Eidgenössische Technische Hochschule Zürich

National University of Singapore - Asian Institute of Digital Finance

Master Thesis

Department of Computer Science

Portfolio Optimization using Deep Deterministic Policy Gradient and LSTM in Reinforcement Learning

Author
Auguste Lefevre

ID Number
16-820-078

Professors
Prof. Ying Chen, NUS
Prof. Thomas Hofmann, ETHZ

Role
External Main Supervisor
Supervisor

Singapore, NUS, August 2023

Contents

List of Figures

List of Tables

Acronyms

AI Artificial Intelligence

AIDF Asian Institute of Digital Finance

BCRP Best Constant Rebalanced Portfolio

BDDPG Basic DDPG Framework Experiment

CNN Convolutional Neural Network

CNNs Convolutional Neural Networks

CRNN Convolutional Recurrent Neural Networks

DATA-ENH Enhanced State Space with Data Inputs Experiment

DDPG Deep Deterministic Policy Gradient

DL Deep Learning

DNN-AC Deep Neural Network for Actor-Critic Networks Experiment

DNNs Deep Neural Networks

DPG Deep Policy Gradient

DQN Deep Q-Network

DRL Deep Reinforcement Learning

ETHZ Eidgenössische Technische Hochschule Zürich

GRU Gated Recurrent Unit

LSTM Long-Short-Term-Memory

LSTMs Long-Short-Term-Memory

MDP Markov Decision Processes

ML Machine Learning

MPT Modern Portfolio Theory

NN Neural Networks

NUS National University of Singapore

OHLC Open High Low and Close Prices

OLMAR On-line Moving Average Reversion

OLPS On-line Portfolio Selection

PO Portfolio Optimization

PPO Proximal Policy Optimization

RL Reinforcement Learning

RNN Recurrent Neural Network

RNNs Recurrent Neural Networks

SR Sharpe Ratio

SVMs Support Vector Machines

SYNERGY Synergistic DNN Architecture and Enhanced State Space Experiment

TD3 Twin Delayed Deep Deterministic Policy Gradient

TRPO rust Region Policy Optimization

UP Universal portfolio

W11 Window Length 11

W3 Window Length 3

W5 Window Length 5

W7 Window Length 7

W9 Window Length 9

1 Acknowledgements

I would like to express my heartfelt gratitude to Prof. Thomas Hoffman, my supervisor from Eidgenössische Technische Hochschule Zürich (ETHZ), for providing me with the opportunity to pursue my master thesis abroad at the Asian Institute of Digital Finance (AIDF), National University of Singapore (NUS). His support and guidance have been instrumental in shaping my research journey.

I am also grateful to my primary supervisor, Prof. Ying Chen, for her invaluable guidance, feedback, and support throughout my research work at AIDF. Her expertise in the field has been invaluable in shaping my research work. Her support, feedback, and constructive criticism have been instrumental in helping me achieve my goals.

I would like to extend my sincere appreciation to the faculty members and staff at AIDF – who provided me with an inspiring and intellectually stimulating environment – for their assistance and support throughout my research.

Finally, I would like to express my heartfelt gratitude to my family and friends for their unwavering support, encouragement, and love. Their constant support and encouragement have been a source of strength and inspiration throughout my academic journey.

Thank you all for your invaluable contributions to my research work, personal growth, and academic success.

2 Abstract

The financial world, integral to our global economic structure, has seen rapid evolution bolstered by advances in technology, including electronic trading platforms, algorithmic trading, and artificial intelligence. Portfolio Optimization, aiming to balance maximizing returns with risk mitigation, remains a pressing challenge.

This study explores the potential of Deep Reinforcement Learning, specifically the Deep Deterministic Policy Gradient (DDPG) methodology, for Portfolio Optimization. We investigated various neural network architectures for approximating value and policy functions and analyzed the influence of different data types, from simple closing prices to complex Technical Indicators, on the state space of Reinforcement Learning.

Our experiments revealed that the integration of the Long-Short-Term-Memory (LSTM) model with the DDPG framework, enhanced with Technical Indicators, offers a potent solution for portfolio optimization. Results from traditional Convolutional Neural Network (CNN) models were intermediate. In contrast, while Convolutional Recurrent Neural Network (CRNN) models displayed potential in theory, they did not match LSTM's and CNN's effectiveness. Notably, augmenting the state space with Open High Low Close (OHLC) data and Technical Indicators proved beneficial across nearly all tested configurations, underscoring their importance in this field.

3 Introduction

The financial system plays a pivotal role in the global economy, governing resource allocation and risk management. Throughout history, advancements in technology have significantly influenced the structure and operations of the financial system. Developments like electronic trading platforms, algorithmic trading, and artificial intelligence have accelerated trading processes, improved efficiency, and facilitated novel investment strategies.

Modern financial markets are intricate, and traders employ sophisticated strategies for competitive advantages. Fields such as asset pricing and financial forecasting have been rigorously studied. However, this thesis focuses on another field of research:Portfolio Optimization (PO).

The central challenge of PO is constructing a set of assets that maximize returns while minimizing associated risks. Portfolio managers aim to effectively allocate funds across various asset classes like stocks, commodities, and bonds to strike a balance between potential returns and risks. Diversification has historically been seen as a method to optimize returns for each unit of risk, leading to questions about how best to select asset combinations and allocate portfolio weights in alignment with an investor's objectives.

Conventional mathematical approaches to these challenges often arise from stochastic controls and processes. However, finding the right balance between simplicity and precision in models can be challenging. While Machine Learning (ML) and Deep Deep Learning (DL) offer some solutions, their need for supervised data sometimes restricts their utility. Herein, Reinforcement Learning (RL), a subset of Artificial Intelligence (AI), offers a compelling alternative due to its ability to learn from interactions without requiring labeled data.

With this context, we propose the use of Deep Reinforcement Learning (DRL) for PO, specifically focusing on the Deep Deterministic Policy Gradient (DDPG) method. Our primary objective is to develop a *Deep Reinforcement Learning Framework for Portfolio Optimization* using DDPG. This work will emphasize the spatial dynamics of RL procedures and the intricacies of DDPG. We also aim to investigate how Neural Networks (NN) architectures influence value and policy function approximations. Our research approach will start by using closing prices as the primary data input. We will then enhance this model by incorporating Open High Low and Close Prices (OHLC) data and subsequently by integrating technical indicators.

This thesis will provide the foundational scientific concepts, explain necessary financial knowledge, and dive deep into RL techniques. We will offer a literature review focusing on the application of DL in PO, aiming to identify gaps in the existing body of research. This discussion will be followed by a comprehensive presentation of our methodology, experiments, and findings. We will conclude by reflecting on our results, their implications, and potential future avenues for exploration.

In the course of our research, we unearthed two pivotal insights that stand to reshape the application of DRL in Portfolio Optimization. Firstly, when it comes to actor-critic networks, the LSTM architecture consistently surpassed its peers in value and policy function approximation. Secondly, we observed a steady uptrend in performance outcomes when we expanded our data to incorporate OHLC and subsequently, Technical Indicators. These critical discoveries, illuminating the importance of architectural choice and data enhancement, have far-reaching implications on practical investment strategies. As you delve deeper into this thesis, you'll uncover the nuances of these findings, which we hope will inspire a keen interest in their broader implications and future applications.

3.1 Motivation and Objectives

Firstly, DRL has emerged as a prominent field of research in recent years, demonstrating promising results in various scientific domains. Although finance has only recently been exposed to RL techniques, and DRL even more so, it presents a fertile ground for exploration. Given the expansion of this field, there are numerous avenues to explore. For instance, one could study different RL and DRL algorithms using OHLC data [8]. Alternatively, researchers have attempted to combine algorithms using ensemble methods [9], while others have explored more intricate approaches involving Multi-Agent RL [10]. It is crucial to note that DRL for finance applications remains a relatively scarcely studied area, leaving ample room for contributions. Consequently, we have chosen to work in this domain, aiming to develop technical expertise in DRL and contribute to the emerging field of DRL for portfolio optimization and gain knowledge for future research in this area.

Our objectives can be summarized as:

1. Develop a framework to investigate the PO problem using DDPG and contrast its performance with algorithms grounded in financial techniques like On-Line Portfolio Selection. *Importance:* Traditional financial strategies, such as On-Line Portfolio Selection, have been foundational in the field of portfolio management. Comparing the DDPG's performance against such a well-regarded method offers a clear benchmark. This allows us to evaluate the potential advantages of integrating deep reinforcement learning into portfolio management. In essence, this objective will highlight whether modern DRL techniques can present tangible improvements over established strategies.

2. Examine how different NN architectures, including CNN, CRNN, and LSTM, affect the efficiency of our DDPG-based models. *Importance:* The choice of a neural network architecture significantly impacts its ability to learn and predict. By investigating various architectures, we aim to pinpoint the most suitable one for the intricate task of Portfolio Optimization using DDPG. This exploration ensures that future research endeavors are focused on architectures that hold the most promise for this application.

3. Understand the influence of different data inputs, starting with closing prices, followed by an enhancement with OHLC, and finally, with the addition of technical indicators. *Importance:* The quality and variety of data inputs shape the effectiveness of machine learning models. By progressively augmenting our data sources, from closing prices to OHLC and technical indicators, we aim to elucidate the relative value each data type brings to the model's performance. This underscores the notion that the quality of input data can be as influential as the complexity of the model itself.

Through achieving these objectives, we aim to expand both the scientific and practical understanding of DDPG's role in portfolio optimization.

4 Scientific Background

In the Scientific Background chapter of this dissertation, we'll be exploring the intricate mechanics of RL, alongside a succinct introduction to essential financial tenets vital to our study's context. We'll initiate our discourse by addressing the portfolio optimization puzzle. This exploration traces its origins, reflecting on conventional strategies like quadratic programming in tackling the Markowitz problem. Alongside, we'll discuss the limitations inherent in these age-old techniques and explore the modern intricacies birthed by the ever-evolving financial milieu. A succinct discussion on the revolutionary advancements in ML and DL domains, which have caused paradigm shifts in portfolio optimization, will ensue. Subsequently, we will delve into the domain of Financial Engineering. This section encapsulates a gamut of terminologies, concepts, the intricacies of financial time series, and prevalent evaluation standards within this ambit. Rounding off, we aim to provide a comprehensive breakdown of RL and DRL, probing into their potential role within portfolio optimization. By the chapter's culmination, readers should find themselves well-versed with both the financial frameworks and DRL methodologies pivotal to our investigation.

