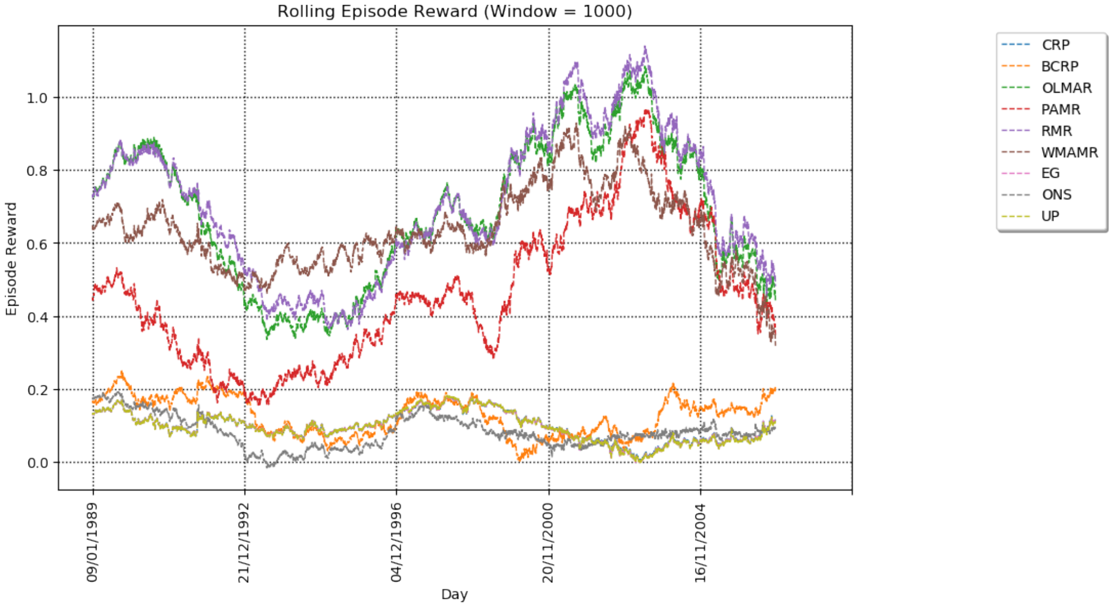


Figure 3.5: Rolling Episode Reward by OLPS Algorithms.

To help us choose the threshold reward value in our experiments, we can first examine the performance done by the benchmark OLPS algorithms. Figure 3.5 visualises the rolling episode reward value achieved by the OLPS algorithms on the NYSE(N) in-sample/training dataset with a window of 1000 steps. The RMR algorithm achieves the greatest episode reward value of approximately 1.1 average logarithmic accumulated return. Naturally, the threshold reward value for our models must exceed this value if we aim for our models to perform better overall. Additionally, if the threshold is set too low, the training of the models may end too quickly without converging to an optimal policy. In our experiments on the NYSE(N) dataset, the threshold value is set to 3 average logarithmic accumulated return to strike a balance. It was observed that models that trained excessively above this threshold had suffered decreased performance during the testing phase. The *value function threshold* during experiments on the SP500 dataset was set to 2, as the OLPS algorithms performed poorer than on the NYSE(N) dataset. These parameters are listed in Table 3.2.

3.4 | Parameters

The following list includes brief discussions on individual hyperparameters tested and used within our experiments. These are summarised in Table 3.2.

- **Episodes and Max Step**: A variety of *episode numbers* were considered and tested in initial runs with the DDPG DRL algorithm. These included 100, 200, 400, 600, and 800 episodes. Training with 400 was found to allow stable convergence without overfitting. (Section 3.3.3)

- **Window length**: The *window lengths* used in our work are the same ones used by Hegde et al. (2019). Larger window lengths were also considered, such as 16, but were found to have adverse performance effects. (Section 3.2.1)

- **Value function threshold**: The *value function threshold* used for the two datasets were based on the performance of the baseline algorithms on the corresponding training subsets. (Section 3.3.4).

- **Max step**: A variety of values could be used in this configurable parameter but is best balanced with the number of episodes to allow for convergence without overfitting.

- **Buffer and Batch size**: ANN training is done with a mini-batch of 64, sampled uniformly from a replay buffer consisting of the agents' history. The mini-batch parameters used throughout our experiments are in line with (Lillicrap et al., 2016). (Section 3.3.3)

- **τ (tau) and γ (gamma)**: The *target network update ratio* and *reward discounting factor* are both in line with the original DDPG implementation by Lillicrap et al. (2016). (Section 3.3.1.3)

- **Actor and Critic learning rate (α, β)** A variety of *actor* and *critic* learning rate combinations were considered in initial testing, including: actor: $1x10^{-4}$ and critic: $1x10^{-4}$, actor: $1x10^{-4}$ and critic: $1x10^{-3}$, and actor: $1x10^{-3}$ and critic: $1x10^{-3}$. The combination selected was found to provide optimal results. This is also the configuration chosen by Lillicrap et al. (2016) in the seminal paper proposing the DDPG framework. The configuration selected is used throughout all experiments. (Section 3.3.1.3)

- **DDPG Parameters**: The parameters selected are taken directly from the seminal paper to provide optimal performance (Lillicrap et al., 2016). Better parameters for our environment and individual datasets could be identified with several trials.
- **TD3 Parameters**: The parameters selected are taken directly from the seminal paper to provide optimal performance (Fujimoto et al., 2018). Better parameters for our environment and individual datasets could be identified with several trials.
- **Ensemble Parameters**: To encapsulate the window lengths used by the internal DRL models, a larger *ensemble window length* was chosen. Our experiments use a window length of 21, which is approximately a month in market open days. Alternate, possibly larger window lengths could be used. The *action length* parameter is set as 1 to fit our problem setting, which consists of daily trading.

Hyperparameter	**Value**	**Description**
Window length	3, 7, 11, and 14	Window or observation size
Value function threshold	3 for NYSE(N), 2 for SP500	Reward threshold used to stop training
Max Step	1000	Number of steps completed in episode
Buffer size	10^5	Size of replay buffer
Batch size	64	Mini-batch size during training
τ (tau)	0.001	Target network update ratio
γ (gamma)	0.99	Reward discounting factor
Actor learning rate (α)	$1x10^{-4}$	Actor learning rate
Critic learning rate (β)	$1x10^{-3}$	Critic learning rate
Seed	1338	Random seed number
(DDPG parameters)		
σ (sigma)	0.2	Ornstein-Uhlenbeck parameter
θ (theta)	0.15	Ornstein-Uhlenbeck parameter
dt	0.002	Ornstein-Uhlenbeck parameter
(TD3 parameters)		
Policy noise	0.2	Exploration noise
Noise clip	0.5	Maximum value of the Gaussian noise
Policy frequency	2	Number of iterations to wait before the policy network updates
(Ensemble parameters)		
Window length	21	Window or observation size
Action length	1	Steps taken with action

Table 3.2: Hyperparameter summary.

3.5 | Experiment 1: Evaluation of DDPG and TD3 based models

The first experiment aims to address the first objective, by investigating how DRL-based portfolio optimisation models using the DDPG and TD3 frameworks perform compared to each other and OLPS algorithm benchmarks. In this experiment we test the hypothesis of whether:

- The portfolio optimisation models using the TD3 DRL framework provide improved results over the ones using DDPG.

Additionally throughout all experiments we test the hypothesis of whether:

- The results obtained by our DRL portfolio optimisation models exceed those of the benchmarks.

In the first experiment, models are trained and tested on the two datasets, using the defined window lengths. The training performance of each model is observed. In line with a number of recent related papers such as (Hegde et al., 2019), (Gran, 2019), (Zhang et al., 2020), and (Vogiatzis, 2019), we make use of a series of criteria to evaluate the performance of our DRL models. These are:

- *Average Daily Yield*, which is simply the mean of all the returns obtained.

- *Sharpe Ratio*, which is a measure created by the Nobel Prize winner Sharpe (1994), to help investors compare the return of an investment with its risk. (Equation 2.45).

- *Sortino Ratio*, which is a risk-aware measure that is very similar to *Sharpe ratio*. The sole difference is that it penalises only downside volatility (Equation 2.42) (Sortino and Price, 1994).

- *Maximum Drawdown* (MDD), which is the maximum loss from a peak to a trough of a portfolio, before a new peak is attained (Equation 2.47).

- *Final Portfolio Value*, which consists of the wealth an agent has after the last step of the test dataset has been completed.

We evaluate the models against each other and against the baselines identified. The model with the best performing window in-sample for each framework is selected for comparison. The best performing framework is then passed on to the next experiment.

3.6 | Experiment 2: Evaluation of Enhancements in State Format

The second experiment aims to address the second objective, which is to investigate the effect of including functionality from OLPS algorithms in the state representations of our DRL-base portfolio optimisation models. In the second experiment, we test the hypothesis of whether:

- The enhancement within the state format positively affects the performance of the DRL-based portfolio optimisation models.

The models are trained, tested and evaluated on both the two datasets using the same evaluation criteria used in the previous experiment. The models with differing state formats are evaluated against each other, along with the benchmarks and OLPS algorithms identified. The model with the best performing in-sample window for each state format is selected for comparison. The best performing state format is then used in the next experiment.

3.6.1 | Motivation

Our models may be improved in a variety of ways. Without any external components, the three essential elements of the RL framework are the State, Action, and Reward. The form of the Action, which is a list of weights assigned to portfolio assets, is essential, and this structure is to remain in any future model. The forms of the State and Reward, on the other hand, can be adjusted in ways that could introduce improvements. For example, Hegde et al. (2019) introduce a risk-aware reward function using Sortino Ratio to improve their model. We would be inclined to try such a reward function, but altering the reward function also implies a shift in the optimisation target. Such a reward function is viable for experiments such as the one by Hegde et al. (2019) as they make use of a *Mean Variance Theory* benchmark, which aim to reduce risk whilst maximising cumulative wealth. On the other hand, PS strategies belonging in the *Capital Growth Theory* school, such as the OLPS benchmarks used, are less risk-averse (Li et al., 2012). The state format, consisting solely of log returns within a window limit, may be considered primitive and might not consist of enough information to make accurate decisions.

3.6.2 | Learning from OLPS algorithms to enhance state representation

The starting point chosen in our study to improve the state is the manipulation of the stock close prices done by the OLPS benchmarks. Two of the best performing OLPS models that consist of a data processing phase in the previous experiment are OLMAR and RMR (Figure 3.1). This is noted by their performance throughout the whole datasets in terms of rolling episode reward. This can be visually seen in figure 3.5, where we show the rolling episode reward achieved by the OLPS portfolios on the NYSE(N) dataset. These models are classified as 'Follow the Loser' algorithms, making use of Mean Reversion strategies. The OLMAR step process starts by calculating the next price relative for the portfolio assets (Li and Hoi, 2014). A simple moving average function, $MA_t = \frac{1}{w}\sum_{i=t-w-+1}^{t} p_i$ is adopted to make this calculation. The corresponding price relative is defined in Equation 2.13. Then, Passive Aggressive online learning from PAMR is adopted to learn a portfolio (Li and Hoi, 2014).

RMR builds on top of OLMAR with an updated equation aiming to reduce estimation errors caused by noises and outliers in the data (Huang et al., 2016). This is done via robust L_1-median estimator. RMR explicitly estimates the next price vector via robust L_1-median estimator at the end of t^{th} period, that is, $\hat{p}_{t+1} = L_1 med_{t+1}(w) = \mu_{t+1}$, where w is the window size, and μ is calculated by solving *Fermat-Weber problem* (Weber, 1929) defined in Equation 2.17. L_1-median is the point with minimal sum Euclidean distance to k given price vectors. Therefore, the expected price relative with the estimator is defined in Equation 2.18. Then RMR follows the similar portfolio optimisation method as OLMAR to learn an optimal portfolio (Li and Hoi, 2014).

3.7 | Experiment 3: Evaluation of Ensemble Strategy

The third experiment aims to address the third objective, which is to investigate whether the combination of multiple DRL-models with differing state formats into one ensemble model improves overall performance. In the third experiment, we test the hypothesis of whether:

- The combination of multiple DRL-models with differing state formats into one ensemble model improves portfolio optimisation performance.

The ensemble models are compared with all the previously highlighted models along with the OLPS algorithm baselines on the two test datasets. The evaluation and comparison of models are made in terms of average daily yield, Sharpe ratio, Sortino ratio, and maximum drawdown evaluation criteria.

3.7.1 | Motivation

The behavioural effect of the DRL window length parameter raises a question about the possibility of an aggregated, optimised model and its' potential. The act of utilising all four models in one optimised model is formulated into an ensemble model, aiming to inherit and integrate the best features of our models, which may have non-correlating periods during testing due to differences in the state format (Yang et al., 2020). This lack of correlation suggests that the models behave differently in different market situations. At any point in time, evaluating a window of past performances by the different models may identify the best way forward.

Unlike Yang et al. (2020), we do not train the models concurrently but separately to allow individual evaluation of performance. These agents are picked and combined into an optimised ensemble model. In our ensemble strategy, the actions created for each agent are all passed in the environment. During this process, the corresponding returns are recorded. We implement the ensemble strategy on top of the enhancements previously identified, using the four window lengths. This is created to address the fourth research question and the third objective in our work. We, therefore, create an ensemble of the models created in the previous experiment. We make use of the *Aver-*

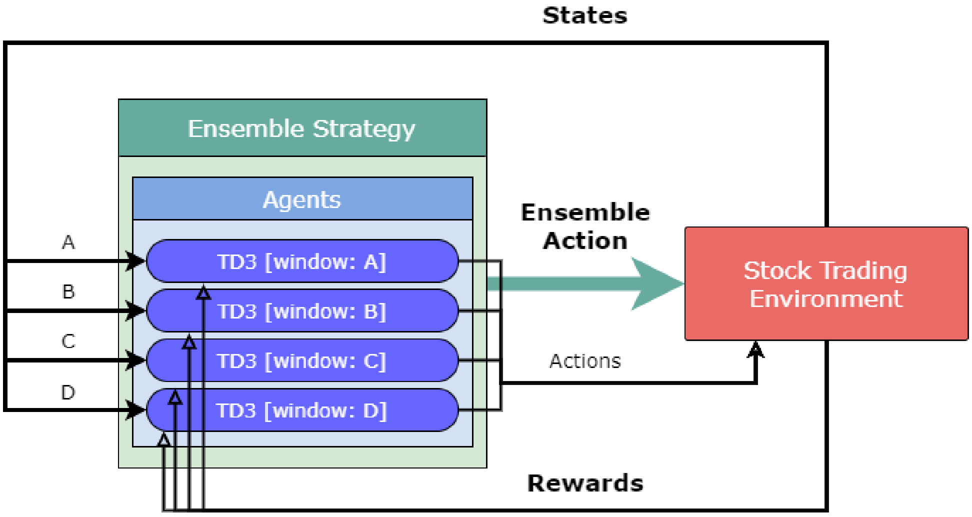


Figure 3.6: Overview of the ensemble deep reinforcement learning-based stock trading strategy proposed.

age Return of an *observation window* (w) as the selection criteria. The similar ensemble model by Yang et al. (2020) in contrast, makes use of Sharpe Ratio (Equation 2.45) as its' selection criteria. In our experiment, a selection criterion based solely on average return is more befitting due to the OLPS algorithms used as benchmarks which belong to the *Capital Growth Theory* school of PS.

3.7.2 | Execution process

The algorithm created uses two parameters: the window length w and action length n. The window length refers to the number of past steps our ensemble algorithm considers when calculating the performance of each of the DRL agents and make its' portfolio choice. On the other hand, the action length refers to the number of steps the portfolio choice taken stands. In our experiment, w is set to 21, approximately a month in trading days. Furthermore, n is set to 1 to have a portfolio choice be made at every step in the out-of-sample dataset. For the first w days, the model that performs the best in-sample is selected. Then, through the remainder of the steps, the average return of each of the DRL models within the previous 21 days is calculated, and the action belonging to the best performing model is selected passed through the environment for 1 step.

3.8 | Benchmarks and OLPS algorithm Parameters

In this work we evaluate our DRL models against a number of benchmarks and OLPS algorithms, which are summarised in Table 3.3. The default parameters or those suggested by their corresponding studies were used for each of the algorithms. The parameters chosen for each are displayed in Table 3.4.

Table 3.3: Benchmarks and OLPS algorithms along with references

Classification	Strategy	References
Benchmark	Constant Rebalanced Portfolios	Kelly (1956)
Benchmark	Best Constant Rebalanced Portfolios	Cover (1996)
Follow-the-Winner	Exponential Gradient	Helmbold et al. (1996, 1998)
Follow the Winner	Universal Portfolios	Cover (1996)
Follow-the-Loser	Passive Aggressive Mean Reversion	Li et al. (2012)
Follow-the-Loser	Online Moving Average Reversion	Li and Hoi (2012)
Follow-the-Loser	Weighted Moving Average Mean Reversion	Gao and Zhang (2013)
Follow-the-Loser	Robust Median Reversion	Huang et al. (2016)
Meta-Learning	Online Newton Step	Agarwal et al. (2006)

Table 3.4: Benchmarks and OLPS algorithm Parameters

Strategy	**Parameter**	**Value**
Constant Rebalanced Portfolios	None	
Best Constant Rebalanced Portfolios	None	
Exponential Gradient	η: Learning Rate	0.05
Universal Portfolios	eval_points: Number of evaluated points	100
	leverage: Maximum leverage used	1
Passive Aggressive Mean Reversion	ϵ: Control Parameter	0.5
Online Moving Average Reversion	w: Window	5
	ϵ: Control Parameter	10
Weighted Moving Average Mean Reversion	w:Window	5
	ϵ: Control Parameter	0.5
Robust Median Reversion	w: Window	5
	ϵ: Control Parameter	10
	τ: Precision for finding median	0.001
Online Newton Step	δ	0.125
	β	1
	η	0

3.9 | Hardware Requirements

Although the minimum hardware requirements have not been identified for this software, we recommend:

- A core i5 processor, or an equivalent Central Processing Unit,
- Ubuntu 20 Operating system. (One may use Virtual Box to emulate this),
- 4 GB of Random-access Memory

By making use of a Graphics Processing Unit, one is able to speed up some of the processing that happens throughout the application. This is not a requirement, and is not possible when using Virtual Box.

3.10 | Software and Libraries

Python was the language chosen to develop our solution, perform our experiments and create visualisations. This is motivated by the multitude of readily available libraries, such as *Tensorflow* and *Keras* that provide a solid a backbone to ML related projects. Both of these libraries were utilised with versions 1.15.0, and 2.3.1 respectively. Additional scientific libraries such as *numpy*, *scipy*, *pandas* are utilised. The visualisations were created with the help of *matplotlib* and *seaborn* libraries.

Our implementation of the stock trading environment and the DDPG models are heavily based, and extend on the work done by Jiang et al. (2017), which is kept publicly available [6], and by further work completed in studies motivated by the work of Jiang and Liang (2018) and Lillicrap et al. (2016) [7]. Fujimoto et al. (2018) also made their state of the art DRL framework (TD3) available [8]. We re-implement this framework with the libraries of our choice onto our problem domain. All the OLPS algorithms used as baselines in our work are available online under an MIT licence [9]. A number of these algorithms were re-implemented to allow their implementation into our experiments.

Table 3.5 below shows the main packages used throughout our study. The anaconda environment used is to be exported and made available to restore.

Table 3.5: Main packages used throughout study.

Package	**Version**	**Description**
numpy	1.18.1	Array processing for numbers, strings, records, and objects
matplotlib	3.1.3	Publication quality figures in python
pandas	0.25.1	High-performance, easy-to-use data structures and data analysis tools
tensorflow	1.15.0	Machine learning library
keras	2.3.1	Deep learning library for tensorflow
tflearn	0.3.2	Modular and transparent deep learning library built on top of Tensorflow
hdf5	1.10.4	Library and file format for storing and managing data
gym	0.17.1	A toolkit for developing and comparing reinforcement learning algorithms
yfinance	0.1.54	Yahoo! Finance market data downloader
fix-yahoo-finance	0.1.30	Fix for Pandas Datareader's 'get_data_yahoo()'

[6] https://github.com/ZhengyaoJiang/PGPortfolio

[7] https://github.com/bassemfg/ddpg-rl-portfolio-management, https://github.com/liangzp/Reinforcement-learning-in-portfolio-management

[8] https://github.com/sfujim/TD3

[9] https://github.com/Marigold/universal-portfolios

4

Results & Evaluation

In this chapter, we present the results and evaluation of the experiments defined in the previous chapter. The objectives defined, along with their corresponding experiments, are the following:

1. Investigate whether recent advances in DRL can lead to improved investment performance when compared to OLPS algorithms.
   Experiment 1: Evaluation of DDPG and TD3 based models.

2. Investigate how OLPS features can be utilised to improve DRL market state representation, leading to increased investment performance.
   Experiment 2: Evaluation of enhancements in State format.

3. Explore possible investment performance improvements through the use of DRL ensemble strategies.
   Experiment 3: Evaluation of Ensemble strategy.

Please refer to Section 4.2 to view the results of Experiment 1, Section 4.3. to view the results of Experiment 2, and Section 4.4 to view the results of Experiment 3. The results are discussed and evaluated in Section 4.5.

4.1 | Use of Benchmarks in Literature

Within our study we make use of a number of standard benchmark algorithms. These are briefly described in Section 2.3, and the full list of benchmark algorithms is available in Table 3.3. The algorithms are made publicly available by Li and Hoi (2012)[1]. To

[1] https://github.com/OLPS/OLPS/tree/master/Strategy

be used in our study, the OLPS algorithms were modified to be able to function with our custom portfolio environment. After modifying them accordingly, each of the algorithms were tested on a number of full standard datasets used in (Li and Hoi, 2012) such as the *NYSE(N)* dataset (Figure 3.1). Only the algorithms with adequately replicated results were included.

A variety of related papers make use of one or more of the benchmarks described and utilised in our study. They make use of these benchmarks in a similar way, but all use their own datasets and environments, with different training to testing ratios and hyperparameters. Patel (2018) makes use of OLMAR and ONS, Li and Peng (2019) make user of CRP, PAMR, OLMAR and WMAMR, Ye et al. (2020) make use of CRP, OLMAR and WMAMR, and Jiang and Liang (2018) make use of CRP, OLMAR, PAMR, WMAMR, RMR, ONS, UP and EG. In all these studies, the writers manage to create solutions that beat all their benchmarks.

4.2 | Experiment 1 Results

In our first experiment, we address our first objective by implementing the portfolio optimisation models using both the DDPG and TD3 frameworks with the window lengths defined in Section 3.2.1. This provided us with four models for each framework, which we train and test one-by-one on the *NYSE(N)* and *SP500* datasets in a 6:1 training to testing ratio. This ratio was found through testing prior to the start of the experiment, to provide an adequate number of steps in the training phase in order to allow the agents to train appropriately. The selected ratio and number of steps is similar to other studies such as the ones done by Vogiatzis (2019) and Kanwar (2019). The model names are formulated using the RL algorithm (e.g. DDPG), the window length (e.g. 3), and the Neural Network format (e.g. lstm). Therefore, the models evaluated consist of:

- DDPG-3-lstm
- DDPG-7-lstm
- DDPG-11-lstm
- DDPG-14-lstm
- TD3-3-lstm
- TD3-7-lstm
- TD3-11-lstm
- TD3-14-lstm

By the end of this experiment we aim to choose the best performing models to be included in the next experiment. We hypothesise that the performance of our DRL models exceeds that of the OLPS benchmarks identified in Section 2.3, and that the portfolio optimisation models using the TD3 DRL framework provide improved results over the ones using DDPG.

4.2.1 | Training Results

With the use of *tensorboard*, we were able to keep track of the performance of each of our models throughout the training phase. This data obtained whilst training on the *NYSE(N)* dataset was compiled and then visualised in Figure 4.1. Whilst performing this test, the *value function threshold* was excluded to evaluate the performance through the full training process consisting of 400 episodes. In the rest of the tests, the value function threshold is set accordingly (Table 3.2). The results obtained showcase the superiority of the TD3 framework, as all the models with the TD3 framework have steeper learning curves from those using the DDPG framework. To aid the visibility of the learning curves for the DDPG models, these are visualised separately in Figure 4.2.

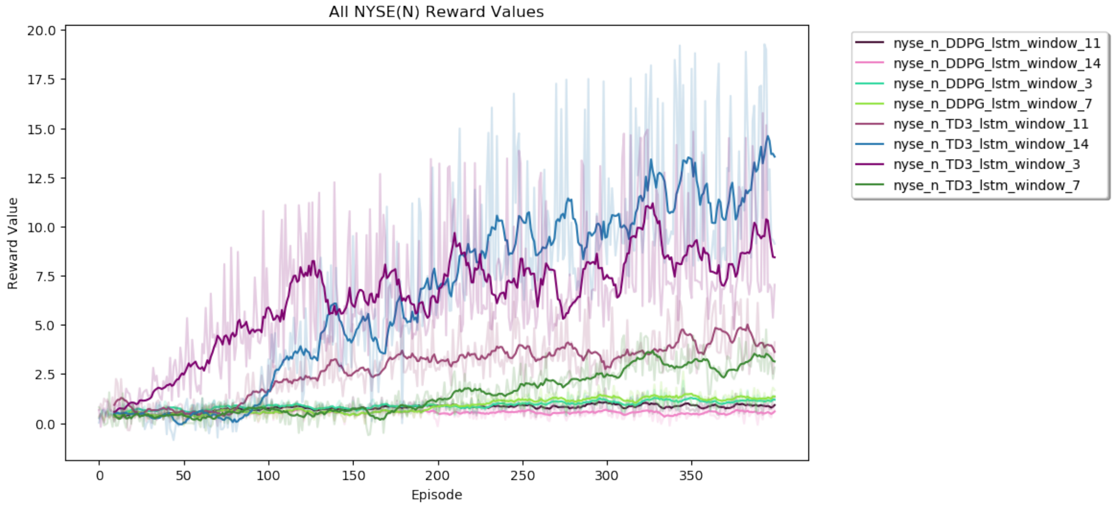


Figure 4.1: Learning curves for each of the framework and window length combination models. The solid lines represent the moving average reward on a 10 epoch window, whereas the shaded lines are the actual reward values obtained for each epoch.

Even though *TD3-3-lstm* is seen converging quicker than all the other models, the *TD3-14-lstm* achieved a greater level of optimisation by the end of the episodes. The fast convergence of *TD3-3-lstm* could be attributed to the smaller state size. Although the *TD3-7-lstm* and *TD-11-lstm* do not reach the same level of reward values, they still outperform all the DDPG based models. By observing the learning curves obtained by the DDPG models in Figure 4.2, we can confirm that *DDPG-7-lstm* achieves the best reward values among the DDPG based models by the end of the training episodes.

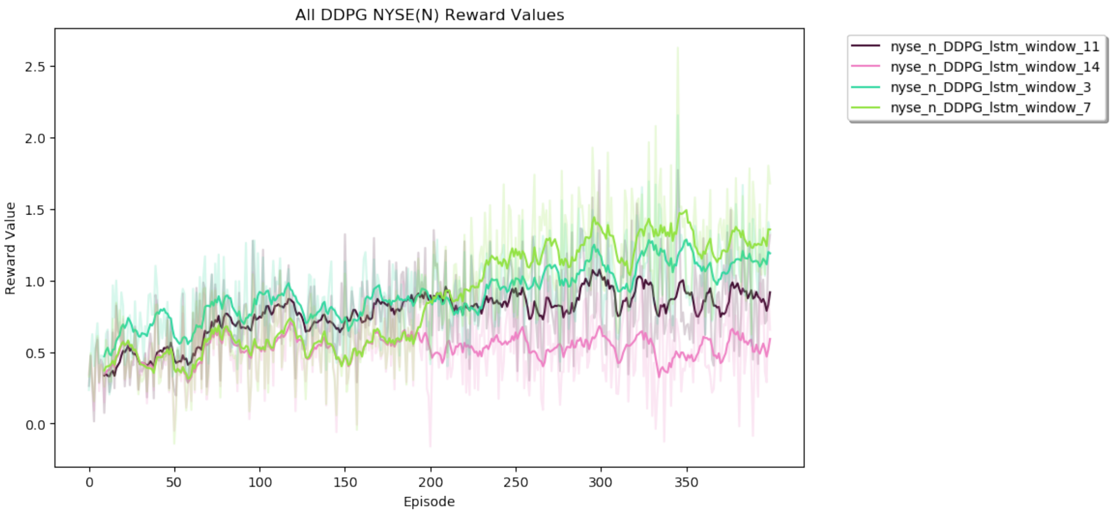


Figure 4.2: Learning curves for the DDPG framework and window length combination models. The solid lines represent the moving average reward on a 10 epoch window, whereas the shaded lines are the actual reward values obtained for each epoch.

4.2.2 | NYSE(N) Testing Results

The trained policy of each model is evaluated on the remaining $\frac{1}{6}$ of the *NYSE(N)* dataset from 03/11/2006 to 29/06/2010. This dataset proved to be a difficult test for our models due to the *"Great Recession"* stock market crash of 2008 [2]. The training subset does consist of another crash in 1987, which could help train our models accordingly. To compare the performance of the two frameworks, we observe the models with the greatest in-sample performance for each corresponding framework. These are *DDPG-7-lstm* and *TD3-14-lstm*. The corresponding portfolio values are visualised in Figure 4.3, along with those obtained by the OLPS algorithms. The results obtained are shown in Table 4.1. To review the robustness of our models, following the approach done by Hegde et al. (2019), we calculate the mean and standard deviation of the evaluation results obtained across all window sizes (Table 4.2).