4.1 Portfolio Optimization

4.1.1 Problem Description

Portfolio optimization, at its core, revolves around the fundamental concept of risk and reward trade-off. It represents a quantitative decision-making process, which seeks to allocate investment among a range of assets in order to optimize a specific objective. Generally, the ultimate goal of portfolio optimization is to construct a portfolio of investments that delivers the highest possible return for a given level of risk or minimizes risk given a level of return the investor which to reach. This concept is deeply rooted in the field of financial economics, with implications spanning risk management, asset allocation, and financial engineering.

In the subsequent sections, we will present a concise overview of historical approaches that have been devised to address the challenges inherent in portfolio optimization. Our primary focus will center on the seminal Markowitz Model, which laid the foundation for modern portfolio theory, and subsequently explore more contemporary solutions harnessing the potential of ML and DL. Within each methodology, we will delve into the intricacies of their problem-solving strategies, as well as analyze the advantages and limitations associated with these techniques.

4.1.2 Markowtiz Model - Modern Portfolio Theory

The Markowitz Model, also known as Modern Portfolio Theory (MPT), is a mathematical model formulated by Harry Markowitz in 1952 [11, 12]. This model presents a solution to the portfolio optimization problem and aims to maximize the expected return for a given level of risk. Alternatively, it can also be used to minimize the risk for a given level of expected return. Markowitz's pioneering work on portfolio theory is widely recognized as a major foundation of modern financial economics. It has shaped investment management practices and influenced various financial models that are based on the trade-off between risk and return.

The Markowitz model, ie. Mean-Variance model, is a formalization and generalization of diversification in investing, emphasizing the importance of portfolios, risk, and the correlations between securities. The main theory states that by combining different assets whose returns are not perfectly positively correlated, investors can potentially construct a portfolio that offers superior risk-return trade-offs [13, 14].

4.1.2.1 Mathematical Formulation of the Markowitz Model

In this section we will see how to mathematically solve the mean-variance optimization [15, 16, 17]. Consider an investment universe consisting of M assets. Each asset i, where $i = 1, 2, \ldots, M$, has a random return, denoted by R_i. We define:

1. **Weights** ($\mathbf{w}$): A vector of weights, $\mathbf{w} = [w_1, w_2, \ldots, w_M]$, where w_i is the proportion of the total investment invested in asset i.
2. **Expected Returns** (μ): A vector of expected returns for each asset, $\mu = [\mu_1, \mu_2, \ldots, \mu_M]$, where μ_i is the expected return of asset i.
3. **Covariance Matrix** (Σ): An $M \times M$ covariance matrix, where the element Σ_{ij} represents the covariance between the returns of asset i and asset j.

Using elementary statistics deinition, the expected portfolio return and the portfolio variance are then defined as follows:

1. **Expected portfolio return** (μ_p): $\mu_p = \mathbf{w}^T \mu$
2. **Portfolio variance** (σ_p^2): $\sigma_p^2 = \mathbf{w}^T \Sigma \mathbf{w}$

Given this formulation, the Mean-Variance Optimization problem can be defined as follows:

$$\begin{array}{ll} \text{Minimize} & \sigma_p^2 = \mathbf{w}^T \Sigma \mathbf{w} \\ \text{Subject to} & \mu_p = \mathbf{w}^T \mu \geq \mu^* \\ & \sum w_i = 1 \\ & w_i \geq 0 \text{ for all } i \end{array} \tag{1}$$

Here, μ^* is the minimum level of expected return. The optimization problem aims to find the weights that minimize the portfolio variance, subject to the constraints that the expected portfolio return is at least as large as μ^* and the weights sum to one and are non-negative.

To solve the optimization problem, it can be reformulated as a Lagrangian function, which allows us to find optimal points even in the presence of constraints. We introduce Lagrange multipliers λ_1 and λ_2 corresponding to the two constraints. The Lagrangian L for the problem is:

$$L(\mathbf{w}, \lambda_1, \lambda_2) = \mathbf{w}^T \Sigma \mathbf{w} - \lambda_1 (\mathbf{w}^T \mu - \mu^*) - \lambda_2 (\mathbf{w}^T \mathbf{e} - 1) \tag{2}$$

where $\mathbf{e}$ is a column vector of ones. The first term is the portfolio variance, the second term represents the expected return constraint, and the third term represents the weights summing to one constraint. The optimal solution satisfies the Karush-Kuhn-Tucker (KKT) conditions. Thus, we need to find the $\mathbf{w}$, λ_1, and λ_2 that satisfy the following equations derived from setting the gradient of L to zero:

$$\begin{aligned} \frac{\partial L}{\partial \mathbf{w}} &= 2\Sigma \mathbf{w} - \lambda_1 \mu - \lambda_2 \mathbf{e} = 0, \\ \frac{\partial L}{\partial \lambda_1} &= \mathbf{w}^T \mu - \mu^* = 0, \\ \frac{\partial L}{\partial \lambda_2} &= \mathbf{w}^T \mathbf{e} - 1 = 0. \end{aligned} \tag{3}$$

Solving these equations will yield the weights that minimize portfolio variance while meeting the desired level of return and the constraints on the weights. From the first KKT condition, we can express the optimal weights $\mathbf{w}^*$ in terms of the Lagrange multipliers λ_1 and λ_2:

$$\mathbf{w}^* = \frac{1}{2}\Sigma^{-1}(\lambda_1\mu + \lambda_2\mathbf{e}) \tag{4}$$

Substituting $\mathbf{w}^*$ into the other two KKT conditions gives two equations:

$$\begin{aligned} \frac{1}{2}\lambda_1\mu^T\Sigma^{-1}\mu + \frac{1}{2}\lambda_2\mu^T\Sigma^{-1}\mathbf{e} &= \mu^* \\ \frac{1}{2}\lambda_1\mathbf{e}^T\Sigma^{-1}\mu + \frac{1}{2}\lambda_2\mathbf{e}^T\Sigma^{-1}\mathbf{e} &= 1 \end{aligned} \tag{5}$$

We can write these equations in matrix form:

$$\begin{pmatrix} \mu^T\Sigma^{-1}\mu & \mu^T\Sigma^{-1}\mathbf{e} \\ \mathbf{e}^T\Sigma^{-1}\mu & \mathbf{e}^T\Sigma^{-1}\mathbf{e} \end{pmatrix} \begin{pmatrix} \lambda_1 \\ \lambda_2 \end{pmatrix} = 2 \begin{pmatrix} \mu^* \\ 1 \end{pmatrix} \tag{6}$$

We can solve this equation for λ_1 and λ_2 and substitute these back into the equation for $\mathbf{w}^*$ to get the optimal weights. Remember that the weights sum to one, which may not be the case if short sales are allowed (some weights could be negative), so in this case a scaling step would be necessary to ensure the weights sum to one.

4.1.2.2 The Efficient Frontier in the Markowitz Model

A key construct in the Modern Portfolio Theory, initially proposed by Harry Markowitz, is the Efficient Frontier. This term refers to the set of portfolios that offer the maximum possible expected return for a given level of risk or, equivalently, the minimum possible risk for a specified level of expected return. Thus, the Efficient Frontier demarcates the boundary of feasible portfolios in the space defined by expected return and portfolio risk.

The Efficient Frontier is derived from the solutions to the mean-variance portfolio optimization problem, as previously described, for various target levels of expected return μ^*. Each point on the Efficient Frontier represents an optimal solution to this optimization problem, i.e., an optimal set of portfolio weights $\mathbf{w}^*$, which yields the maximum expected return for a given level of risk or the minimum risk for a given expected return.Formally, a portfolio lies on the Efficient Frontier if and only if there does not exist another portfolio with a higher expected return but the same risk, or a lower risk but the same expected return. Therefore, portfolios that lie on the Efficient Frontier are deemed 'efficient' in the sense that no other portfolio offers a more favorable risk-return tradeoff.

In conclusion, the Efficient Frontier offers a powerful conceptual tool to guide the process of portfolio construction. By optimizing the tradeoff between risk and return, it helps investors to make more informed and rational investment decisions. However, it's important to bear in mind that the actual computation of the Efficient Frontier relies on the assumptions of the Markowitz model, including the accurate estimation of expected returns and covariances, which may not always hold in practice.

4.1.2.3 Limitation of the Markowitz Model

The Markowitz model, despite its significant contributions to modern portfolio theory, presents several limitations that may hinder its effectiveness under certain conditions. In the following, we critically examine some of the key limitations of the Markowitz model:

1. **Assumption of Normal Distributions:** The model presumes that the returns of the assets are normally distributed. Nonetheless, real-world financial markets often display

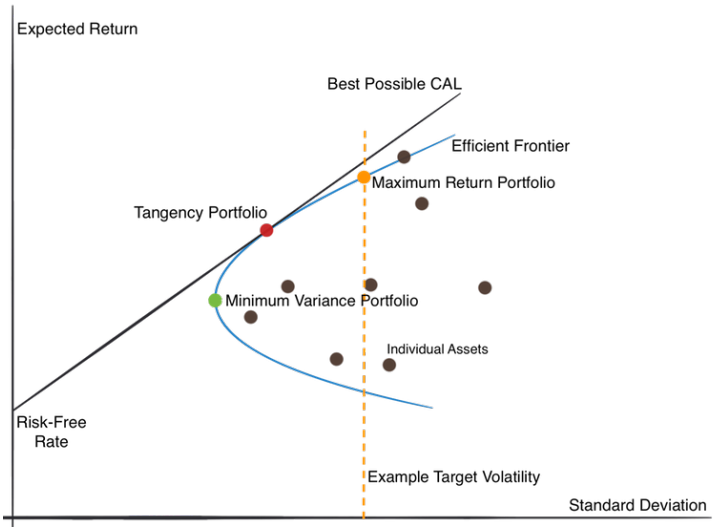

Figure 2: Illustration of the Efficient Frontier in the risk-return space.

skewness and kurtosis, where asset returns exhibit asymmetry and heavy tails respectively, violating the assumption of normality. This can lead to erroneous predictions and risk assessments. DRL models have the potential to learn from historical data without assuming normal distributions, thus offering a potential solution for managing complex and irregular patterns prevalent in financial markets.

2. **Static Model:** Another inherent drawback is that the Markowitz model is static. It does not adapt to evolving market conditions but determines the optimal portfolio based on a single period's historical data. In contrast, DRL can accommodate dynamic adjustments to the portfolio weights over time, offering the capability to learn and adapt to changes in the market, presenting a more realistic and practical approach.

3. **Estimation Error:** Markowitz model is highly sensitive to input estimations, especially those of expected returns, variances, and covariances. Minor changes in these estimates can result in drastically different optimal portfolios, an issue commonly known as the "error maximization problem". DRL models, particularly those based on policy gradients, have the potential to be more robust to estimation errors, as they aim to directly learn the optimal policy that maximizes the expected return, rather than attempting to estimate individual asset returns and variances.

In conclusion, the Markowitz model, despite its pioneering role in portfolio optimization, carries limitations that may be potentially addressed by more advanced techniques such as DRL. By learning directly from historical data and exhibiting the ability to adapt over time, DRL provides a dynamic, flexible, and potentially superior approach to portfolio optimization that may yield improved performance under real-world market conditions.

4.1.3 Modern Solutions: Machine and Deep Learning

The recent proliferation of big data and the exponential increase in computational power have catalyzed the application of ML and DL methods in finance [18]. These advances have opened new doors for sophisticated data-driven models to capture intricate patterns and non-linear relationships in financial markets, leading to improved forecasting and decision-making capabilities [19]. ML and DL offer alternative methods to address the complexities inherent in portfolio optimization problems, which classical approaches like the Markowitz model may struggle to handle effectively [20]. Let's see how.

First, the robustness of ML and DL methods makes them particularly suited to handling non-linear relationships between variables, high dimensionality, and non-normal distributions. In contrast to the assumptions required by traditional methods, these techniques can learn complex mappings between inputs and outputs from the data itself, making fewer assumptions about the underlying data distribution.

Second, machine learning and deep learning can deal more effectively with the issue of estimation error, a significant limitation of the Markowitz model. By learning directly from the data, these models can reduce reliance on specific, potentially inaccurate estimates of expected returns, variances, and covariances. Instead, the learning process focuses on optimizing a specific performance measure, such as the Sharpe ratio or portfolio return, thereby directly aligning the learning process with the investment objectives [21].

Finally, while the Markowitz model is essentially static, machine learning and deep learning models can be inherently dynamic. They can adapt to new information as it becomes available and can therefore respond more effectively to changing market conditions. This dynamic capability may be particularly useful in financial markets, which are characterized by frequent shifts in patterns and trends.

In the subsequent subsections, we will delve into the methodologies rooted in ML and DL. Crucially, we will elucidate key concepts in ML and DL that serve as foundational knowledge for comprehending DRL.

4.1.3.1 Relevant Machine and Deep Learning Methods

A wide range of ML and DL methods as been used to solve financial problem. Here we will focus on Support Vector Machines (SVMs) and Deep Neural Networks (DNNs) as they are the more relevant to our project, especially the DNNs that are a central piece of DRL

1. **Support Vector Machines**: SVMs represent one of the machine learning techniques that have been employed for portfolio optimization. SVMs are supervised learning models typically used for classification and regression analysis. They are particularly noted for their ability to handle high dimensional data, making them suitable for financial applications where numerous factors may influence investment returns [22]. In the context of portfolio optimization, SVMs have been utilized to predict the future prices of financial assets, which then serve as inputs for portfolio allocation decisions [23]. These predictive models can capture non-linear relationships between a multitude of factors, overcoming the Markowitz model's limitation related to assuming linear relationships and normal distributions of returns. Further, SVMs can be considered more robust to estimation errors compared to the Markowitz model. By focusing on the construction of hyperplanes that separate classes in the data (for classification tasks) or that fit the data points (for regression tasks), SVMs can be less sensitive to slight changes in input values, thus mitigating the 'error maximization' problem [22].

   Given a training dataset of instance-label pairs $(\mathbf{x_i}, y_i)$, $i = 1, ..., n$, where $\mathbf{x_i} \in \mathbb{R}^d$ and $y_i \in -1, 1$ for classification tasks (or $y_i \in \mathbb{R}$ for regression tasks), the SVMs aims to find a function $f(\mathbf{x}) = \mathbf{w}^\top \phi(\mathbf{x}) + b$, where $\mathbf{w} \in \mathbb{R}^d$, $b \in \mathbb{R}$, and $\phi(\cdot)$ is a feature mapping function. The parameters $\mathbf{w}$ and b are determined by solving the following optimization problem [23]:

$$\begin{aligned} \min_{\mathbf{w},b,\xi} \quad & \frac{1}{2}|\mathbf{w}|^2 + C\sum_{i=1}^{n}\xi_i \\ \text{s.t.} \quad & y_i(\mathbf{w}^\top\phi(\mathbf{x_i}) + b) \geq 1 - \xi_i, \quad i = 1, ..., n \\ & \xi_i \geq 0, \quad i = 1, ..., n, \end{aligned} \tag{7}$$

where ξ_i are slack variables that allow for misclassifications (for classification tasks) or errors (for regression tasks), and $C > 0$ is a regularization parameter controlling the trade-off between margin maximization and error minimization.

In the context of portfolio optimization, $\mathbf{x_i}$ might represent a set of features describing the state of the market or a particular asset at a certain time, and y_i might represent the direction of price movement (for classification tasks) or the actual price (for regression tasks) of a particular asset at the next time step. The SVM, by using the kernel trick, can capture non-linear relationships in the data. Instead of directly computing the feature mapping function $\phi(\cdot)$, we use a kernel function $K(\mathbf{x_i}, \mathbf{x_j}) = \phi(\mathbf{x_i})^\top\phi(\mathbf{x_j})$, which enables efficient computation even in high-dimensional feature spaces. The choice of the kernel function (linear, polynomial, radial basis function, etc.) and the tuning of the hyperparameters (e.g., the degree of the polynomial kernel, the width of the radial basis function kernel, and the cost parameter C) have a great impact on the performance of the SVMs. They are usually selected through a process of cross-validation or grid search to find the values that yield the best predictive performance. Given the predicted future prices of the assets, we can then use these predictions as inputs for making portfolio allocation decisions, effectively integrating the predictive power of SVMs into the portfolio optimization process.

However, despite the potential advantages, it's crucial to note the SVM model's sensitivity to its hyperparameters. The selection of an appropriate kernel and the tuning of the cost parameter C and the kernel parameters could greatly impact the model's performance. Techniques such as cross-validation and grid search are typically employed for this hyperparameter optimization process.

2. **Deep Neural Networks:** DNNs have recently garnered significant attention in the finance domain, particularly within the sphere of portfolio optimization. Their popularity can be attributed to their superior ability to model intricate non-linear relationships, efficiently manage high-dimensional data, and dynamically adapt to fresh information over time [24]. DNNs are a specialized form of deep learning models that employ numerous layers between the input and output layers. This allows the DNN to learn data representations effectively and map inputs to outputs, thereby facilitating nuanced financial decision-making processes.

   From a mathematical perspective, a DNN can be described as a function f mapping an input vector x to an output vector y through multiple layers of transformations. A single layer can be represented mathematically as $h_i = \sigma(W_i h_{i-1} + b_i)$, where W_i and b_i are the weights and biases of the i-th layer, σ is a non-linear activation function (such as a ReLU, sigmoid, or tanh function), and h_i is the output of the i-th layer (with $h_0 = x$). The DNN output is then $y = f(x) = h_n$, where n is the number of layers.In the portfolio optimization context, DNNs have been widely utilized in various capacities, such as predicting asset prices, forecasting volatility, estimating correlations, and even determining optimal portfolio weights directly [25]. One approach might involve employing a DNN to model and predict each asset's future returns. Subsequently, these anticipated returns could be utilized as inputs for portfolio allocation decisions.

Deep learning offers a variety of architectures that can be selected depending on the requirements of the problem at hand. In portfolio optimization, three architectures, namely, and Recurrent Neural Networks (RNNs), Long-Short-Term-Memory (LSTM), Convolutional Neural Networks (CNNs), have been frequently used due to their respective advantages.

- **Recurrent Neural Networks**: RNNs have cyclic connections making them powerful models for time-series data, such as financial data. This architecture has the capacity to use its internal memory to process sequences of inputs, which makes it perfect for tasks where past information is crucial to compute the output [26]. The base RNN can be expressed mathematically as follows:

$$h_t = \tanh(W_{hh}h_{(t-1)} + W_{xh}x_t + b_h) \tag{8}$$

  where h_t is the hidden state at time t, W_{hh} is the weight matrix for the hidden layer, W_{xh} is the weight matrix for the input layer, x_t is the input at time t, and b_h is the bias.

- **Long Short-Term Memory**: This architecture is a specialized type of RNNs designed to prevent the vanishing and exploding gradient problem, allowing the model to capture long-term dependencies effectively. In the LSTM architecture , a series of repeating modules known as memory cells replace the traditional neurons in the hidden layer. Each memory cell contains three crucial elements - input gate, forget gate, and output gate, allowing the model to regulate the flow of information [27]:

$$\begin{aligned}
i_t &= \sigma(W_{ii}x_t + b_{ii} + W_{hi}h_{(t-1)} + b_{hi}),\\
f_t &= \sigma(W_{if}x_t + b_{if} + W_{hf}h_{(t-1)} + b_{hf}),\\
g_t &= \tanh(W_{ig}x_t + b_{ig} + W_{hg}h_{(t-1)} + b_{hg}),\\
o_t &= \sigma(W_{io}x_t + b_{io} + W_{ho}h_{(t-1)} + b_{ho}),\\
c_t &= f_t * c_{(t-1)} + i_t * g_t,\\
h_t &= o_t * \tanh(c_t),
\end{aligned} \tag{9}$$

  where i_t, f_t, g_t, o_t, c_t, and h_t denote the input gate, forget gate, cell input, output gate, cell state, and hidden state, respectively, at time t.