[2] https://www.thestreet.com/markets/history-of-stock-market-crashes-14702941

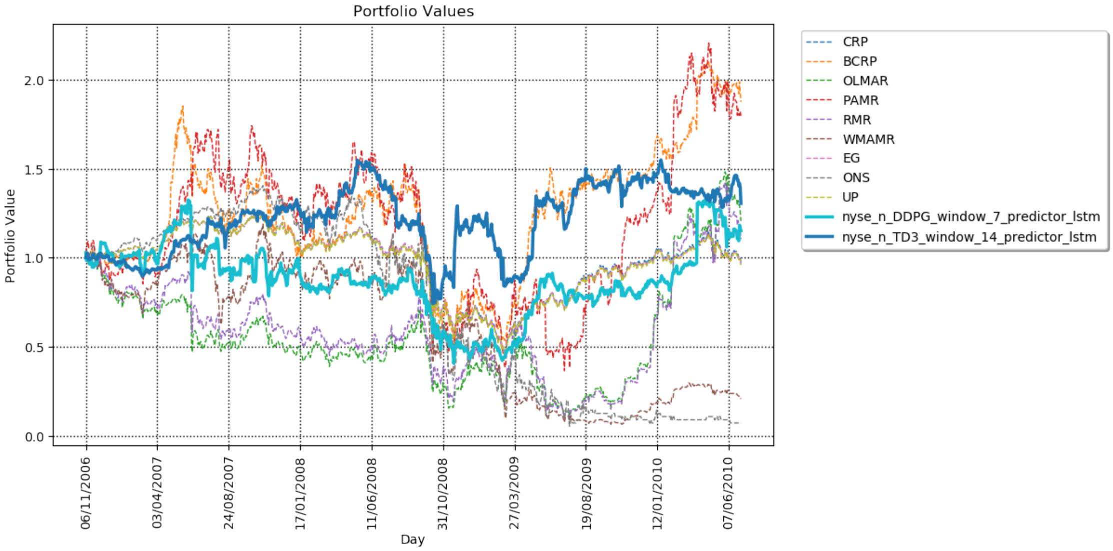


Figure 4.3: Portfolio values for *DDPG-7-lstm* and *TD3-14-lstm* models on the *NYSE(N)* test dataset along with OLPS algorithms.

Model	Average Daily Yield (%)	Sharpe Ratio (%)	Sortino Ratio (%)	Maximum Drawdown (%)	Final Portfolio Value
DDPG-7-lstm	**0.071**	2.062	2.760	69.351	1.151
TD3-14-lstm	0.059	**2.382**	**3.301**	**52.122**	**1.305**
Market Value (UCRP)	0.015	0.768	0.982	63.279	0.97
BCRP	0.114	**3.737**	**5.803**	73.208	**1.876**
OLMAR	**0.22**	3.414	4.678	91.443	1.206
PAMR	0.182	3.666	5.072	79.034	1.797
RMR	0.210	3.26	4.527	91.612	1.104
WMAMR	0.024	0.377	0.478	94.642	0.206
EG	0.013	0.711	0.907	**63.122**	0.962
ONS	0.006	0.076	0.085	96.181	0.072
UP	0.013	0.695	0.89	63.674	0.958

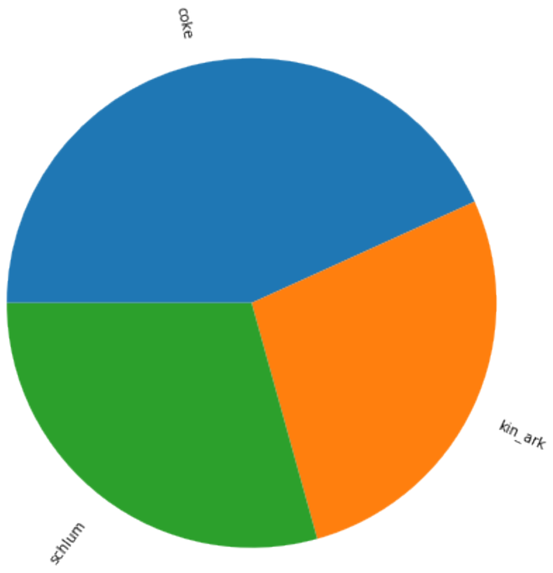
Table 4.1: The evaluation criteria results for *DDPG-7-lstm* and *TD3-14-lstm* models on the *NYSE(N)* test dataset along with OLPS algorithms. The best value for each criteria is bold.

Model	Average Daily Yield (%)	Sharpe Ratio (%)	Sortino Ratio (%)	Maximum Drawdown (%)	Final Portfolio Value
DDPG models	0.039 ± 0.049	1.164 ± 1.255	1.577 ± 1.703	73.992 ± 20.141	0.887 ± 0.338
TD3 models	0.032 ± 0.063	1.401 ± 2.225	2.08 ± 3.218	70.764 ± 18.387	1.027 ± 0.718

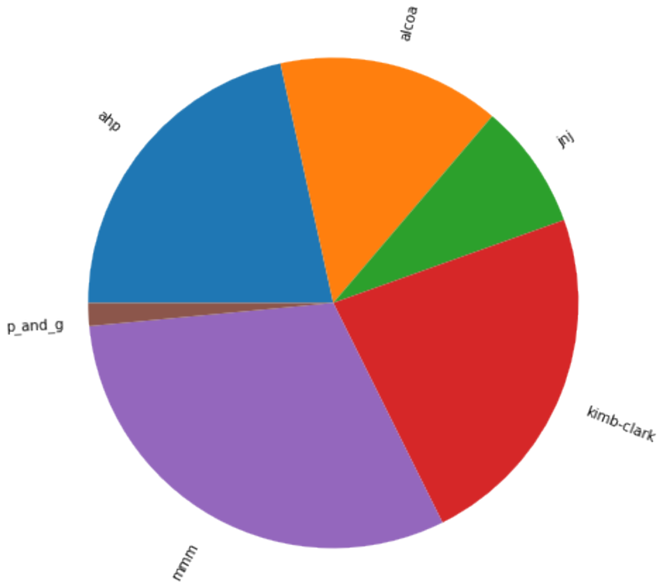
Table 4.2: Result comparison of DDPG and TD3 models on the *NYSE(N)* test dataset over all window sizes.

4.2.2.1 | Additional Visualisations

To investigate the diversification factor of the portfolios generated, pie charts of the average weight allocated to each asset are visualised in Figure 4.4. Furthermore, plots of the main selected assets for each step, superimposed over the portfolio value graph, are displayed in Figure 4.5. Due to the continuous action space, the points within the plot were created by selecting the asset with the greatest weight for each step. These are accompanied by a plot of the normalised values of the corresponding assets. Although this information is not tied to the defined aims and objectives, it allows for more insight into the behaviours of the two models. Additionally, this could also help justify their performance.

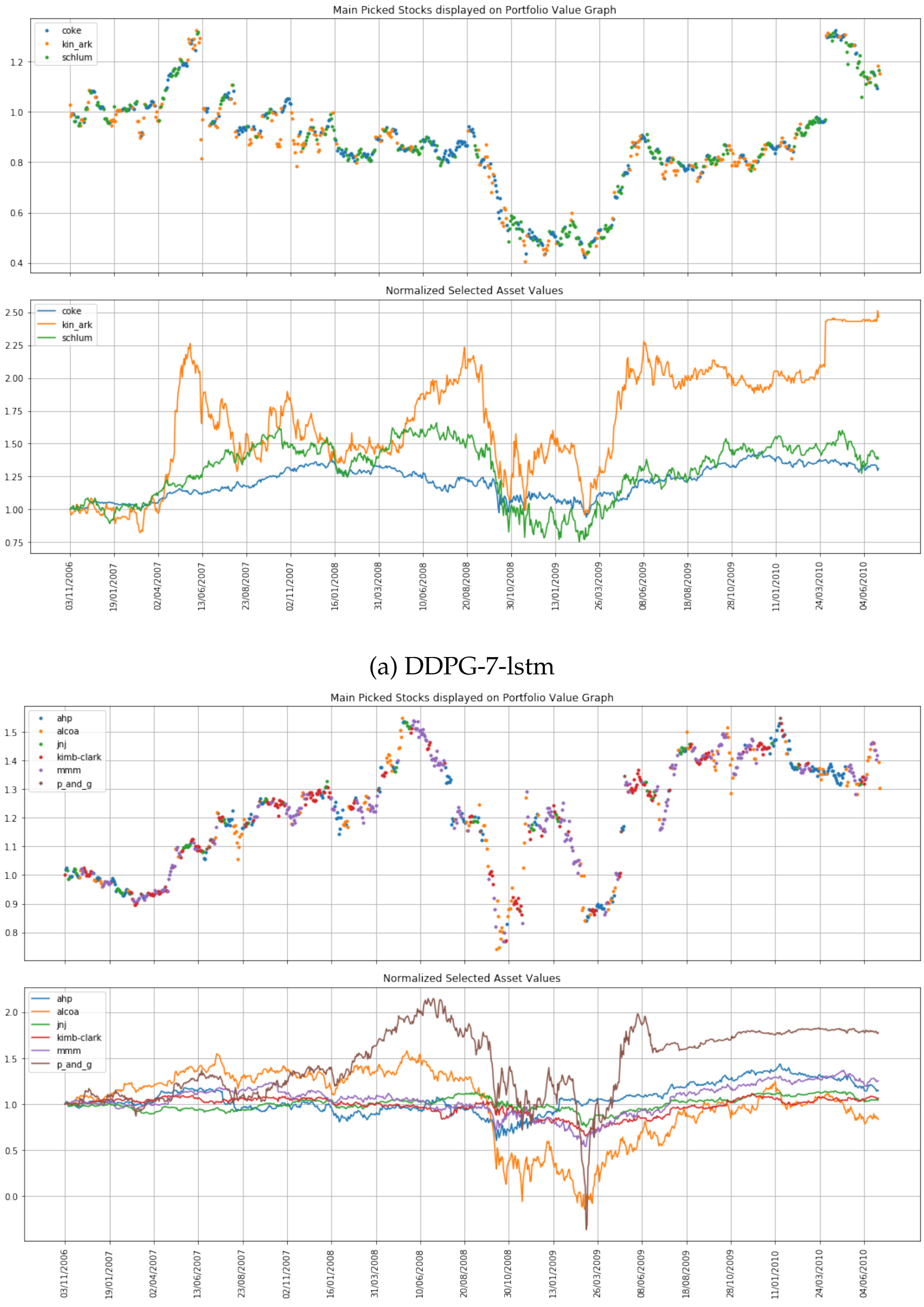


(a) *DDPG-7-lstm*



(b) *TD3-14-lstm*

Figure 4.4: Pie charts visualising the average allocation of portfolio weights on the *NYSE(N)* test dataset.



(a) DDPG-7-lstm

(b) TD3-14-lstm

Figure 4.5: Plots of the main stocks picked by the agents on the *NYSE(N)* test dataset, superimposed on the portfolio value graph, and the corresponding asset values normalised to the same starting point.

4.2.3 | SP500 Testing Results

To compare the performance of the two DRL frameworks on the *SP500* dataset, we use the best performing models on the in-sample dataset for each corresponding framework. In this case, both models consisted of a window length of 11. Therefore, the models selected are *DDPG-11-lstm* and *TD3-11-lstm*. The portfolio values are visualised in Figure 4.6, along with those obtained by the OLPS algorithms. Additionally, the evaluation criteria results obtained by the two identified models are shown in Table 4.3. Finally, the mean and standard deviation of the evaluation criteria results obtained across all window sizes are shown in Table 4.4.

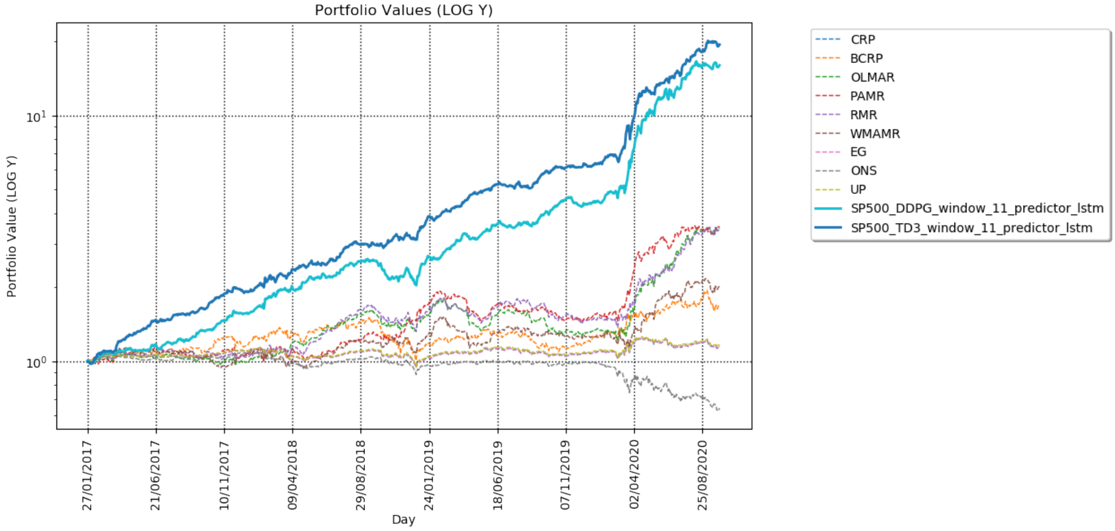


Figure 4.6: Portfolio values for *DDPG-11-lstm* and *TD3-11-lstm* models on the *SP500* test dataset along with OLPS algorithms.

Model	Average Daily Yield (%)	Sharpe Ratio (%)	Sortino Ratio (%)	Maximum Drawdown (%)	Final Portfolio Value
DDPG-11-lstm	0.31	21.21	35.0	21.945	15.985
TD3-11-lstm	**0.331**	**24.852**	**43.255**	**12.265**	**19.412**
Market Value (UCRP)	0.017	2.312	2.815	17.072	1.14
BCRP	0.07	4.211	5.625	33.206	1.671
OLMAR	0.145	9.702	13.834	32.493	3.481
PAMR	0.146	10.508	15.173	24.744	3.506
RMR	0.144	9.725	14.117	24.196	3.451
WMAMR	0.086	5.803	8.109	26.658	2.023
EG	0.017	2.346	2.852	17.111	1.143
ONS	-0.044	-4.256	-5.281	41.92	0.633
UP	0.018	2.506	3.046	16.63	1.154

Table 4.3: The evaluation criteria results for *DDPG-11-lstm* and *TD3-11-lstm* models on the *SP500* test dataset along with OLPS algorithms. The best value for each criteria is bold.