- **Convolutional Neural Networks**: CNNs are particularly suited for grid-like data, including images and time-series data, where spatial or temporal relationships exist. A key feature of CNNs is their convolutional layers that apply a set of filters across the input data, preserving spatial relationships and reducing the number of parameters [28].

$$x_{kj} = f\left(\sum_{i=1}^{M} w_i x_{(k+i-1),j} + b\right) \tag{10}$$

  Here, f is the activation function, w are the weights, b is the bias, x represents the inputs and outputs, and M is the number of connections to the previous layer.

DNNs are noted for their ability to learn hierarchical representations, enabling them to decipher complex patterns in financial data, which may be overlooked by simpler models. Such an ability can potentially facilitate improved predictions and thus, more effective portfolio allocation decisions [24].Moreover, DNNs efficiently tackle the dynamic nature of financial markets. By continually updating the weights within the network with new data, DNNs can seamlessly adapt to fluctuating market conditions over time. This offers a distinct advantage over the static nature of traditional methods like the Markowitz model, offering a dynamic portfolio optimization approach.

However, it is crucial to acknowledge that while DNNs present potential advantages, they also pose unique challenges. One primary concern is their 'black box' nature, making their predictions notoriously difficult to interpret. In a field like finance, interpretability can often be as important as predictive accuracy, thus the opacity of DNNs might pose a significant hurdle [29]. Additionally, DNNs require vast amounts of data for training, and risks of overfitting may arise if not properly regularized [4]. In summary, while DNNs, similar to SVMs, offer promising solutions to some of the limitations inherent in the Markowitz model, their effective application to portfolio optimization requires careful consideration of their unique challenges and potential pitfalls. Note that we will describe in details our implementation and the logic behind each network in the Methods & Experiments section of this report.

4.1.3.2 Limitation of Machine and Deep Learning Methods

While ML methods such as SVMs and DL techniques such as DNNs present sophisticated ways to approach the portfolio optimization problem, there are notable limitations

- **Static Decision-Making Process**: ML and DL models such as SVMs and DNNs typically make predictions in a static manner, with the models being trained on a historical dataset and used to predict future outcomes. While DNNs can adapt to new data to some extent, they do not explicitly consider the sequential nature of decision-making in portfolio optimization. Investment decisions in portfolio management are inherently sequential and dynamic, with each decision impacting future investment opportunities and portfolio returns.

- **Lack of Exploration Mechanism**: ML and DL models, in their standard form, lack an explicit mechanism for exploration, i.e., for balancing the trade-off between exploiting current knowledge and exploring new potential strategies. This is an essential aspect of portfolio optimization, as markets can change in unexpected ways, and sticking to a single strategy can lead to sub-optimal performance.

- **Single-step Ahead Prediction**: While ML and DL models are powerful tools for prediction tasks, they generally focus on single-step ahead prediction when used in a forecasting setting. In contrast, in portfolio optimization, decisions made today can have impacts that unfold over multiple future time periods.

4.1.4 Modern Complexities in Portfolio Optimization

Portfolio optimization, a cornerstone of modern financial theory, has evolved to address an array of complexities that arise in real-world scenarios. These complexities challenge traditional optimization models, necessitating the exploration of advanced techniques like DRL to tackle them effectively.

1. **Market Frictions:** Financial markets, in reality, are riddled with non-idealities such as transaction costs, market impact, and liquidity constraints. These frictions introduce non-linearities into the optimization problem, making it more intricate than classical models suggest [30]. DRL, with its ability to learn and adapt from interactions with the environment, can model and optimize in the presence of these non-linearities, potentially leading to more realistic and efficient trading strategies.

2. **Integer Variables:** Portfolio decisions often involve discrete choices, like determining the number of shares to transact. This introduces integer variables into the optimization problem, leading to mixed-integer optimization challenges [31]. DRL algorithms, especially

those designed for discrete action spaces, can naturally handle such integer decisions, optimizing actions in a high-dimensional and mixed-variable space.

3. **Dynamic Market Conditions:** The financial market's inherent non-stationarity and stochastic nature demand models that can adapt in real-time [32]. DRL excels in environments that are dynamic and uncertain. By continuously learning from new market data, DRL models can adapt their strategies to changing market conditions, offering a more robust solution than static models.

4. **Non-Stationarity of Financial Markets:** Financial markets are characterized by their non-stationary nature, where the statistical properties can evolve over time [33]. DRL, with its emphasis on exploration and exploitation, can adapt to these changing dynamics, ensuring that the trading strategy remains relevant even as market properties shift.

5. **Impact of Global Events:** Global incidents, be it geopolitical events, pandemics, or significant policy shifts, can induce abrupt and unpredictable market changes [34]. Traditional models might struggle to adapt swiftly, but DRL, with its ability to learn from new experiences rapidly, can potentially adjust its strategy in real-time, reacting effectively to unforeseen market shocks.

As evident from the observation, modern techniques and even traditional approaches like the Markowitz model exhibit certain limitations that can be addressed through the utilization of RL and, specifically, DRL. In the subsequent subsection, we will delve into the scientific background of DRL to establish the foundation for our work.

4.2 Financial Engineering

This research looks at how financial engineering, signal processing, and control theory can work together in portfolio optimization using DRL methods. In finance, working with time-series data (data points in time order) is essential. Signal processing gives us tools to better understand and predict this data. Using these tools, we can work on solving real-world finance and business problems.Control Theory deals with systems that change over time, and it's a key idea in Reinforcement Learning. In this study, we applied these methods to decide how to distribute assets in a portfolio. When we combine signal processing, systems, and control theory with finance, we're diving into what's called Financial Engineering.Our research mainly looks at PO and how DRL methods can help. In the following parts, we'll explain key ideas in financial engineering, talk about time-series data and its uses, and discuss how we measure success in this field. With this foundation, we'll then see how we can use DRL to solve the PO problem.

4.2.1 Terms and Concepts

To effectively convey our task, we introduce basic financial terms pertinent to our study.

- **Asset**: A financial asset, such as cash or stocks, represents economic value. Unlike tangible assets like land, its value derives from market dynamics and related risks [35].

- **Portfolio**: Constituted of multiple financial assets, it's defined by its components and vector representation. The representation is given by:

$$\boldsymbol{w}_t = \left[\; w_{1,t}, \quad w_{2,t}, \quad \ldots, \quad w_{M,t} \; \right]^T \in \mathbb{R}^M \quad \text{with} \quad \sum_{i=1}^{M} w_{i,t} = 1$$

  This conceptualization allows portfolios to be assessed like individual assets.

- **Technical Indicator**: "A technical indicator is a mathematical pattern derived from historical data used by technical traders or investors to predict future price trends and make trading decisions. It uses a mathematical formula to derive a series of data points from past price, volume, and open interest data." [36].

4.2.2 Financial Time-Series

Financial markets are fundamentally time-series data due to the continuous evolution of prices in response to changing supply and demand. This section presents how asset prices, the core of financial instruments, are used to construct time series. These series, in turn, give insights into market dynamics, aiding decision-making and understanding of the financial realm.

4.2.2.1 Price-based

A financial time series is often symbolized by the price variable, usually represented as $p_t \in \mathbb{R}$, and can be seen as an univariate time-series: $p_1, p_2, ..., p_T$. For clarity, prices of individual assets over time are denoted as:

$$\overrightarrow{p}_{i,1:T} = \begin{bmatrix} p_{i,1} \\ p_{i,2} \\ \vdots \\ p_{i,T} \end{bmatrix} \in \mathbb{R}_+^T \tag{11}$$

Furthermore, the price vector of multiple assets at time t is:

$$p_t = \begin{bmatrix} p_{1,t}, & p_{2,t}, & \dots, & p_{M,t} \end{bmatrix} \in \mathbb{R}_+^M \tag{12}$$

Combining these, the price matrix of M assets over time window T is:

$$\overrightarrow{P}_{1:T} = \begin{bmatrix} \overrightarrow{p}_{1,1:T}, & \overrightarrow{p}_{2,1:T}, & \cdots, & \overrightarrow{p}_{M,1:T} \end{bmatrix} \in \mathbb{R}_+^{T \times M} \tag{13}$$

However, merely considering price is limiting. OHLC data—Open, High, Low, Close prices, along with trading volume—offers a detailed snapshot of asset price dynamics. The extended price matrix that integrates this data is:

$$\overrightarrow{P}_{1:T} = \begin{bmatrix} \begin{bmatrix} o_{1,1} & h_{1,1} & l_{1,1} & c_{1,1} & v_{1,1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ o_{M,1} & h_{M,1} & l_{M,1} & c_{M,1} & v_{M,1} \end{bmatrix} \vdots \\ \begin{bmatrix} o_{1,T} & h_{1,T} & l_{1,T} & c_{1,T} & v_{1,T} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ o_{M,T} & h_{M,T} & l_{M,T} & c_{M,T} & v_{M,T} \end{bmatrix} \end{bmatrix}$$

This matrix, with dimensions $T \times M \times 5$, provides a full view of price dynamics and trading activity for each asset across time. It's useful for analyses like technical analysis, trend identification, volatility estimation, and model creation. Additional technical indicators can also be added to enrich the matrix.

4.2.2.2 Return-based

Returns provide insights into changes in asset prices. In Financial Engineering, simple returns and log returns are pivotal.

- **Simple return**: The simple return, r_t, is the percentage change in asset price from time *(t - 1)* to time t:

$$r_t \triangleq \frac{p_t - p_{t-1}}{p_{t-1}} = \frac{p_t}{p_{t-1}} - 1 = R_t - 1 \in \mathbb{R} \tag{14}$$

Simple returns are essential for analyzing asset performance, training Deep Reinforcement Learning agents, and portfolio optimization.

The simple return matrix, $\overrightarrow{\boldsymbol{R}}_{1:T}$, is constructed by stacking the simple returns of the assets:

$$\overrightarrow{\boldsymbol{R}}_{1:T} = \begin{bmatrix} \overrightarrow{\boldsymbol{r}}_{1,1:T} & \overrightarrow{\mathbf{r}}_{2,1:T} & \cdots & \overrightarrow{\boldsymbol{r}}_{M,1:T} \end{bmatrix} \in \mathbb{R}^{T\times M} \tag{15}$$

The portfolio's simple return is:

$$r_t \triangleq \sum_{i=1}^{M} w_{i,t} r_{i,t} = \boldsymbol{w}_t^T \boldsymbol{r}_t \in \mathbb{R} \tag{16}$$

- **Log-return**: The log-return, ρ_t, is:

$$\rho_t \triangleq \ln\left(\frac{p_t}{p_{t-1}}\right) = \ln\left(R_t\right) \in \mathbb{R}. \tag{17}$$

Log-returns are preferred for their additive properties and stabilized variance, especially beneficial in DRL.

4.2.3 Evaluation Metrics

Evaluation metrics play a crucial role in assessing the performance and effectiveness of strategies and models in financial engineering, with specific relevance to DRL. In the field of finance, commonly used evaluation criteria include risk-adjusted metrics such as the Sharpe ratio, which measures the excess return generated per unit of risk. Other metrics like the Sortino ratio focus on downside risk. These criteria help determine the efficiency of investment strategies and provide insights into their risk-return trade-offs. In DRL, additional evaluation criteria may be employed, considering factors like portfolio turnover, transaction costs, and the ability of the agent to adapt to changing market conditions. Ultimately, evaluation criteria serve as benchmarks for assessing the performance of financial models and DRL algorithms, aiding in the selection of optimal strategies and driving innovation in the field. Here we will define some of them as they are fundamental for the understanding of the project.

- **Cumulative (log) return**: Cumulative (log) returns are a widely used measure in financial engineering and DRL to evaluate the overall performance of investment strategies. They represent the aggregated effect of successive returns over time and provide a comprehensive view of profits and loss (PnL). In financial engineering, cumulative (log) returns help assess historical performance and compare portfolios. In DRL, they serve as a key metric for training agents to maximize profits and outperform benchmarks. Overall, cumulative (log) returns play a vital role in evaluating performance and driving innovation in portfolio optimization.

Mathematically the **cumulative simple return** $r_{t\to T}$ is given by:

$$r_{t\to T} \triangleq \frac{p_T}{p_t} - 1 = \left[\prod_{i=t+1}^{T} R_i - 1\right] = \left[\prod_{i=t+1}^{T} (1 + r_i) - 1\right] \in \mathbb{R} \tag{18}$$

while the **cumulative log return** $\rho_{t\to T}$ is:

$$\rho_{t\to T} \triangleq \ln\left(\frac{p_T}{p_t}\right) = \ln\left(\prod_{i=t+1}^{T} R_i\right) = \sum_{i=t+1}^{T} \ln\left(R_i\right) = \sum_{i=t+1}^{T} \rho_i \in \mathbb{R} \tag{19}$$

Obviously one is looking forward to maximize cumulative return over time, rather we are talking about simple or log return.

- **Sharpe Ratio (SR)**: The Sharpe Ratio is a widely used measure in financial engineering and DRL to assess the risk-adjusted performance of investment strategies. It quantifies the excess return generated per unit of risk, providing insights into the efficiency of strategies. In DRL, agents aim to maximize the Sharpe Ratio by learning optimal portfolio policies, striking a balance between risk and return. The Sharpe Ratio plays a crucial role in evaluating risk-adjusted performance and guiding decision-making in finance and DRL.

  We can describe the SR as the ratio of expected returns to their standard deviation of T samples, adjusted by a scaling factor. Therefore it is represented with the following formula:

  $$SR_{1:T} \triangleq \sqrt{T}\frac{\mathbb{E}\left[\mathbf{r}_{1:T}\right]}{\sqrt{\mathrm{Var}\left[\mathbf{r}_{1:T}\right]}} \in \mathbb{R} \tag{20}$$

  Again, one is looking forward to maximize the Sharpe Ratio as it means you have high expected returns and low variance over you return which is more or less the definition of a portfolio with good risk management.

- **Sortino Ratio**: The Sortino Ratio is a risk-adjusted performance measure commonly used in finance and DRL. It evaluates the returns of an investment relative to the downside risk, specifically focusing on the volatility of negative returns. Unlike the Sharpe Ratio, which considers total volatility, the Sortino Ratio emphasizes the risk associated with downward price movements. The formula to compute the Sortino Ratio is as follows [37]:

  $$\text{Sortino Ratio} = \frac{R - R_f}{\text{Downside Deviation}} \tag{21}$$

  where R represents the average return of the investment, R_f is the risk-free rate of return, and the downside deviation measures the volatility of negative returns. The Sortino Ratio provides a valuable metric for investors and DRL agents to assess the risk-adjusted performance of an investment strategy, with a higher ratio indicating a more favorable risk-return trade-off.

- **Maximum Drawdown**: The Maximum Drawdown is a critical metric in assessing the downside risk of an investment portfolio or asset within a specific time frame. It measures the largest observed loss experienced by the investment from its highest peak to the subsequent lowest point, prior to reaching a new peak. In financial analysis, the MDD plays a pivotal role in helping investors comprehend the potential extent of losses during adverse market conditions, allowing for more informed risk management decisions. Mathematically, the MDD is calculated using the formula:

  $$\text{MDD} = \frac{\text{Trough Value} - \text{Peak Value}}{\text{Peak Value}} \tag{22}$$

  where the *Through Value* is the lowest point to which the portfolio value drops after reaching a peak. And the *Peak Value* is the highest point the portfolio value attains before a subsequent decline. The computed MDD is then expressed as a percentage, providing a clear depiction of the drawdown's magnitude relative to the peak value.

4.3 Reinforcement Learning

In preceding sections, we delved into financial engineering concepts, focusing on portfolio optimization and its traditional solutions like the Markowitz model. However, these classical approaches exhibit limitations, suggesting a need for a more dynamic and flexible alternative. RL, a field in machine learning, emerges as a promising candidate due to its decision-making capabilities under uncertain conditions. The essence of RL lies in learning from interaction. An

RL agent explores the environment, observes the outcome of its actions, and adjusts its strategy based on the received feedback. This strategy, or policy, determines the agent's behavior at any given time. The learning process is guided by the concept of a reward signal, which the agent aims to maximize over a period of time as illustrated in Fig. 3.

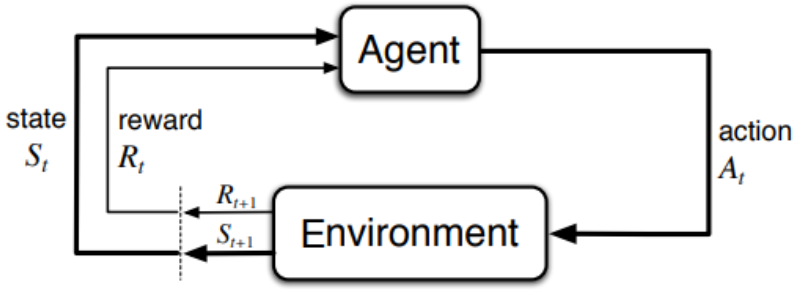

Figure 3: Typical Reinforcement Learning Cycle

As a quick reminder, here is a partial definition of the different elements represented in Fig. 3:

- *Agent:* The decision-making entity that interacts with the environment, learning from it through trial and error.
- *Environment:* The setting in which the agent functions, offering feedback to the agent's moves and guiding the changes in state.
- *Rewards:* Immediate feedback provided by the environment in response to the agent's actions, guiding the learning process towards beneficial states.
- *States:* A formal description of the condition of the environment at a specific point in time, forming the basis for the agent's decisions.
- *Actions:* The set of all possible maneuvers the agent can execute, influencing the state of the environment and the received rewards.

We now turn our attention to understanding RL's underpinning principles, its operation, and its potential as a novel approach for portfolio optimization. We'll start by examining the Markov Decision Process, then we'll describe the key elements of RL. We'll conclude with a detailed look at DRL.

4.3.1 Markov Decision Process

Markov Decision Processes (MDP) play a pivotal role in the realm of reinforcement learning. Essentially, MDPs serve as a mathematical representation for scenarios where decision-making is influenced by both randomness and the choices of a decision-maker. This model offers a structured approach to understand decision-making in contexts where outcomes are influenced by a mix of chance and deliberate actions, capturing the core aspects of a learning challenge.

From this idea, MDPs represent a special type of discrete-time stochastic dynamical systems with robust properties that assure convergence to an optimal policy, that is, the best possible strategy. By adjusting some of the assumptions, MDPs can depict any dynamical system, offering a potent representational framework and a universal approach to control dynamical systems. This are the properties that make MDPs an essential tool in the field of reinforcement learning, as they provide a solid mathematical foundation for learning optimal strategies in various domains, including portfolio optimization. [38].

4.3.1.1 Terms and Concepts

A finite discrete-time fully observable MDP can be represented as a quintuple, denoted as $\langle S, A, P, R, \gamma \rangle$ [39], where:

- S refers to a finite collection of states, also known as the state space.
- A signifies a finite assortment of actions, referred to as the action space.
- P represents the state transition probability matrix, illustrating the likelihood of transitioning from one state to another given an action.
- R serves as the function for generating rewards, signifying the payoff or benefit obtained from a state or an action.
- γ stands for the discount factor, a measure that balances immediate and future rewards in the decision-making process.