Model	Average Daily Yield (%)	Sharpe Ratio (%)	Sortino Ratio (%)	Maximum Drawdown (%)	Final Portfolio Value
DDPG models	0.085 ± 0.17	5.558 ± 11.799	9.125 ± 18.683	35.468 ± 20.746	4.903 ± 7.412
TD3 models	0.078 ± 0.169	5.812 ± 12.713	10.317 ± 21.975	31.984 ± 13.616	5.515 ± 9.266

Table 4.4: Result comparison of DDPG and TD3 models on the *SP500* test dataset over all window sizes.

4.3 | Experiment 2 Results

In the second experiment, we address our second objective by including the results from the enhanced moving average equation used in RMR within the State format of our DRL models, along with the windowed normalised logarithmic returns used in the previous experiment. This is applied on models with all the window lengths, but only using the TD3 framework. TD3 is the chosen DRL framework due to its performance in the previous experiment. Therefore, four new models are trained, tested and evaluated on the *NYSE(N)* and *SP500* datasets. Likewise, the four new model names are created using the RL algorithm (e.g. DDPG), the window length (e.g. 3), and the Neural Network format (e.g. lstm). These also include a tag referring to the enhancement (e.g. rmr). These models consist of *TD3-3-lstm-rmr*, *TD3-7-lstm-rmr*, *TD3-11-lstm-rmr*, and *TD3-14-lstm-rmr*. We hypothesise that the inclusion of stock movement prediction indicators within the state format has a positive effect on the performance of the DRL-based portfolio optimisation models.

4.3.1 | NYSE(N) Testing Results

To compare the performance of the two state formats on the *NYSE(N)* dataset, the best model in the previous experiment (*TD3-14-lstm*) is compared against its counterpart, which makes use of the enhanced state format (*TD3-14-lstm-rmr*). The portfolio values are visualised in Figure 4.7, along with those obtained by the OLPS algorithms. Additionally, the results obtained are shown in Table 4.5. Finally, the mean and standard deviation of the evaluation criteria results obtained across the window sizes are shown in Table 4.6. The models with window lengths of 11 were excluded in this table due to poor performance. Despite the increased volatility in *TD3-14-lstm-rmr* granting it a worse maximum drawdown score, the model achieves greater results on the rest of the evaluation criteria.

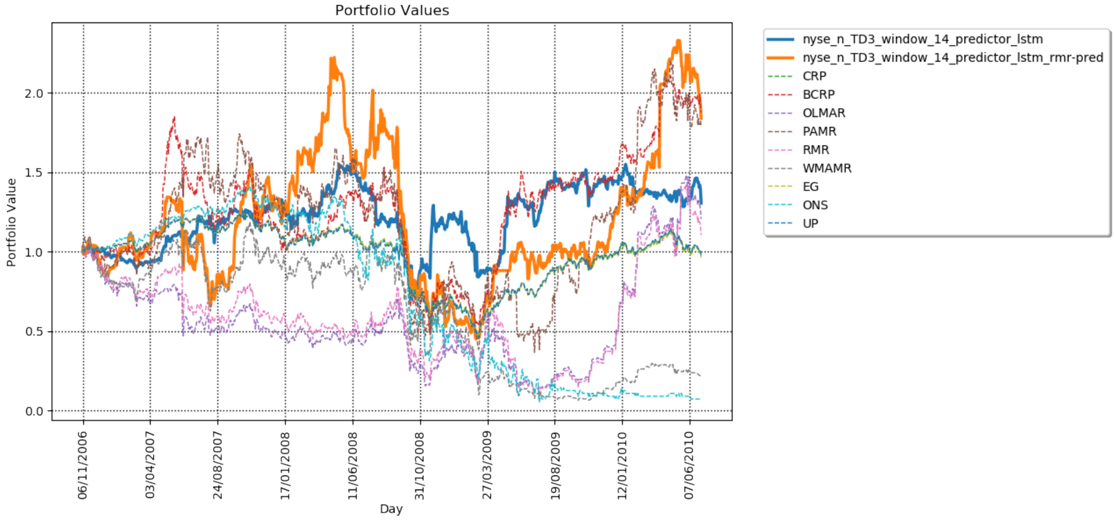


Figure 4.7: Portfolio values for *TD3-14-lstm* and *TD3-14-lstm-rmr* on the *NYSE(N)* test dataset along with benchmark OLPS algorithms.

Model	Average Daily Yield (%)	Sharpe Ratio (%)	Sortino Ratio (%)	Maximum Drawdown (%)	Final Portfolio Value
TD3-14-lstm	0.059	2.382	3.301	**52.122**	1.305
TD3-14-lstm-rmr	**0.151**	**3.584**	**5.264**	79.801	**1.838**

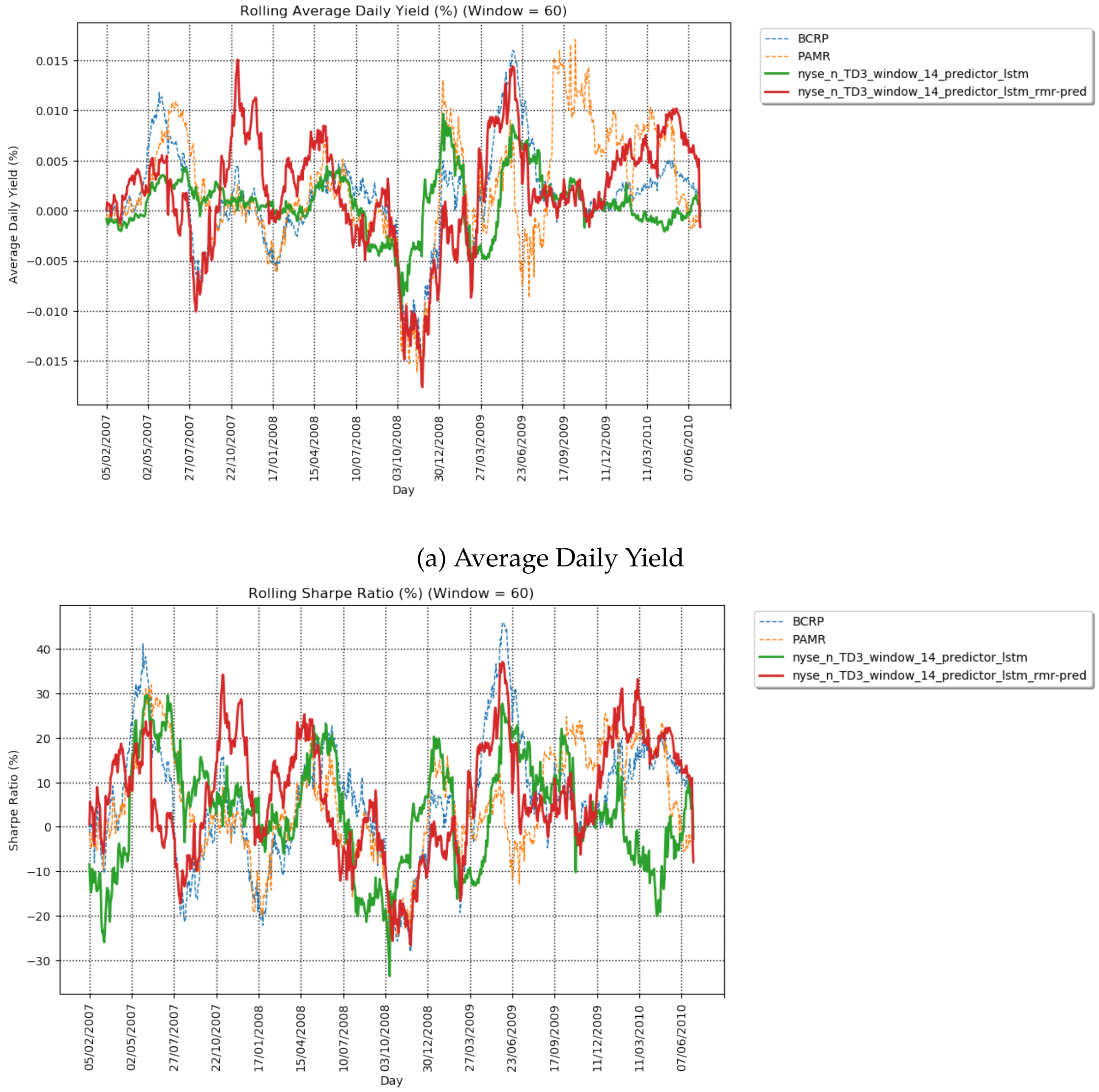
Table 4.5: The evaluation criteria results for *TD3-14-lstm* and *TD3-14-lstm-rmr* models on the *NYSE(N)* test dataset along with benchmark OLPS algorithms. The best value for each criteria is bold.

Model	Average Daily Yield (%)	Sharpe Ratio (%)	Sortino Ratio (%)	Maximum Drawdown (%)	Final Portfolio Value
TD3 models	0.04 ± 0.075	1.778 ± 2.565	2.664 ± 3.67	69.712 ± 22.371	1.166 ± 0.812
TD3-rmr models	0.15 ± 0.046	4.565 ± 2.01	6.95 ± 3.653	58.511 ± 23.606	2.559 ± 1.401

Table 4.6: Result comparison of TD3 and TD3-rmr models on the *NYSE(N)* test dataset. (Excluding window 11 due to poor performance)

4.3.1.1 | Further statistical testing

To further evaluate the performance of our models and compare against the OLPS benchmark algorithms, we calculate the rolling window values for each evaluation criteria with a window of 60 days/steps. Sixty steps in our dataset are approximately a quarter of a year (3 months) in trading days. This allows us to test the stability of our models on the test datasets and evaluate if the results obtained are achieved by chance or possibly by great performance in short intermittent periods. Naturally, from an investor perspective, stable performance is key (Markowitz, 1952). We visualise a number of these results in Figure 4.8 and use the mean and standard deviation of the acquired data in t-tests to compare the enhanced model (*TD3-14-lstm-rmr*) with the corresponding model from the previous experiment (*TD3-14-lstm*) and against all benchmarks. The t-test results obtained are shown in Table 4.7.



(a) Average Daily Yield

(b) Sharpe Ratio

Figure 4.8: Rolling window analysis on the *NYSE(N)* dataset, comparing the performance of *TD3-14-lstm* and *TD3-14-lstm-rmr* with two of the best performing benchmarks.

Table 4.7: T-test results between the rolling evaluation criteria results obtained by *TD3-14-lstm-rmr* and the benchmark OLPS algorithms on *NYSE(N)* dataset.

Model	Average Daily Yield		Sharpe Ratio		Sortino Ratio		Maximum Drawdown	
	t-stat	p-value	t-stat	p-value	t-stat	p-value	t-stat	p-value
CRP	8.181	5.196e-16	4.237	2.375e-05	9.254	5.822e-20	18.643	3.617e-71
BCRP	2.356	1.855e-02	1.793	7.319e-02	3.475	5.233e-04	6.903	6.987e-12
OLMAR	-2.068	3.876e-02	5.612	2.311e-08	1.143	2.531e-01	-12.047	3.207e-32
PAMR	-0.806	4.206e-01	3.289	1.025e-03	3.269	1.101e-03	-3.290	1.02e-03
RMR	-1.816	6.956e-02	5.231	1.878e-07	1.232	2.181e-01	-11.686	1.763e-30
WMAMR	4.358	1.385e-05	7.484	1.109e-13	3.589	3.401e-04	-11.439	2.572e-29
EG	8.255	2.88e-16	4.294	1.846e-05	9.370	2.053e-20	18.779	4.237e-72
ONS	6.888	7.758e-12	8.164	5.961e-16	11.142	6.048e-28	-8.305	1.907e-16
UP	8.186	5.015e-16	4.146	3.531e-05	9.204	9.100e-20	18.806	2.755e-72
TD3-14-lstm	8.186	1.425e-08	6.436	1.561e-10	5.160	2.733e-07	17.193	1.543e-61

4.3.2 | SP500 Testing Results

To compare the performance of the two state formats on the *SP500* dataset, we use the best performing models on the in-sample dataset for each state format. These are *TD3-11-lstm* and *TD3-3-lstm-rmr*. The portfolio values are visualised in Figure 4.7, along with those obtained by the OLPS algorithms. Additionally, the results obtained are shown in Table 4.8. Finally, the mean and standard deviation of the evaluation criteria results obtained across the window sizes are shown in Table 4.9.

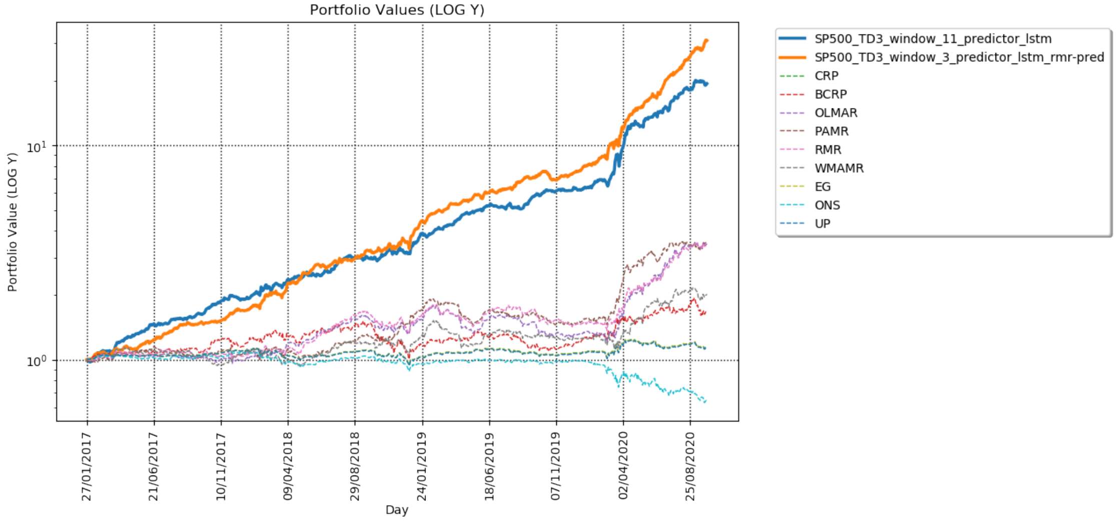


Figure 4.9: Portfolio values for *TD3-11-lstm* and *TD3-3-lstm-rmr* models on the S&P500 test dataset along with OLPS algorithms.

Model	Average Daily Yield (%)	Sharpe Ratio (%)	Sortino Ratio (%)	Maximum Drawdown (%)	Final Portfolio Value
TD3-3-lstm-rmr	**0.379**	**30.306**	**49.204**	**10.848**	**30.843**
TD3-11-lstm	0.331	24.852	43.255	12.265	19.412
Market Value (UCRP)	0.017	2.312	2.815	17.072	1.140
BCRP	0.070	4.211	5.625	33.206	1.671
OLMAR	0.145	9.702	13.834	32.493	3.481
PAMR	0.146	10.508	15.173	24.744	3.506
RMR	0.144	9.725	14.117	24.196	3.451
WMAMR	0.086	5.803	8.109	26.658	2.023
EG	0.017	2.346	2.852	17.111	1.143
ONS	-0.044	-4.256	-5.281	41.920	0.633
UP	0.018	2.506	3.046	16.63	1.154

Table 4.8: The evaluation criteria results for *TD3-11-lstm* and *TD3-3-lstm-rmr* models on the S&P500 test dataset along with OLPS algorithms. The best value for each criteria is bold.