Before going any further let's define one of the most fundamental property of MDP: the Markov Property. The Markov property is defined by the principle that the future state depends solely on the current state and action, and not on the sequence of events that preceded it. Mathematically, this can be described as follows:

- **Markov Property:**

$$P[S_{t+1} = s' | S_t = s, A_t = a, \ldots, S_0 = s_0, A_0 = a_0] = P[S_{t+1} = s' | S_t = s, A_t = a] \tag{23}$$

This implies that given the present, the future is independent of the past. This feature simplifies both the representation and computation involved in decision-making problems [40]. One can get a better visualization of the MDP with rewards in Figure 4

One last concept I would like to introduce is the **exploration-exploitation dilemma**, which is relevant for MDP and therefore also relevant for RL. The problem of search under uncertainty is a ubiquitous aspect of both biological and artificial systems [41]. This issue presents itself as a classic dilemma in reinforcement learning, often characterized as a trade-off between exploitation of known opportunities and exploration for potentially superior options elsewhere. This necessitates an agent occasionally taking what might appear as sub-optimal actions, a process which can potentially lead to the discovery of superior strategies [42]. Every reinforcement learning algorithm addresses this exploration-exploitation dilemma, aiming to balance the pursuit of novel opportunities (exploration) with the execution of secure, well-understood actions (exploitation). From an optimization perspective, a purely exploitative or greedy approach might lead to rapid convergence but is at risk of settling into a local minimum. On the other hand, the exploration, while initially slowing convergence, has the potential to expose previously uncharted regions of the search space, potentially yielding a more optimal solution [43]. Traditional algorithms introduce exploration either by injecting artificial noise into the actions, which decreases as the agent gains experience, or by modeling the uncertainty inherent in each action within the Bayesian optimization framework [43].

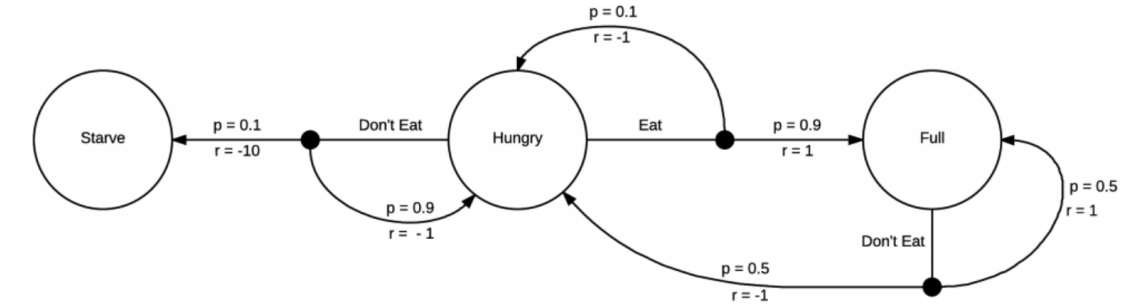

Figure 4: Markov Decision Process with Rewards, schema

4.3.1.2 Evaluation and Bellman Equations

MDPs exhibit a notable characteristic of expressiveness, which enables them to be optimally solved for a variety of problems. The optimality in MDPs can be further elucidated in two main contexts: Value Function and Policy.

1. **State-Value Function**: Denoted by v^*, this function represents the optimal state-value achievable across all potential policies. Formally, for every state s in the set of states S, it is defined as:
$$v^*(s) = \max_{\pi} v_\pi(s) \quad \forall s \in S \tag{24}$$

2. **Action-Value Function**: Symbolized as q^*, this function indicates the supreme action-value attainable across all possible policies. For every state-action pair (s, a) in the cartesian product of sets S and A, it's given by:
$$q^*(s, a) = \max_{\pi} q_\pi(s, a) \quad \forall (s, a) \in S \times A \tag{25}$$

3. **Policy**: Policies in MDPs can be partially ordered based on their corresponding value functions. For any two policies, π and π', we say that $\pi \leq \pi'$ if and only if the value function under π is less than or equal to that under π' for all states $s \in S$, as expressed in Equation 26:
$$\pi \leq \pi' \iff v_\pi(s) \leq v_{\pi'}(s) \quad \forall s \in S \tag{26}$$
This partial ordering among policies culminates in the existence of an optimal policy, π^*, that is superior to or at least as good as all other policies, as stated in Equation 27:
$$\pi^* \geq \pi \quad \forall \pi \tag{27}$$

These statements establish the optimality of MDPs and serve as the theoretical foundation for numerous reinforcement learning algorithms [39].

Within the context of MDPs, the Bellman Equation plays a pivotal role in the understanding and application of these systems, particularly in the realm of portfolio optimization. The Bellman Equation, introduced by Richard Bellman in the late 1950s [44], offers a recursive breakdown of the valuation of a decision-making task at a specific moment. This breakdown is based on the rewards obtained from preliminary decisions and the valuation of the subsequent decision task arising from those preliminary decisions. The Bellman Equation for MDPs hinges on the *Markov Property*, which implies that the future system dynamics are dependent solely on the current state and not on the sequence of events that preceded it.

The policy in an MDP, denoted by π, represents a mapping from states to actions and is generally considered to be stochastic due to the inherent uncertainty in the environment dynamics.

This means that the policy gives the probability of taking an action a in state s, mathematically expressed as:

$$\pi(s|a) = P[a_t = a|s_t = s] \tag{28}$$

In line with the Markov property, the policy is stationary and depends only on the current state, making it time-independent. Consequently, the action at any time t is drawn from the policy given the current state, denoted as:

$$a_t \sim \pi(\cdot|s_t), \forall t > 0 \tag{29}$$

The state-value function v_π under a policy π can be broken down into two parts: the immediate reward and the discounted future reward of the successor state. This decomposition can be captured by the Bellman Expectation Equation for the state-value function as:

$$v_\pi(s) = E_\pi[r_{t+1} + \gamma v_\pi(s_{t+1})|s_t = s] \tag{30}$$

Likewise, the action-value function q_π, which represents the expected return of taking an action a in a state s under policy π, can be decomposed in a similar manner as:

$$q_\pi(s, a) = E_\pi[r_{t+1} + \gamma q_\pi(s_{t+1}, a_{t+1})|s_t = s, a_t = a] \tag{31}$$

In the context of portfolio optimization, the states might represent the current holdings and market conditions, the actions would correspond to buy/sell decisions, and the reward function would reflect the financial return from those decisions. The optimal policy derived from the Bellman Equation would then provide the best strategy for managing the portfolio over time. The Bellman Expectation Equations provide the theoretical foundation for solving MDPs, and have played a vital role in the development of reinforcement learning methods used for various decision-making problems, including but not limited to, portfolio optimization.

4.3.1.3 Continuous Markov Decision Process

Until now, we have considered the MDP in scenarios with finite Action and State spaces, denoted as A and S respectively. However, in the context of portfolio optimization, it becomes apparent that both the Action and State spaces are non-discrete and finite. This necessitates the introduction of an **Infinite Markov Decision Process**, which permits the existence of an infinite Action and/or State space.

Implementation of the policy π and/or the action-value function q_π in such a context calls for the application of differentiable function approximation methods. As stated earlier, DNNs are capable of acting as general function approximators. Therefore, one could leverage DNNs to evaluate the policy and/or the action-value function in these scenarios [45]. This approach paves the way for the integration of the DRL concept, which we will discuss in greater depth later.

4.3.2 From Markov Decision Process to Reinforcement Learning

So far, we have been working with stochastic processes. How do we transition to a learning problem? We refer to learning when the transition dynamic P and the reward function R are unknown. Therefore, the MDP becomes an RL problem, which is to find the optimal stationary

policy π while simultaneously learning P and R. This learning process can be either explicit or implicit, leading to model-based and model-free reinforcement learning, respectively. RL offers a robust structure for tackling issues that align well with the MDP framework. The success of RL is contingent upon how fitting the MDP model is to the given issue. It's not imperative to be aware of every potential transition, but a precise overarching depiction is vital. On the flip side, issues that cannot be encapsulated by an MDP might not be suitable for the RL methodology.

4.3.2.1 Terms and Concepts

Once more we will introduce in this section the more relevant terms and concepts needed to better understand how RL works. In adherence to the MDP definitions, an RL system is typically constituted by four main components. These include the reward mechanism, which provides the feedback signal used to optimize decisions; the value functions, which estimate the expected cumulative future reward from a given state or state-action pair; the policy, which determines the action selection behavior; and the environment, which encapsulates the dynamics of the decision-making problem. You can see how it lies down with the schema presented in the introduction in Fig. 3. The RL framework applied within these parameters exhibits significant potential for robust and effective solutions in fields such as finance. Let's detailed each component and provide an example of how they could be involve in the use case of solving portfolio optimization problem using RL.

- **Policy (π):** This represents the plan of action adopted by the agent, essentially a function that associates states with actions. It dictates the agent's actions in the environment, steering it towards decisions that result in optimal rewards [39]. Mathematically, the policy quantifies the likelihood of executing action a_t when in state s_t, denoted as $\pi(a|s) = P[At = a|St = s]$. In the context of portfolio management, the policy might determine factors like the fraction of total capital to allocate to each potential investment.

- **Rewards (R):** These are feedback signals received from the environment following an action taken by the agent. The objective of the agent is to maximize the cumulative reward over time [39]. For portfolio optimization, rewards might correspond to the financial return or profit from investment decisions.

- **Value Function (v_π, q_π):** This function assesses the expected long-term return from a given state or action-state pair, under a particular policy π [46]. The state-value function $v_\pi(s)$ is defined as $E_\pi[G_t|st = s]$, and the action-value function $q_\pi(s, a)$ is defined as $E_\pi[G_t|st = s, at = a]$. In portfolio optimization, these functions would help evaluate the prospective returns from different investment strategies.

- **Environment (Model):** This refers to the setting in which the agent operates, influencing the states, actions, and rewards. The model predicts the next state s_{t+1} and reward r_{t+1} given the current state s_t and action a_t, represented by a state transition probability matrix P given by $P^a_{ss'} = P[s_{t+1} = s'|st = s, at = a]$ [39]. In portfolio optimization, the environment could encompass factors like market conditions, asset prices, and the portfolio's current holdings.

4.3.3 Reinforcement Learning Methods and Algorithm

The selection of an appropriate method or algorithm for solving the problem framed as a MDP hinges upon various factors. These factors include the nature of the environment, the structure of the state and action spaces (discrete or continuous), and more. Consequently, one method might be favored over another based on these considerations. In the ensuing sections, we will initially delineate the concepts of model-free and model-based algorithms. Thereafter, our focus will shift to model-free algorithms, where we will distinguish between value and policy

approaches. Subsequently, we will explicate the concepts of on-policy and off-policy algorithms. The overarching objective of this analysis is to obtain a comprehensive overview of the methods available, thereby facilitating the identification of the most suitable approach for our specific use case: portfolio optimization. One may note that in Table 19 you can find a list of RL algorithms taken from the biggest encyclopedia in the world, wikipedia. I also unclude a schema representing the Taxonomy of RL algorithm in Figure 5 taken from [1].

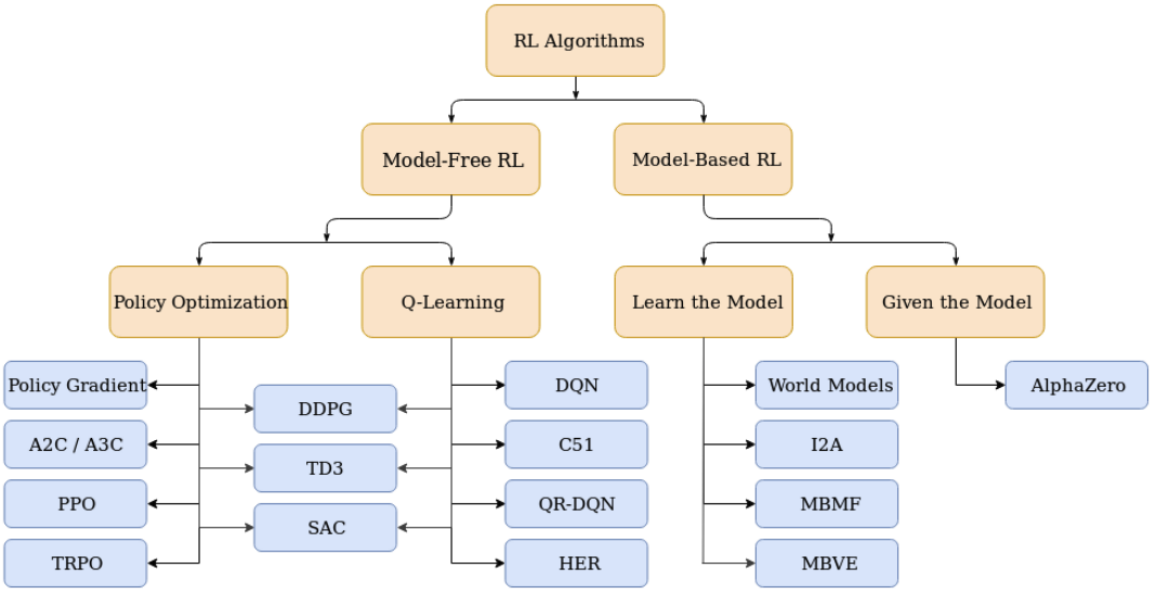

Figure 5: Taxonomy of RL Algorithms [1]

4.3.3.1 Model-free vs. Model-based

In the field of RL, methods are often categorized as either model-based or model-free, based on their utilization of environmental dynamics during learning [39].

Model-based methods attempt to learn a model of the environment's dynamics and rewards explicitly. This framework is capable of forecasting the subsequent state and associated reward based on the present state and chosen action. The advantage of these methods is their efficiency; they typically require fewer interactions with the environment to achieve good performance, as the learned model is used to simulate experiences. However, they can be computationally expensive, especially in environments with high-dimensional state and action spaces, as the model might need to be accurate and comprehensive [47].

On the other hand, model-free methods bypass the necessity of building an explicit model of the environment. They directly learn either a value function or a policy from interactions with the environment, making no assumptions about the underlying dynamics of the environment [39]. These methods can be simpler to implement and typically less computationally demanding than model-based methods, but they may require more interactions with the environment to achieve optimal performance [47].

In the context of portfolio optimization, model-free methods might be more suitable for several reasons. The primary reason is the complex and stochastic nature of financial markets, which makes it extremely challenging to accurately model the dynamics. Additionally, financial data are often noisy, and the underlying processes can change over time, resulting in non-stationarity [48]. Given these factors, a model-free approach, which can adapt to the dynamics through

direct interactions rather than relying on an approximate model, is often more effective and robust for portfolio optimization [49].

4.3.3.2 Value-based vs. Policy-based approach

As stated earlier, we generally distinguish between two principal methodologies: value-based and policy-based approaches. Value-based methods, such as Q-Learning [50] and Deep Q Networks, operate by determining the value of each state or action, hence building what we call a value function. The goal is to maximize the cumulative reward by choosing actions that lead to states with the highest value. Conversely, policy-based methods, like the Policy Gradient method or Actor-Critic methods, directly optimize the policy function without the need for a value function. These methods adjust the policy parameters in a way that maximizes the expected return. Studies have shown that policy-based methods usually have better convergence properties than the value-based approaches [51] [52]. However, these methods may suffer from high computational complexity as the dimensionality of the state and/or action space increase.

For portfolio optimization problems, value-based methods are typically favored. This preference primarily stems from the inherent structure of these problems. Portfolio optimization often involves a large state space and requires the ability to quantify the value of holding different portfolios, which aligns well with the mechanisms of value-based methods. They are designed to effectively handle high dimensional spaces and can provide precise valuations of different states, which, in this context, represent different portfolio configurations. Furthermore, value-based methods are generally more stable and have fewer convergence issues, making them particularly suitable for financial applications where stability and reliability are critical.

4.3.3.3 On-Policy vs. Off-policy

Finally, we can categorize the algorithms as either on-policy or off-policy based on how they use their experience to learn. On-policy methods, such as SARSA (State-Action-Reward-State-Action) [53], learn the value of the policy being executed, meaning they are continually learning about and updating the policy that they are using to make decisions. In other words, the data for learning and the data for making decisions are collected from the same policy.

Contrarily, off-policy methods, like Q-learning, learn the value of an optimal policy while following an exploration policy. They are able to learn from the actions of any policy, regardless of the one currently in use. This provides them with a significant advantage: they can learn from past experiences, even those taken under different policies, allowing the use of previously collected data for improved learning efficiency.

In the case of portfolio optimization, off-policy methods are often preferable. Their ability to learn from old experiences can prove highly beneficial given the dynamic nature of financial markets, where old data can still provide valuable insights. Furthermore, these methods can optimize the policy without exposing it to potential risks during the learning process. This is particularly important in a financial context, where making sub-optimal decisions can have significant monetary consequences.

4.3.4 Deep Reinforcement Learning

When the state and action spaces are very large and highly dimensional, it is practical to use neural network-based approximations for both value functions or policies. The problem then becomes one of functional approximation, and neural networks excel in this kind of problem. Indeed, neural networks are well suited for high-dimensional inputs and can be trained incrementally, making use of additional samples obtained as learning happens. Different architectures of

neural networks, such as Long-Short-Term-Memory (LSTMs), CNNs, or RNNs, can be used to solve approximate functions. This kind of process where you use a neural network in order to evaluate the approximation of the value or policy function is known as DRL. One may observe a schematize representation of a cylce in DRL in Figure 6

In the context of value-based methods, a Deep Value-Based RL algorithm has been introduced: Deep Q-Network (DQN). It can be seen as a generalization of the Q-learning algorithm with functional approximation. Basically, the Q-function is parameterized using a neural network function. At each step, the parameters are updated towards the target value using, for example, the stochastic gradient descent method in order to minimize the squared loss between the target function and the neural network approximation. For best results, DQN [54] proposes two novel ideas to avoid instability and overfitting. These ideas are the slow-update of the target function and the experience replay [55].

On the other hand, a Deep Policy Gradient RL algorithm has also been introduced. Instead of parameterizing the Q-function, the policy pi is parameterized by a neural network. This method leads to neural Actor-critic algorithms, neural Proximal Policy Optimization (PPO) / rust Region Policy Optimization (TRPO) [56], and DDPG [57]. DDPG can be seen as a combination of DQN and Deep Policy Gradient (DPG) algorithms. It is a model-free, online, off-policy RL method. A DDPG agent is an actor-critic reinforcement learning agent that searches for an optimal policy that maximizes the given reward wich in PO is usually defined as the expected cumulative long-term reward. The state space can be either continuous or discrete, the action space is continuous, and the critic agent is defined as the Q-value function while the actor agent is defined as the deterministic policy agent. Finally, Twin Delayed Deep Deterministic Policy Gradient (TD3) [58] is a reinforcement learning algorithm that combines elements of both DDPG and DQN. It is designed to address the issue of overestimation bias in traditional Q-learning methods. TD3 employs twin value networks and delayed policy updates to improve stability and convergence in training. By utilizing multiple critic networks and target networks, TD3 enhances the estimation of the Q-values and improves the performance of the learned policy.

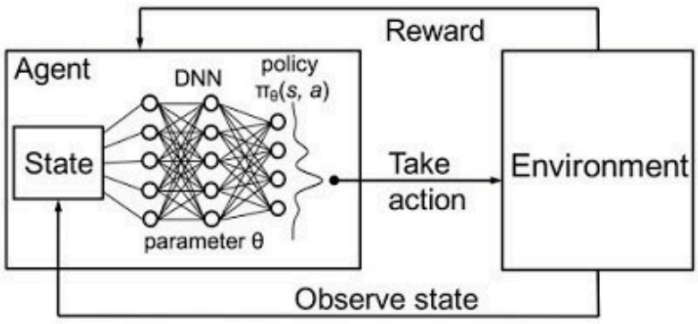

Figure 6: DRL cycle schema [2]

4.3.5 Reinforcement Learning and Portfolio Optimization

4.3.5.1 The Ideal Approach ?

We have discussed the various mechanisms behind RL and explored which methods are best suited for portfolio optimization and why. Now, let's take a look at how RL overcomes the limitations of traditional methods discussed earlier in this report.