Model	Average Daily Yield (%)	Sharpe Ratio (%)	Sortino Ratio (%)	Maximum Drawdown (%)	Final Portfolio Value
TD3 models	0.078 ± 0.169	5.812 ± 12.713	10.317 ± 21.975	31.984 ± 13.616	5.515 ± 9.265
TD3-rmr models	0.251 ± 0.142	18.853 ± 10.1	29.754 ± 16.981	18.365 ± 5.983	15.263 ± 13.481

Table 4.9: Result comparison of TD3 and TD3-rmr models on the S&P500 test dataset.

Both *TD3-11-lstm* and *TD3-3-lstm-rmr* achieve greater overall results than the OLPS algorithms. Additionally, *TD3-3-lstm-rmr* beats *TD3-11-lstm* on all evaluation criteria. The statistical analysis done for the *NYSE(N)* dataset is excluded for the *SP500* dataset, due to the conclusive results currently obtained.

4.4 | Experiment 3 Results

In the third experiment, we address the third objective by combining the four models trained in the previous experiment (*TD3-3-lstm-rmr*, *TD3-7-lstm-rmr*, *TD3-11-lstm-rmr*, and *TD3-14-lstm-rmr*) into one via an ensemble strategy. The proposed ensemble algorithm makes use of a window or observation length w to make use of the best performing agent in terms of *average return* at each step. Throughout our experiment, w is set to 21, which is approximately one month in trading days. We hypothesise that the combination of multiple DRL-models with differing state formats into one ensemble model improves portfolio optimisation performance.

4.4.1 | NYSE(N) Testing Results

The portfolio values achieved by the ensemble model are visualised along with its' parent models and benchmark OLPS algorithms in Figure 4.10. The daily portfolio choices are visualised in Figure 4.11. This is accompanied by a plot of the portfolio values from the parent DRL models for reference. The evaluation criteria for the portfolio results obtained are available in Table 4.10. The ensemble model achieves a greater average daily yield, Sharpe ratio, Sortino ratio and final portfolio value than all our DRL models and OLPS algorithms, although the *TD3-3-lstm-rmr* model had a better maximum drawdown score.

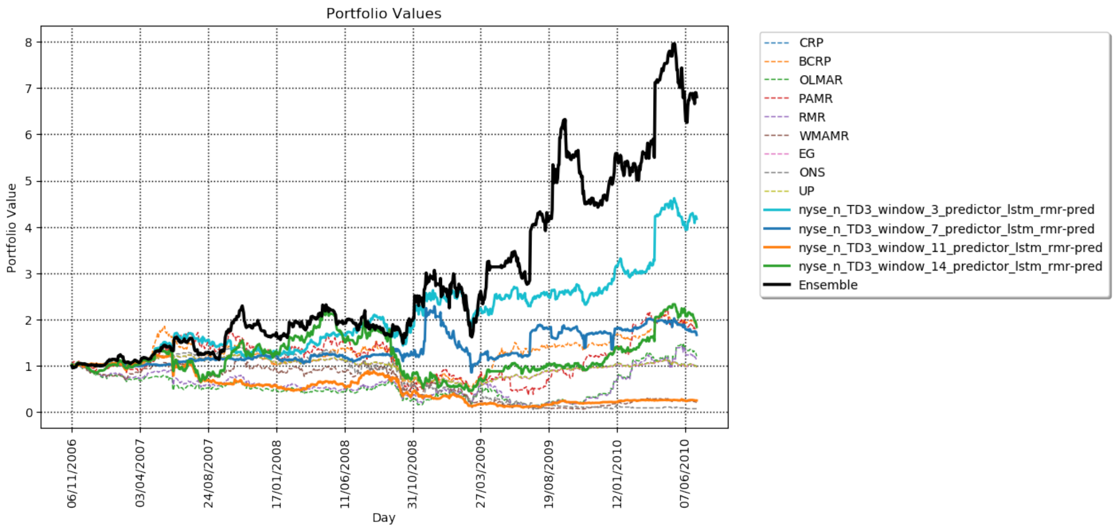


Figure 4.10: Portfolio values for the ensemble model along with our DRL models from the previous experiment, along with OLPS algorithms on the *NYSE(N)* test dataset.

Model	Average Daily Yield (%)	Sharpe Ratio (%)	Sortino Ratio (%)	Maximum Drawdown (%)	Final Portfolio Value
Ensemble	**0.276**	**7.387**	**12.703**	47.108	**6.810**
TD3-3-lstm-rmr	0.195	6.877	11.145	**33.125**	4.174
TD3-7-lstm-rmr	0.103	3.233	4.448	62.606	1.670
TD3-11-lstm-rmr	-0.066	-1.611	-2.035	92.460	0.250
TD3-14-lstm-rmr	0.151	3.584	5.264	79.801	1.838
Market Value (UCRP)	0.015	0.768	0.982	63.279	0.970
BCRP	0.114	3.737	5.803	73.208	1.876
OLMAR	0.220	3.414	4.678	91.443	1.206
PAMR	0.182	3.666	5.072	79.034	1.797
RMR	0.210	3.260	4.527	91.612	1.104
WMAMR	0.024	0.377	0.478	94.642	0.206
EG	0.013	0.711	0.907	63.122	0.962
ONS	0.006	0.076	0.085	96.181	0.072
UP	0.013	0.696	0.892	63.490	0.958

Table 4.10: The evaluation criteria results for each model on the *NYSE(N)* test dataset. The best value for each criteria is bold.

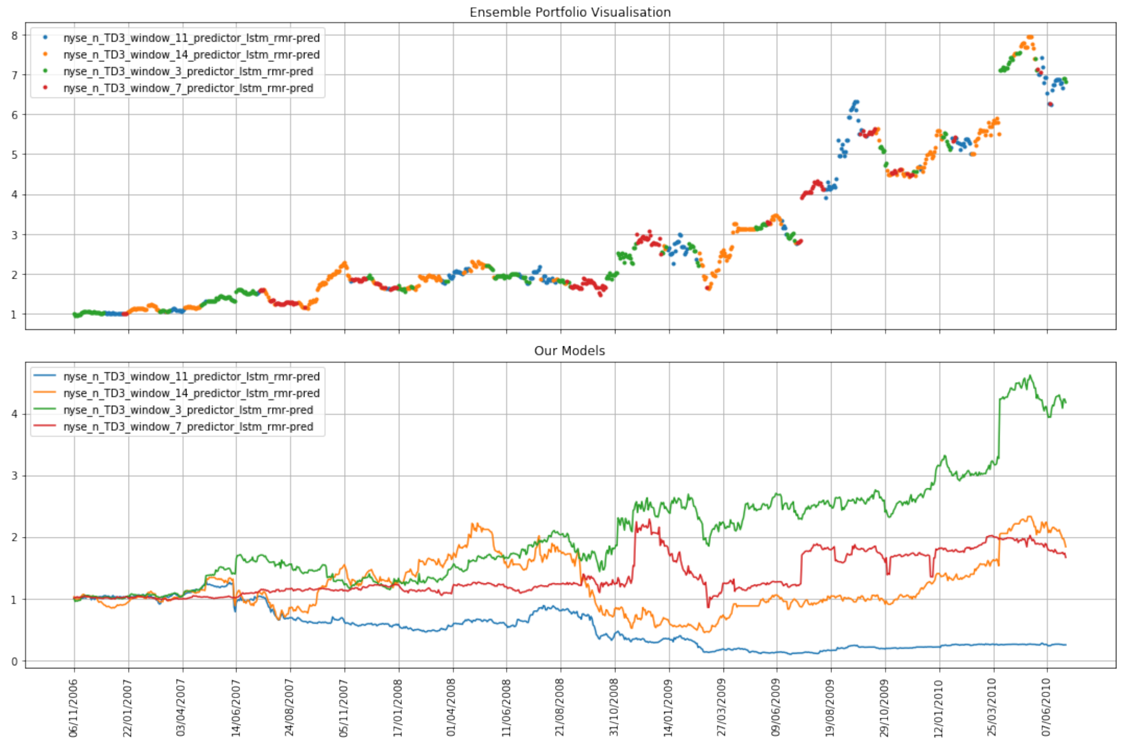
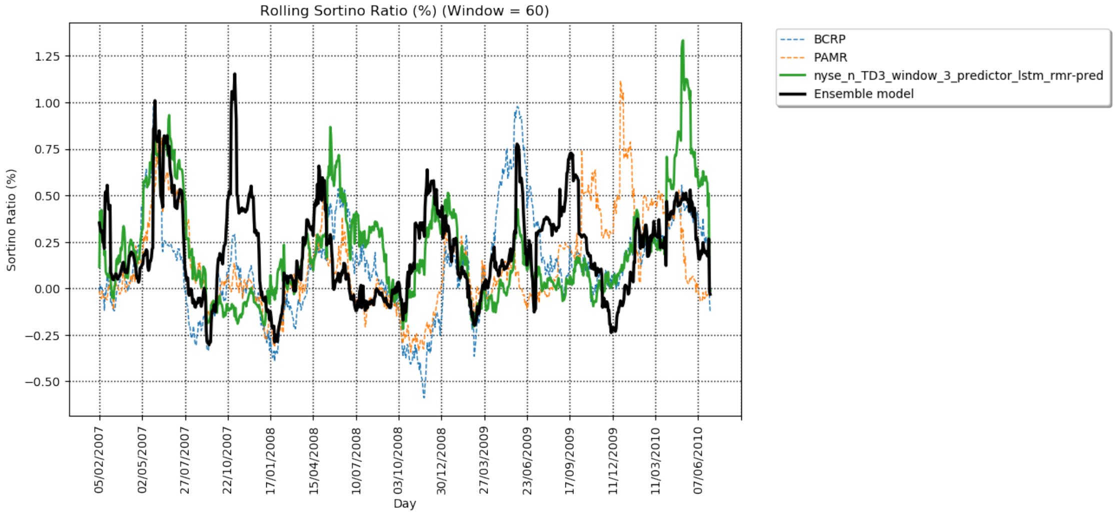


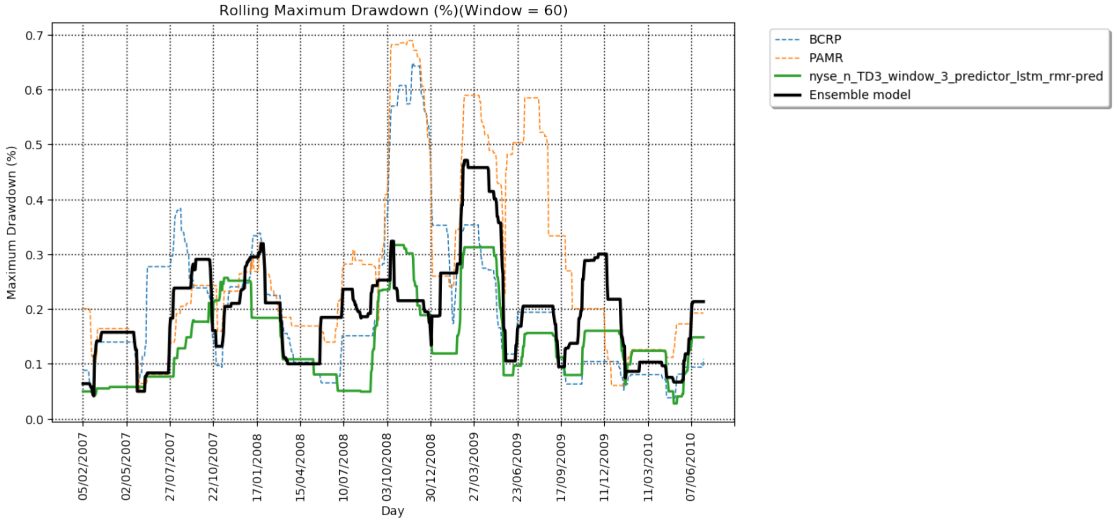
Figure 4.11: Plots of the portfolio choices made by the ensemble agent on the *NYSE(N)* test dataset, superimposed on the ensemble portfolio value graph, with the corresponding portfolio values normalised to the same starting point.

4.4.1.1 | Further statistical testing

To further evaluate the performance of our ensemble model, along with our DRL models and the OLPS algorithms, we calculate the rolling window values for each evaluation criteria with a window of 60 days/steps. We visualise a number of these results in Figure 4.12 and use the mean and standard deviation of the acquired data in t-tests to compare the portfolio generated by the ensemble agent against the parent DRL models and the benchmark OLPS algorithms.



(a) Sortino Ratio



(b) Maximum Drawdown

Figure 4.12: Rolling window analysis on the *NYSE(N)* dataset, comparing the performance of the ensemble model with two of the best performing benchmarks, and one of the best performing DRL models.

Table 4.11: T-test results between the rolling evaluation criteria results obtained by our ensemble model, and that of the models created in the previous experiment along with the benchmark and OLPS algorithms on the *NYSE(N)* test dataset.

Model	Average Daily Yield		Sharpe Ratio		Sortino Ratio		Maximum Drawdown	
	t-stat	p-value	t-stat	p-value	t-stat	p-value	t-stat	p-value
CRP	17.23	8.745e-62	5.704	1.363e-08	13.23	3.002e-38	13.63	2.353e-40
BCRP	7.942	3.45e-15	3.087	0.002053	6.74	2.111e-11	-0.7948	0.4268
OLMAR	1.303	0.1926	7.167	1.105e-12	3.504	0.0004698	-20.64	2.859e-85
PAMR	3.689	0.0002314	4.806	1.665e-06	6.7	2.761e-11	-12.05	3.251e-32
RMR	1.562	0.1185	6.767	1.762e-11	3.646	0.0002733	-20.09	2.529e-81
WMAMR	9.119	1.934e-19	9.215	8.271e-20	6.402	1.949e-10	-19.85	1.302e-79
EG	17.34	1.799e-62	5.764	9.631e-09	13.36	6.382e-39	13.81	2.529e-41
ONS	13.27	1.912e-38	10.09	2.464e-23	15.25	1.534e-49	-15.01	4.294e-48
UP	17.19	1.559e-61	5.652	1.832e-08	13.22	3.439e-38	13.49	1.277e-39
TD3-3-lstm-rmr	5.147	2.933e-07	-0.1003	0.9201	-0.08957	0.9286	13.22	3.61e-38
TD3-7-lstm-rmr	9.717	8.442e-22	6.217	6.246e-10	7.93	3.78e-15	8.535	2.882e-17
TD3-11-lstm-rmr	15.78	1.074e-52	13.07	2.099e-37	14.34	2.756e-44	-12.04	3.294e-32
TD3-14-lstm-rmr	4.781	1.885e-06	1.269	0.2046	3.208	0.001357	-8.66	1.015e-17

4.4.2 | SP500 Testing Results

An ensemble model is similarly created for the *SP500* dataset, using *TD3-3-lstm-rmr*, *TD3-7-lstm-rmr*, *TD3-11-lstm-rmr*, and *TD3-14-lstm-rmr*. The portfolio values achieved by the ensemble model are visualised along with the parent models and benchmark OLPS algorithms in Figure 4.13. Furthermore, the daily portfolio choices are visualised in Figure 4.14. This is accompanied by a plot of the portfolio values from the parent DRL models for reference. The evaluation criteria for the portfolio results obtained are available Table 4.12.

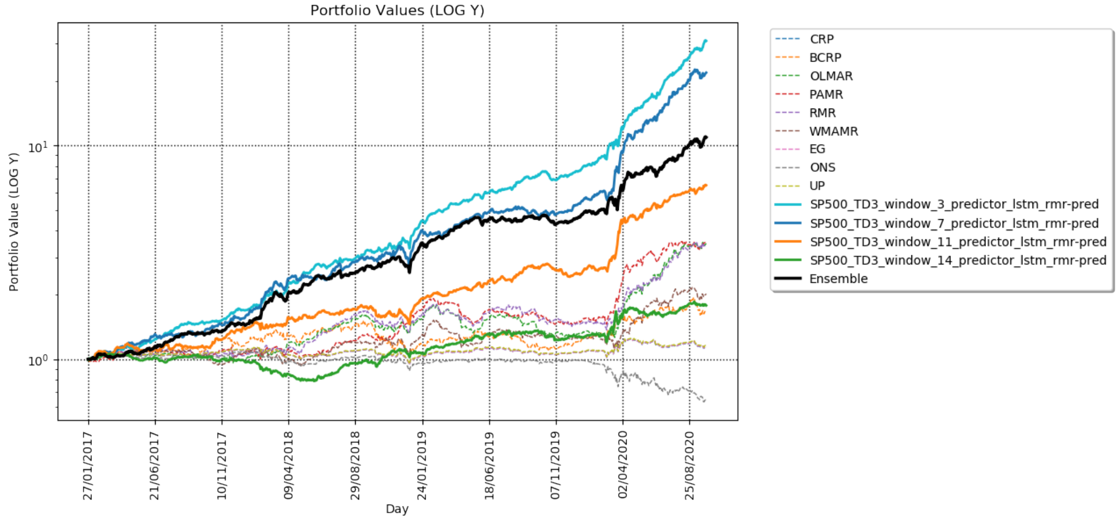


Figure 4.13: Portfolio values for the ensemble model along with our DRL models from the previous experiment and the ones acquired by OLPS algorithms on the *SP500* test dataset.

Model	Average Daily Yield (%)	Sharpe Ratio (%)	Sortino Ratio (%)	Maximum Drawdown (%)	Final Portfolio Value
Ensemble	0.268	19.775	29.750	15.724	10.917
TD3-3-lstm-rmr	**0.379**	**30.306**	**49.204**	**10.848**	**30.843**
TD3-7-lstm-rmr	0.346	22.460	35.208	18.581	21.913
TD3-11-lstm-rmr	0.211	16.246	25.876	18.537	6.513
TD3-14-lstm-rmr	0.069	6.401	8.730	25.494	1.785
Market Value (UCRP)	0.017	2.312	2.815	17.072	1.140
BCRP	0.070	4.211	5.625	33.206	1.671
OLMAR	0.145	9.702	13.834	32.493	3.481
PAMR	0.146	10.508	15.173	24.744	3.506
RMR	0.144	9.725	14.117	24.196	3.451
WMAMR	0.086	5.803	8.109	26.658	2.023
EG	0.017	2.346	2.852	17.111	1.143
ONS	-0.044	-4.256	-5.281	41.920	0.633
UP	0.018	2.506	3.046	16.63	1.154

Table 4.12: The evaluation criteria results for each model on the *SP500* test dataset. The best value for each criteria is bold.

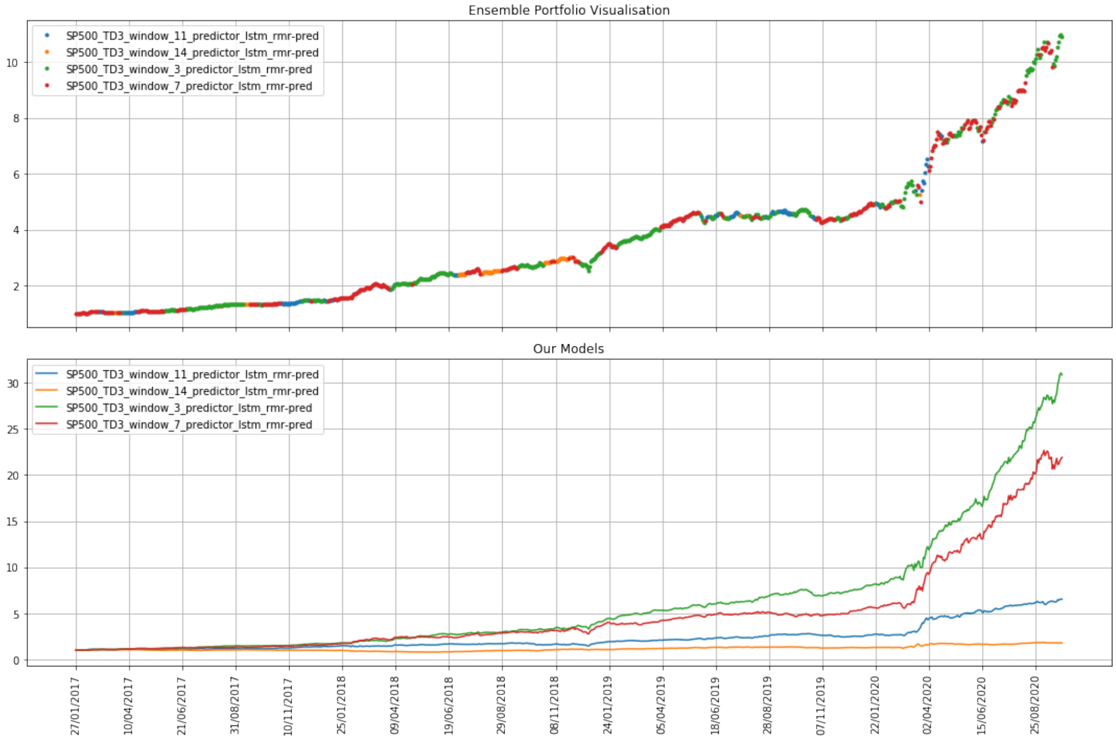
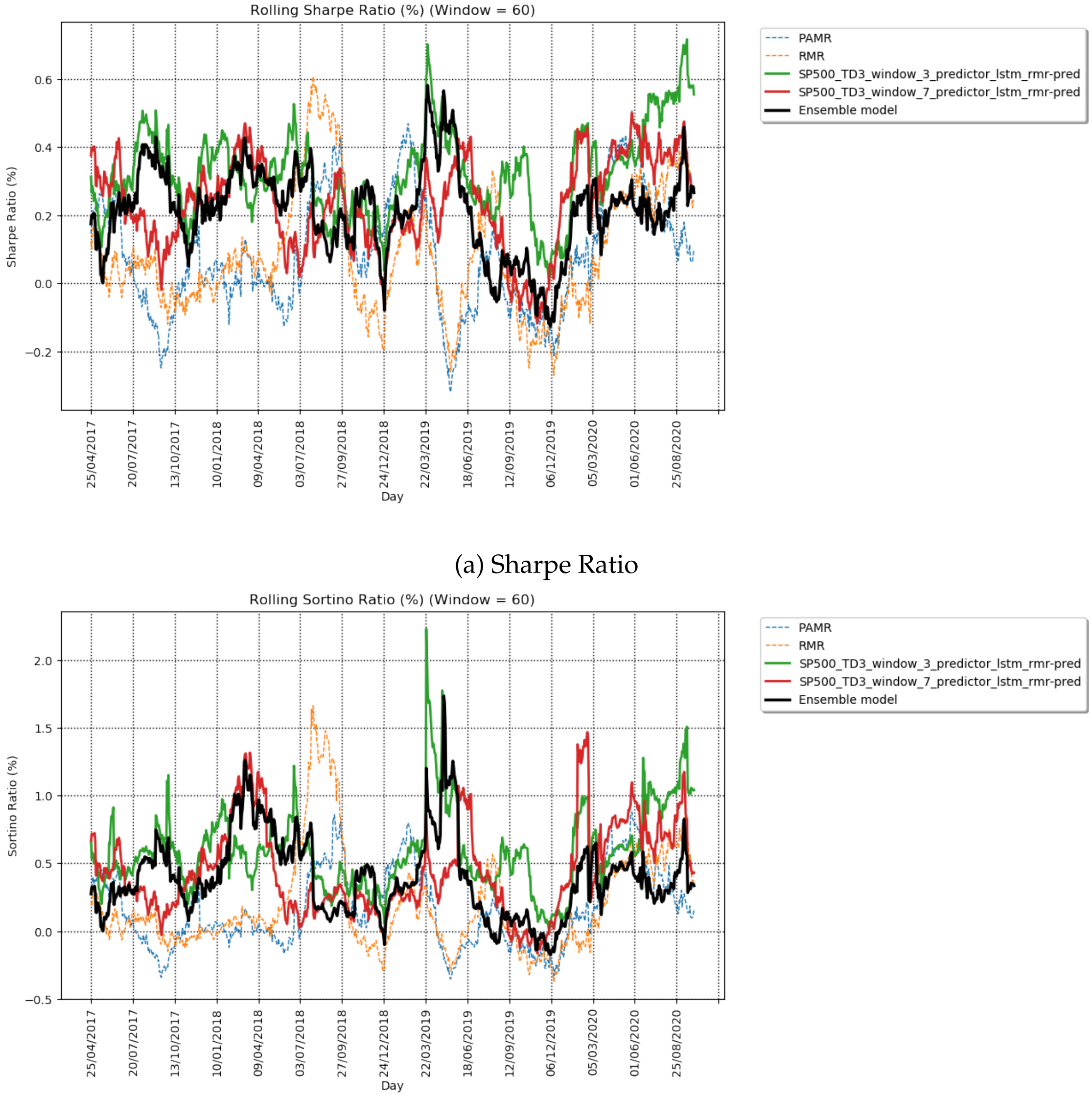


Figure 4.14: Plots of the portfolio choices made by the ensemble agent on the *SP500* test dataset, superimposed on the ensemble portfolio value graph, with the corresponding portfolio values normalised to the same starting point.

4.4.2.1 | Further statistical testing

To further evaluate the performance of our ensemble model, along with our DRL models and the OLPS algorithms, we calculate the rolling window values for each evaluation criteria with a window of 60 days/steps. Finally, we visualise a number of these results in Figure 4.12 and use the mean and standard deviation of the acquired data in t-tests to compare the portfolio generated by the ensemble agent against all benchmark OLPS algorithms.



(a) Sharpe Ratio

(b) Sortino Ratio

Figure 4.15: Rolling window analysis on the *SP500* dataset, comparing the performance of the ensemble model with two of the best performing benchmarks, and two of the best performing DRL models.

Table 4.13: T-test results between the rolling evaluation criteria results obtained by our ensemble model and the benchmark OLPS algorithms on the *SP500* test dataset.

Model	Average Daily Yield		Sharpe Ratio		Sortino Ratio		Maximum Drawdown	
	t-stat	p-value	t-stat	p-value	t-stat	p-value	t-stat	p-value
CRP	36.62	2.911e-221	25.38	3.751e-122	24.17	2.055e-112	11.54	8.312e-30
BCRP	23.3	1.661e-105	29.63	2.872e-158	27.45	1.756e-139	-12.19	6.149e-33
OLMAR	10.14	1.492e-23	17.46	2.58e-63	13.73	6.167e-41	-9.972	7.55e-23
PAMR	9.904	1.446e-22	17.49	1.622e-63	17.15	2.682e-61	-9.148	1.49e-19
RMR	11.59	4.927e-30	16.91	8.884e-60	12.44	3.568e-34	-11.91	1.467e-31
WMAMR	18.89	6.552e-73	25.78	1.736e-125	21.78	8.713e-94	-14.16	2.747e-43
EG	36.56	9.441e-221	25.37	4.154e-122	24.22	8.896e-113	11.52	1.023e-29
ONS	42.74	3.294e-278	39.48	7.878e-248	33.7	1.529e-194	-2.371	0.01782
UP	36.46	9.234e-220	25	4.472e-119	23.77	3.198e-109	11.68	1.745e-30
TD3-3-lstm-rmr	-10.66	8.831e-26	-18.63	3.972e-71	-14.28	6.044e-44	13	4.787e-37
TD3-7-lstm-rmr	-6.77	1.723e-11	-3.438	0.000598	-3.479	0.0005156	0.7714	0.4406
TD3-11-lstm-rmr	5.756	1.008e-08	9.091	2.456e-19	7.774	1.252e-14	0.6315	0.5278
TD3-14-lstm-rmr	23.96	9.908e-111	23.02	2.464e-103	21.86	2.09e-94	-1.424	0.1545

4.5 | Evaluation & Discussion

In this section, we evaluate the results obtained in our experiments with regards to the research objectives and corresponding hypotheses defined.

4.5.1 | Experiment 1

The first experiment addressed the first research objective, which involved investigating how the DRL-based portfolio optimisation models perform compared to OLPS algorithm benchmarks, and whether models using the TD3 framework perform better than those with DDPG. Below, we highlight each hypothesis pertaining to this experiment and evaluate the results obtained.

- **Hypothesis:** The portfolio optimisation models using the TD3 DRL framework provide improved results over the ones using DDPG.

  The training results obtained (Figure 4.1) show the TD3 frameworks' superiority in the training phase over the DDPG framework. This was expected, as the results obtained by Fujimoto et al. (2018) on openAI Gym environments had shown similar results. When applying the models with the best in-sample performance for each framework on the test datasets, the TD3 based models were found to achieve greater results. This is true for both the *NYSE(N)* dataset, where the TD3 based model outperforms the DDPG based one on most of the evaluation criteria (Table 4.1), and the *SP500* dataset, where the TD3 based model outperforms the DDPG based one on all criteria (Table 4.3). Additionally, through further analysis of the generated portfolios, the TD3 based portfolio was found to be more diversified, and consisted of more frequent weight shifts (Figures 4.4 and 4.5). Based on these results obtained, we conclude that the models using the TD3 framework do, in fact, perform better than those with DDPG. Despite this, within the result comparisons which take all the trained models into consideration (Tables 4.2 and 4.4), the standard deviation scores on certain evaluation criteria shed light on the inconsistent outcomes achieved on different window lengths. The enhancement in the state format discussed and evaluated within the second experiment could address this inconsistency between performances on different window lengths.

- **Hypothesis:** The results obtained by our DRL portfolio optimisation models exceed those of the benchmarks.

  This hypothesis fails on the *NYSE(N)* dataset due to BCRP achieving the overall best Sharpe ratio, Sortino ratio and final portfolio value, and OLMAR achieving the overall best average daily yield. Despite this, *TD3-14-lstm* does achieve the best maximum drawdown due to the superior performance during the crash period in the test dataset (Table 4.1). On the other hand, the results obtained on the *SP500* dataset, shown in Table 4.8, indicates that the *TD3-11-lstm* outperformed all the benchmarks on all the evaluation criteria observed. The difference in benchmark performance between datasets could be attributed to possible dataset bias on the *NYSE(N)* dataset (Nazir, 2019). This is due to the dataset being a standard one used in numerous studies. The recurring use of the same standard dataset could possibly cause the algorithms to be tailored towards the dataset itself rather than the problem in general. Dataset bias on these datasets is not proven, and the *NYSE(N)* dataset, along with others, still is considered a fair testing dataset (Li and Hoi, 2014). This hypothesis is to be revisited on the other experiments.

4.5.2 | Experiment 2

The second experiment attempted to improve the TD3 model from the first experiment to address the second research objective, which involves investigating whether the inclusion of the enhanced moving average equation used in RMR within the state format has a positive effect on the performance of the DRL-based portfolio optimisation models. Below, we highlight each hypothesis pertaining to this experiment and evaluate the results obtained.

- **Hypothesis:** The enhancement within the state format positively affects the performance of the DRL-based portfolio optimisation models.

  The adjustment in the state format was found to improve the winning model from the first experiment on the *NYSE(N)* dataset, as the portfolio generated achieved significantly greater rolling values of average daily yield, Sharpe ratio and Sortino ratio than its' predecessor on the *NYSE(N)* dataset (Tables 4.5 and 4.7). Conversely, the predecessor still had a greater rolling maximum drawdown value. Furthermore, on the *SP500* dataset, the *TD3-3-lstm-rmr* model outperformed the *TD3-11-lstm* model on all the evaluation criteria observed. Based on these results obtained, we conclude that the enhancement within the state format did, in fact, have a positive effect on the performance of the DRL-based portfolio optimisation models. Additionally, the result comparisons which take all the trained models into consideration (Tables 4.6 and 4.9), show a light decrease in the inconsistency previously discussed in the first experiment.

- **Hypothesis:** The results obtained by our DRL portfolio optimisation models exceed those of the benchmarks.

  The models with the improved state format generally outperform all the benchmarks on most evaluation criteria, as seen in Tables 4.7 and 4.8.

4.5.3 | Experiment 3

The third and final experiment introduces the ensemble framework that made use of the trained models from the second experiment to create a further optimised portfolio. This is done to address the third research objective, which involves investigating if the combination of multiple DRL-models into one ensemble model improves the overall performance. The ensemble model is tested on both the *NYSE(N)* and *SP500* datasets, as visualised in Figures 4.10 and 4.13.

- **Hypothesis:** The combination of multiple DRL-models with differing state formats into one ensemble model improves portfolio optimisation performance.

  On the *NYSE(N)* dataset, the ensemble portfolio surpassed all other models and benchmarks tested, except for *TD3-3-lstm-rmr* on the maximum drawdown criterion (Tables 4.10 and 4.11). Figure 4.11 serves as a proof of concept, showing the decisions being made daily by the ensemble algorithm. An area worth highlighting is the conversion of the light bump felt by the *TD3-11-lstm-rmr* around 19/08/2009 into a peak in the ensemble portfolio. Despite the ensemble algorithm not being as successful on the *SP500* dataset as on the *NYSE(N)* dataset, it still performed exceptionally well, as it outperforms all the OLPS algorithms, along with two of the DRL models created in the second experiment (Table 4.13).

- **Hypothesis:** The results obtained by our DRL portfolio optimisation models exceed those of the benchmarks.

  The ensemble models generally outperform all the benchmarks on most evaluation criteria, as seen in Tables 4.11 and 4.13.

5

Conclusion & Future Work

In this work, we have implemented an ensemble strategy for portfolio optimisation, using deep reinforcement agents trained with state formats consisting of different observation windows. These agents are built with the state-of-the-art TD3 DRL framework, proven to perform successfully in problem domains consisting of continuous action spaces. Our models are tested on two identified datasets, *NYSE(N)* and *SP500*. Before implementing the ensemble framework, agents created with the TD3 framework were compared with those created using its' predecessor, DDPG, which could be seen implemented in multiple solutions found in recent related literature. These agents used states with four different observation window sizes identified in the literature observed. The performance of these agents is also compared against portfolio selection OLPS algorithms. The results acquired suggested that the TD3-based models had acquired an enhanced portfolio performance on the *NYSE(N)* and *SP500* datasets. After an investigation on possible improvements on our models, we optimised the agents by implementing an enhanced state format, using an enhanced moving average function found in the RMR OLPS algorithm. This adjustment was proven to enhance the performance on the *TD3-14-lstm* model on the *NYSE(N)* dataset, and had a similarly successful results on the *SP500* dataset. These enhanced models were trained and used in our ensemble algorithm to be tested on the two datasets. The ensemble strategy was found to enhance performance in terms of Average daily yield, Sharpe Ratio and Sortino Ratio on the *NYSE(N)* dataset, but failed to beat all our previous models on the *SP500* dataset. Despite this, the ensemble models consistently achieved greater overall evaluation criteria results than the benchmarks and OLPS algorithms.

5.1 | Revisiting Aim and Objectives

Three objectives have been defined in Chapter 1, including the following:

1. **Investigate whether recent advances in DRL can lead to improved investment performance when compared to OLPS algorithms.**

   In the first experiment, we compared the TD3 against its predecessor, DDPG, in a portfolio optimisation setting using the *NYSE(N)* and *SP500* datasets. The experiment addressed the first objective and tested the hypotheses of whether:

   - The portfolio optimisation models using the TD3 DRL framework provide improved results over the ones using DDPG.
   - The results obtained by our DRL portfolio optimisation models exceeds those of the benchmarks.

   The results acquired suggest that the TD3-based models observed achieved better portfolio performance on both datasets. This is possibly attributed to the enhanced learning speed and further optimisation that the TD3 framework provided. The results obtained are shown in Tables 4.1 and 4.3. Based on these results obtained, we conclude that the corresponding hypothesis has been achieved and that, therefore, the portfolio optimisation models using the TD3 DRL framework provide improved results over the ones using DDPG. On the other hand, the models observed in the first experiment fail to consistently provide evaluation criteria scores greater than those from the benchmarks defined. Due to this, the second hypothesis fails.

2. **Investigate how OLPS features can be utilised to improve DRL market state representation, leading to increased investment performance.**

   The experiment in the second experiment addressed the second objective and tested the hypotheses of whether:

   - The enhancement within the state format positively affects the performance of the DRL-based portfolio optimisation models.
   - The results obtained by our DRL portfolio optimisation models exceeds those of the benchmarks.

   The inclusion of the enhanced moving average function from RMR within the session state was found to have a positive effect on the *TD3-14-lstm-rmr* and *TD3-3-lstm-rmr* models on the *NYSE(N)* and *SP500* datasets respectively. The models obtain greater overall evaluation criteria results than their counterparts missing the enhancement in the state format. This can be observed in Tables 4.7 and 4.8. Based on these results obtained, we conclude that the corresponding hypothesis has been achieved and that, therefore, the enhancement within the state format positively affects the performance of the DRL-based portfolio optimisation models. Furthermore, the models observed were found to achieve better overall evaluation criteria results than all the benchmarks and OLPS algorithms. Therefore, we conclude that the second hypothesis has been achieved and that hence, the results obtained by our DRL portfolio optimisation models exceed those of the benchmarks.

3. **Explore possible investment performance improvements through the use of DRL ensemble strategies.**

   In our paper, we are able to see the potential of the proposed ensemble DRL model to perform more profitable daily trades by selecting agents based on their performance in the current market. The third experiment addressed the third objective and tested the hypotheses of whether:

   - The combination of multiple DRL-models with differing state formats into one ensemble model improves portfolio optimisation performance.
   - The results obtained by our DRL portfolio optimisation models exceeds those of the benchmarks.

   The proposed ensemble DRL model fails to consistently achieve better results than the previous models observed, as *TD3-3-lstm-rmr* and *TD3-7-lstm-rmr* achieved better overall results than the ensemble model on the *SP500* test dataset. This can be observed in Tables 4.11 and 4.13. Therefore, the first hypothesis fails. The ensemble models achieved better overall evaluation criteria results than all the benchmark OLPS algorithms on both datasets. Therefore, we conclude that the second hypothesis has been achieved and that hence, the results obtained by our DRL portfolio optimisation models do, in fact, exceed those of the benchmarks.

5.2 | Contributions

The main contributions of this work, achieved with the completion of the highlighted objectives, include addressing three gaps found in literature by:

- Applying the TD3 DRL framework on the portfolio optimisation domain. Many of the existing literature opted for the use of its predecessor, DDPG, as discussed in Sections 1.3.1 and 2.5.1, and thus, we compare our implementation using the TD3 algorithm against an implementation using the DDPG algorithm. We make use of our implementation using the DDPG algorithm, and not extend one of the studies in recent literature observed due to the lack of a standard stock trading environment, which is required to accurately compare the performance of portfolio optimisation models (as discussed in Section 1.3.2).
- Applying OLPS functionality within the DRL state format, in our proposed DRL models. No state format has been identified as the best for this problem domain in literature observed, as discussed in Section 1.3.2. Additionally, none of the observed recent literature has attempted to use functions from OLPS algorithms within the state format to aid the DRL agent.
- Creating an ensemble portfolio optimisation model, with state format being the differing element between the combined DRL algorithms within. In contrast, literature observed had used DRL framework as the differing element. Additionally, DRL ensemble algorithms have been scarcely studied on the portfolio selection problem.

All the software developed for this work has been open-sourced and made publicly available on GitHub [1].

5.3 | Critique and Limitations

Whilst training all the models discussed, it was observed that the models using the TD3 DRL framework had required an increased training time. The increased training time could be attributed to the increased complexity in the critic network. This is counteracted by the *value function threshold*, as the TD3 based models managed to converge to an optimal result with less episodes than the DDPG models. In the second experiment, even though the inclusion of the RMR enhanced moving average function provided greater performance on some window-lengths, the *TD3-11-lstm-rmr* model under-performed on the *NYSE(N)* test dataset, and the *TD3-14-lstm-rmr* model under-performed on the *SP500* test dataset. It was noticed that these two models had possibly converged too soon to a local optimum and reached the *value function threshold* in a small number of training episodes.

[1] `https://github.com/NigelCusc/DDPG_TD3_PortfolioOptimization_tensorflow-1.15.4`

5.4 | Future Work

In our experiments, we evaluate our models on the *NYSE(N)* and *SP500* datasets using real asset data from the NYSE and S&P500 stock trading markets, respectively. Despite these being very popular markets, others exist, such as DJIA and cryptocurrencies. Therefore, the models observed in this work could be evaluated on additional datasets using real asset data from alternate stock trading markets. Additionally, new agents built using other DRL frameworks such as Proximal Policy Optimisation (PPO) and Advantage Actor-Critic (A2C) could be included within the ensemble model.

Additionally, one known limitation of this kind of work is the limited quantity of data available for training. A possible solution to this is to alter the environment to allow intra-day trading. Although this would increase the number of steps, it might still not be enough information. Another solution is to consider data augmentation strategies to create new time-series data for portfolio optimisation agents to train on.

In this work, we focused on the *Capital Growth Theory*, as we evaluated our models against OLPS algorithms instead of alternative methods based on the *Mean Variance Theory*. The study can be shifted into one focused on the *Mean Variance Theory* with a more risk-aware solution. This could be done by changing the reward function into a function such as Sharpe Ratio or Sortino Ratio.

Prior to the experiments, we proposed an enhancement to the RL state format. Due to time constraints, only one function was proposed and evaluated. Further enhancements could be included in the state format, expanding on what we have done in the second experiment. Throughout our experiments, the training performance of TD3 based models were not affected by the size of the state vector as much as the DDPG based models. This would suggest that the TD3 models could potentially benefit from increasing the observable state size. Furthermore, alternate functions used inside OLPS algorithms or technical indicators could be included within the state to optimise the solution further. Although, greater observation state sizes could slow down the training speed of the model.

Additionally, the models' training performance could be enhanced further by using a genetic algorithm in a pre-training phase, such as the one used by Gran (2019). Alternatively, studies such as the ones by Khadka et al. (2019), and Pourchot and Sigaud (2019) introduce frameworks that combine the TD3 with Evolutionary frameworks to create Evolutionary Reinforcement Learning frameworks. However, despite these algorithms showing great potential in their respective literature, they have not yet been evaluated on our problem domain.

5.5 | Final Remarks

At the time of this work, the TD3 DRL framework has been scarcely used in our problem domain. Throughout our experiments, the TD3-based agents with the state enhancement manage to achieve successful portfolios that beat the benchmarks and OLPS algorithms identified. The TD3 framework is compared against its predecessor, DDPG, which has been applied in various recent related studies. Ensemble algorithms have also been scarcely applied in portfolio selection. Unlike the recent related study observed, our ensemble algorithm combined DRL models with differing window lengths. Despite the ensemble algorithm not achieving the intended goal on both datasets, they show great potential as they consistently beat the identified benchmarks.

A

Installation Instructions

Our project was created on Ubuntu 20, using Anaconda. *PIP* and *Anaconda* requirements are available in both *.txt* and *.yml* format. You may choose one as you deem fit. These are:

- **requirements.txt** - `pip install -user -requirement requirements.txt`
- **tensor_keras_portfolio.yml** - `conda env create -f tensor_keras_portfolio.yml`

Further information can be found in the readme file, within the Github repository.

References

Agarwal, A., Hazan, E., Kale, S., and Schapire, R. E. Algorithms for portfolio management based on the Newton method. *ACM International Conference Proceeding Series*, 148(January 2006):9–16, 2006. doi: 10.1145/1143844.1143846.

Bellman, R. The theory of dynamic programming. Technical report, Rand corp santa monica ca, 1954.

Buhlmann, P. and Yu, B. Boosting with L2 loss: Regression and classification. *Journal of the American Statistical Association*, 98(324338), 2003.

Cao, H. K. and Cao, H. K. DELAFO: An Efficient Portfolio Optimization Using Deep Neural Networks. In *Pakdd*, volume 3, pages 512–523. Springer International Publishing, 2020. ISBN 9783030474263. doi: 10.1007/978-3-030-47426-3. URL `http://dx.doi.org/10.1007/978-3-030-47426-3{_}40`.

Cover, T. M. Universal Portfolios. *Department of Statistics and Electrical Engineering, Stanford University*, 1996.

Cumming, J. An Investigation into the Use of Reinforcement Learning Techniques within the Algorithmic Trading Domain. 2015.

Dankwa, S. and Zheng, W. Modeling a Continuous Locomotion Behavior of an Intelligent Agent Using Deep Reinforcement Technique. *2019 IEEE 2nd International Conference on Computer and Communication Engineering Technology, CCET 2019*, pages 172–175, 2019. doi: 10.1109/CCET48361.2019.8989177.

Deng, Y., Bao, F., Kong, Y., Ren, Z., and Dai, Q. Deep direct reinforcement learning for financial signal representation and trading. *IEEE Transactions on Neural Networks and Learning Systems*, 2017.

Emmert-Streib, F., Yang, Z., Feng, H., Tripathi, S., and Dehmer, M. An Introductory Review of Deep Learning for Prediction Models With Big Data. *Frontiers in Artificial Intelligence*, 3:4, 2020. doi: 10.3389/frai.2020.00004.

Estalayo, I., Ser, J. D., Osaba, E., Bilbao, M. N., Muhammad, K., Galvez, A., and Iglesias, A. Return, Diversification and Risk in Cryptocurrency Portfolios using Deep Recurrent Neural Networks and Multi-Objective Evolutionary Algorithms. *2019 IEEE Congress on Evolutionary Computation, CEC 2019 - Proceedings*, (Dl):755–761, 2019. doi: 10.1109/CEC.2019.8790121.

Fagiuoli, E., Stella, F., and Ventura, A. Constant rebalanced portfolios and side-information. *Quantitative Finance*, 7(2):161–173, 2007. ISSN 14697688. doi: 10.1080/14697680601157942.

Fujimoto, S., Van Hoof, H., and Meger, D. Addressing Function Approximation Error in Actor-Critic Methods. *35th International Conference on Machine Learning, ICML 2018*, 4:2587–2601, 2018.

Gao, L. and Zhang, W. Weighted Moving Average Passive Aggressive Algorithm for Online Portfolio Selection. *5th International Conference on Intelligent Human-Machine Systems and Cybernetics*, pages 327–330, 2013.

Gran, P. A Deep Reinforcement Learning Approach to Stock Trading. *Msc*, (June), 2019.

Greaves, J. Everything You Need to Know to Get Started in Reinforcement Learning, 2017. URL `https://joshgreaves.com/reinforcement-learning/introduction-to-reinforcement-learning/`.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80, pages 1861–1870, 2018.

Hayes, A., Mansa, J., and Munichiello, K. How Does the Stock Market Work?, 2021. URL `https://www.investopedia.com/articles/investing/082614/how-stock-market-works.asp`.

Heaton, J. B., Polson, N. G., and Witte, J. H. Deep learning for finance: deep portfolios. *Applied Stochastic Models in Business and Industry*, 33(1):3–12, 2017. ISSN 15264025. doi: 10.1002/asmb.2209.

Hegde, S., Kumar, V., and Singh, A. Risk aware portfolio construction using deep deterministic policy gradients. *Proceedings of the 2018 IEEE Symposium Series on Computational Intelligence, SSCI 2018*, (2017): 1861–1867, 2019. doi: 10.1109/SSCI.2018.8628791.

Helmbold, D. P., Schapire, R. E., Singer, Y., and Warmuth, M. K. On-line portfolio selection using multiplicative updates. *In Proceedings of the International Conference on Machine Learning*, pages 243–251, 1996.

Helmbold, D. P., Schapire, R. E., Singer, Y., and Warmuth, M. K. A comparison of new and old algorithms for a mixture estimation problem. *Machine Learning 27*, 1:97–119, 1997.

Helmbold, D. P., Schapire, R. E., Singer, Y., and Warmuth, M. K. On-line portfolio selection using multiplicative updates. *Mathematical Finance 8*, 4:325–347, 1998.

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., and Meger, D. Deep Reinforcement Learning that Matters. 2018.

Hochreiter, S. Untersuchungen zu dynamischen neuronalen Netzen. *Unpublished doctoral dissertation, Institut für Informatik, Technische Universität, Munchen*, pages 1–71, 1991. ISSN 18168957 18163459.

Hochreiter, S. and Schmidhuber, J. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 1997. ISSN 08997667. doi: 10.1162/neco.1997.9.8.1735.

Hu, Y. J. and Lin, S. J. Deep Reinforcement Learning for Optimizing Finance Portfolio Management. *Proceedings - 2019 Amity International Conference on Artificial Intelligence, AICAI 2019*, (Dl):14–20, 2019. doi: 10.1109/AICAI.2019.8701368.

Huang, D. J., Zhou, J., Li, B., Hoi, S. C., and Zhou, S. Robust Median Reversion Strategy for Online Portfolio Selection. *IEEE Transactions on Knowledge and Data Engineering*, 28(9):2480–2493, 2016. ISSN 10414347. doi: 10.1109/TKDE.2016.2563433.

Islam, R., Henderson, P., Gomrokchi, M., and Precup, D. Reproducibility of Benchmarked Deep Reinforcement Learning Tasks for Continuous Control. *Transportation Quarterly.*, 2017.

Jiang, Z. and Liang, J. Cryptocurrency portfolio management with deep reinforcement learning. *2017 Intelligent Systems Conference, IntelliSys 2017*, 2018-Janua:905–913, 2018. doi: 10.1109/IntelliSys.2017.8324237.

Jiang, Z., Xu, D., and Liang, J. A Deep Reinforcement Learning Framework for the Financial Portfolio Management Problem. (June), 2017.

Kang, Q., Zhou, H., and Kang, Y. An asynchronous advantage actor-critic reinforcement learning method for stock selection and portfolio management. *ACM International Conference Proceeding Series*, pages 141–145, 2018. doi: 10.1145/3291801.3291831.

Kanwar, N. Deep Reinforcement Learning-based Portfolio Management [m]. (May), 2019.

Kelly, J. L. A new interpretation of information rate. *Bell Systems Technical Journal*, 35:917–926, 1956.

Khadka, S., Majumdar, S., Nassar, T., Dwiel, Z., Tumer, E., Miret, S., Liu, Y., and Tumer, K. Collaborative Evolutionary Reinforcement Learning. 2019.

Khushi, M. and Meng, T. L. Reinforcement learning in financial markets. *Data*, 4(3):1–17, 2019. ISSN 23065729. doi: 10.3390/data4030110.

Kriesel, D. *A brief introduction to Neural Networks*, volume 34. 2007. doi: 10.1007/978-3-662-47484-6_1.

Kumar, I., Dogra, K., Utreja, C., and Yadav, P. A Comparative Study of Supervised Machine Learning Algorithms for Stock Market Trend Prediction. *Proceedings of the International Conference on Inventive Communication and Computational Technologies, ICICCT 2018*, (Icicct):1003–1007, 2018. doi: 10.1109/ICICCT.2018.8473214.

Li, B. and Hoi, S. C. On-line portfolio selection with moving average reversion. *Proceedings of the 29th International Conference on Machine Learning, ICML 2012*, 1:273–280, 2012.

Li, B. and Hoi, S. C. Online portfolio selection: A survey. *ACM Computing Surveys*, 46(3), 2014. ISSN 03600300. doi: 10.1145/2512962.

Li, B., Zhao, P., Hoi, S. C., and Gopalkrishnan, V. PAMR: Passive aggressive mean reversion strategy for portfolio selection. *Machine Learning*, 87(2):221–258, 2012. ISSN 08856125. doi: 10.1007/s10994-012-5281-z.

Li, J. and Yu, T. Deep Reinforcement Learning based Multi-Objective Integrated Automatic Generation Control for Multiple Continuous Power Disturbances. *National Natural Science Foundation of China*, 2020. doi: 10.1109/ACCESS.2020.3019535.

Li, X. and Peng, Z. Portfolio optimization under the framework of reinforcement learning. *Proceedings - 2019 11th International Conference on Measuring Technology and Mechatronics Automation, ICMTMA 2019*, pages 799–802, 2019. doi: 10.1109/ICMTMA.2019.00180.

Liang, Z., Chen, H., Zhu, J., Jiang, K., and Li, Y. Adversarial Deep Reinforcement Learning in Portfolio Management. 2018.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. Continuous learning control with deep reinforcement. 2016.

Liu, L., Ouyang, W., Wang, X., Fieguth, P., Chen, J., Liu, X., and Pietikäinen, M. Deep Learning for Generic Object Detection: A Survey. *International Journal of Computer Vision*, 128(2):261–318, 2020. ISSN 15731405. doi: 10.1007/s11263-019-01247-4.

MacHalek, D., Quah, T., and Powell, K. M. Dynamic Economic Optimization of a Continuously Stirred Tank Reactor Using Reinforcement Learning. *Proceedings of the American Control Conference*, 2020-July: 2955–2960, 2020. ISSN 07431619. doi: 10.23919/ACC45564.2020.9147706.

Markowitz, H. Portfolio Selection. *The Journal of Finance*, 7(1), 1952.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015. ISSN 14764687. doi: 10.1038/nature14236.

Mnih, V., Badia, A. P., Mirza, L., Graves, A., Harley, T., Lillicrap, T. P., Silver, D., and Kavukcuoglu, K. Asynchronous methods for deep reinforcement learning. *33rd International Conference on Machine Learning, ICML 2016*, 4:2850–2869, 2016.

Nazir, A. Efficient online portfolio selection with heuristic AI algorithm. *Statistics, Optimization and Information Computing*, 7(2):329–347, 2019. ISSN 23105070. doi: 10.19139/soic.v7i2.484.

Niaki, S. T. A. and Hoseinzade, S. Forecasting S&P 500 index using artificial neural networks and design of experiments. *Journal of Industrial Engineering International*, 9(1):1–9, 2013. ISSN 2251712X. doi: 10.1186/2251-712X-9-1.

Noda, K., Yamaguchi, Y., Nakadai, K., Okuno, H. G., and Ogata, T. Audio-visual speech recognition using deep learning. *Applied Intelligence*, 42(4):722–737, 2015. ISSN 15737497. doi: 10.1007/s10489-014-0629-7.

Olah, C. Understanding LSTM Networks, 2015. URL `https://colah.github.io/posts/2015-08-Understanding-LSTMs/`.

Ormos, M. and Urbán, A. Performance analysis of log-optimal portfolio strategies with transaction costs. *Quantitative Finance*, 2013.

Patel, S. S. A Deep Reinforcement Learning Approach to the Portfolio Management Problem [m]. *ProQuest Dissertations and Theses*, page 141, 2018.

Patterson, J. and Gibson, A. *Deep Learning-A Practitioner's Practice*. 2017. ISBN 9781491914250.

Polyak, B. T. and Juditsky, A. Acceleration of Stochastic Approximation by Averaging. *SIAM Journal on Control and Optimization*, 30(4):838–855, 1992. doi: 10.1137/0330046.

Pourchot, A. and Sigaud, O. CEM-RL: Combining Evolutionary and Gradient-based methods for policy search. *conference paper at ICLR 2019*, 2019.

Quantivity. Why Log Returns, 2011. URL `https://quantivity.wordpress.com/2011/02/21/why-log-returns/`.

Ravikumar, S. and Saraf, P. Prediction of stock prices using machine learning (regression, classification) Algorithms. *2020 International Conference for Emerging Technology, INCET 2020*, pages 1–5, 2020. doi: 10.1109/INCET49848.2020.9154061.

Rokach, L. Ensemble Methods for Classifiers. *Data Mining and Knowledge Discovery Handbook*, pages 957–980, 2006. doi: 10.1007/0-387-25465-x_45.

Sato, Y. Model-Free Reinforcement Learning for Financial Portfolios: A Brief Survey. pages 1–20, 2019.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Shah, D. and Isah, H. Stock Market Analysis : A Review and Taxonomy of Prediction Techniques. (ii), 2019.

Sharpe, W. F. The sharpe ratio. *The Journal of Portfolio Management*, 21(1):49–58, 1994.

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M. Deterministic policy gradient algorithms. *31st International Conference on Machine Learning, ICML 2014*, 1:605–619, 2014.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016. ISSN 14764687. doi: 10.1038/nature16961.

Song, Y.-G., Zhou, Y.-L., and Han, R.-J. Neural networks for stock price prediction. 00(00):1–13, 2018.

Sortino, F. A. and Price, L. N. Performance Measurement in a Downside Risk Framework. *The Journal of Investing*, 3(3):59–64, 1994. doi: https://doi.org/10.3905/joi.3.3.59.

Sutskever, I., Vinyals, O., and Le, Q. V. Sequence to sequence learning with neural networks. *Advances in Neural Information Processing Systems*, 2014.

Sutton, R. S. and Barto, A. G. Reinforcement Learning: An Introduction. *Trends in Cognitive Sciences*, 3(9):360, 1999. ISSN 13646613. doi: 10.1016/s1364-6613(99)01331-5.

Thrun, S. and Schwartz, A. Issues in using function approximation for reinforcement learning. *In Proceedings of the 1993 Connectionist Models Summer School Hillsdale, NJ. Lawrence Erlbaum*, 1993.

Trunk, G. V. A Problem of Dimensionality: A Simple Example. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-1(3):306–307, 1979.

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

Van Hasselt, H. Double Q-learning. *Advances in Neural Information Processing Systems*, pages 2613–2621, 2010.

Van Hasselt, H., Guez, A., and Silver, D. Deep reinforcement learning with double q-learning. *AAAI*, pages 2094–2100, 2016.

Vogiatzis, A. Reinforcement Learning for Financial Portfolio Management [m]. (February), 2019.

Weber, A. Theory of the Location of Industries. *Chicago : The University of Chicago Press*, 1929.

Wiering, M. A. and van Hasselt, H. Ensemble algorithms in reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 38(4):930–936, 2008. ISSN 10834419. doi: 10.1109/TSMCB.2008.920231.

Yang, H., Liu, X.-Y., Zhong, S., and Walid, A. Deep Reinforcement Learning for Automated Stock Trading: An Ensemble Strategy. *SSRN Electronic Journal*, 2020. doi: 10.2139/ssrn.3690996.

Ye, Y., Pei, H., Wang, B., Chen, P.-Y., Zhu, Y., Xiao, J., and Li, B. Reinforcement-Learning Based Portfolio Management with Augmented Asset Movement Prediction States. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(01):1112–1119, 2020. ISSN 2159-5399. doi: 10.1609/aaai.v34i01.5462.

Zhang, Y., Zhao, P., Li, B., Wu, Q., Huang, J., and Tan, M. Cost-Sensitive Portfolio Selection via Deep Reinforcement Learning. *IEEE Transactions on Knowledge and Data Engineering*, 4347(c):1–1, 2020. ISSN 1041-4347. doi: 10.1109/tkde.2020.2979700.