Firstly, RL addresses two of the main drawbacks of Modern Portfolio Theory. It does not

depend on specifying the joint dynamics between assets, and RL methods are particularly effective at maximizing long-term rewards even if it means taking sub-optimal actions in the short term [59]. This ability to make long-term decisions is why RL methods are an ideal choice for portfolio management.

Secondly, RL also overcomes the limitations of modern trading decision techniques, whether using deep learning methods or not. RL takes advantage of large datasets and new computational methods to identify patterns without requiring specific information about their location or appearance. Moreover, RL is an unsupervised learning method that doesn't require any labeling, which reduces the need for manual work and dependence on labeling methods.

4.3.5.2 Recent Advances in Research

In order to solve the portfolio optimization problem, both value-based methods such as Q-learning [50], SARSA [53], and DQN [60]; and policy-based methods such as DPG and DDPG [61, 62, 63] have been used - usually extended to Deep RL approaches. We have already mentioned that state variables often consist of asset prices, time, past returns of assets, current holdings, remaining balance, and more, while the control variable is typically the amount/proportion of wealth invested in each component of the portfolio. Rewards are usually described as portfolio returns [53, 63], profit, and sometimes Sharpe Ratio [50, 53]. More recently, studies have tried to incorporate transaction costs and risk-free assets in the equation to build more realistic portfolios with realistic market conditions. Initially, studies were limited to a small number of assets, sometimes only 2 (a basic asset and the risk-free asset). However, thanks to Deep RL methods such as DDPG or TD3, recent studies have attempted to increase the number of assets available in the portfolio collection. The importance of data is fundamental, and some papers have leveraged Generative Adversarial Networks (GANs) to generate synthetic market data, allowing for better exploration during the training of RL methods. However, such methods are limited by the data used to build the GAN. It is also worth noting that many types of markets have been studied, from the S&P 500 to the cryptocurrency market. For an extensive review of RL methods used for portfolio optimization, we recommend chapter 4.3 from [64], which lists dozens of papers related to this area of research.

4.3.5.3 Drawbacks of Current Approaches

Even though there has been a surge of interest in using RL for portfolio optimization in recent years, much of the research has focused on two main areas: testing and comparing a wide range of RL methods on different markets to maximize returns over time. It is clear that most papers use returns or profit as a reward for their RL methods and usually implement various value-based and policy-based methods to compare their effectiveness. However, portfolio optimization is not just about returns, but also about risk management. The fact that most of the RL methods used are not risk-driven usually results in high variance and large draw-down during testing. Another major limitation observed in the research is that most of them try to build a single optimal portfolio, rather than building multiple portfolios adapted to different risk profiles. Again, this view does not represent the investor's needs in terms of portfolio management. Interesting research such as [10] has built different portfolios with low correlation using a multi-agent reinforcement learning method and a well defined global loss function. However, portfolios are then aggregated/combined to build the optimal portfolio once again. Moreover, even if some constraints have been included to reduce correlation between the portfolios of each agent, the reward used was the same for each of them. Another limitation observed is that most of the research usually uses the same state space. While some have tried to include sentiment analysis of the market or other alternative data, the majority of research uses the "open, high, low, close" signals of assets as input data. Lastly, while most research tries to implement and compare

different Deep RL methods, almost none of them focus on the "deep" part of the model, i.e. how the neural network architecture used to approximate policy or value function impacts the results of the model. Therefore, we will focus on these limitations to propose an innovative approach to studying the portfolio optimization problem.

4.3.6 Our Novel Direction for Investigation

Based on the description of the portfolio optimization problem, the methods used to solve it, and the limitations of such methods, we propose an approach to study this problem. Our initial focus will be on the S&P 500 market, using a pre-selection of a certain amount of assets and daily data. However, including other asset classes such as treasury bonds could be interesting, as it would allow agents to use assets with lower correlation, lower volatility, and risk. This is the kind of behavior we can observe in asset management funds such as family offices. We might therefore think about working with this kind of asset class during our study. To address the limitations mentioned earlier, we will mainly focus on the following points of interest:

- Build a framework that allows for the study of portfolio optimization problem using DRL methods and comparing them with traditional investing strategies as benchmarks.
- Investigate and compare the performance of different neural network architectures, such as CNN, RNN, and CRNN, in approximating value or policy functions in DRL methods, and analyze their impact on the overall results.
- Incorporate additional data, such as technical indicators computed from OHLC data or alternative data sources, into the model and state space. Explore the use of well-known technical indicators (e.g., RSI, Williams % R, MACD) and consider incorporating indicators from established research papers (e.g., 101 Formulaic Alphas). Investigate the integration of alternative data like press releases, public reports, SEC filings, and other sources for enhanced portfolio optimization.
- Explore the application of multi-agent reinforcement learning to construct diverse portfolios driven by different risk-oriented rewards, leveraging risk-free assets to emphasize risk management. This includes considering rewards based on metrics like asset volatility, Sharpe ratio, R-squared, and more. Additionally, develop strategies for combining the results of each agent to propose optimal portfolios at various risk levels.

5 Methods and Experiments

5.1 Libraries

In this project, we employed various libraries to facilitate the development of our portfolio optimization system based on DRL. The entire project was implemented using the Python programming language.

1. **Scientific Libraries:** To perform essential scientific computations, we utilized common scientific libraries. For array processing, we relied on NumPy, a fundamental library providing efficient array operations. Additionally, Pandas was employed to optimize data structures, allowing us to handle and manipulate large datasets effectively.

2. **Basic Libraries:** For the seamless interaction with the framework, we integrated commonly used libraries such as argparse, sys, pprint, and json.

3. **Data Retrieval and Storage:** To obtain the daily OHLC data of selected assets, we leveraged the yfinance library, which offers a simple interface to access financial data from Yahoo Finance. The collected and aggregated data was then stored in the .h5 format using the hdf5 library.

4. **Deep Reinforcement Learning Framework and Environment:** The construction of the environment within our DRL framework was accomplished using the gym library provided by OpenAI. Gym offers a convenient platform to implement and compare various DRL algorithms. It is noteworthy that the development of our framework involved building upon and extending the work of other researchers who publicly shared their code and/or research outcomes. Notably, Jiang [62] [1], [2] and [3], implementation significantly contributed to the development of our framework. Note that our benchmark algorithms are directly taken from the Universal Portfolio repository [4].

5. **Artificial Intelligence Components:** ML and DL components were pivotal to our project's success. Specifically, DNN architectures and behaviors were designed using either the relearn or TensorFlow libraries. The primary architecture of our final framework was implemented using the tflearn library, ensuring consistency and modularity for deep learning implementations in Python. It is essential to mention that tflearn builds on top of TensorFlow, enabling better transparency in the implementation of deep learning models.

6. **Visualization:** To visualize our results through graphs and plots, we employed seaborn and matplotlib. These libraries provided us with powerful tools to create figures for this report, allowing us to communicate our findings effectively.

In the appendix, you will find the detailed information regarding the specific versions of the libraries used in this project. It is pertinent to note that we deliberately utilized older versions of certain libraries to ensure compatibility and to facilitate the reuse of existing code with minimal modifications.

5.2 Assumptions

In order to advance our research, we have formulated certain assumptions concerning the market. These assumptions are designed to streamline the process by simplifying complex aspects involved in portfolio optimizations. The key assumptions made are as follows:

[1] https://github.com/ZhengyaoJiang/PGPortfolio
[2] https://github.com/NigelCusc/
[3] https://github.com/bassemfg/ddpg-rl-portfolio-management
[4] https://github.com/Marigold/universal-portfolios

1. **Zero Market Impact:** We assume that when trading an asset, our actions do not significantly disturb the balance between supply and demand (following the supply and demand law), especially when dealing with large volumes. While buying or selling an asset can potentially impact the market and affect asset prices, this phenomenon has been extensively studied, with some researchers attempting to predict or define rules to incorporate its effects into their studies. Nonetheless, for the sake of simplicity and efficiency in our research, we adopt the following assumption: the capital invested by the agent, irrespective of the trading volume and the specific asset, does not exert any influence on the market or asset prices. This primary assumption forms the basis for the two subsequent assumptions.

2. **Optimal Liquidity:** We define an asset as "liquid" if it can be rapidly converted into cash with minimal or no loss in value. A highly liquid asset ensures that the agent can execute buying or selling transactions promptly whenever required. Consequently, we assume that all assets within our scope of study are liquid, and any transaction can be executed instantaneously without encountering liquidity issues.

3. **Zero Slippage:** Slippage, referring to the difference between the price set when placing an order and the actual price at which the order is executed, often arises in two scenarios: firstly, when the asset experiences high volatility, making it challenging to place an order at a specific price and execute it due to rapid price fluctuations; secondly, when the asset's liquidity is low, resulting in difficulties finding suitable orders that match the expected value for both buyers and sellers. In order to mitigate such complications, we make the assumption that the market's assets possess sufficiently high liquidity, enabling orders to be executed instantaneously at the most recent price, hence implying zero slippage.

These assumptions find widespread application and are commonly adopted in numerous research projects. Furthermore, it is noteworthy that the viability of these assumptions is contingent upon our utilization of daily prices and our practice of reallocating our portfolio solely on a daily basis. It is important to acknowledge that these assumptions may become unrealistic if one were to work with high-frequency data and allocation. However, within the context of our research, where the traded assets' volume in the market is substantial, these assumptions remain valid [65].

5.3 Data

In the following section, you will find an extensive description of the datasets used and the key processing steps. All of our datasets share the same basis, consisting of selected assets and a specific time period, to ensure the relevance of our comparisons. We will first describe the basis used for our datasets, and then elaborate on the three different datasets built.

5.3.1 Basis

To conduct our research, we needed to determine the market on which to focus. This decision involved considering various options such as the cryptocurrency market, forex market, and stock markets. Ultimately, we chose to work with the stock markets due to several reasons. Firstly, since we aim to perform daily allocation, it was more feasible to work with a market that operates on a daily basis, unlike the cryptocurrency market, which is open 24/7. Additionally, focusing on stock markets provided us with a solid baseline, as they are well-established and widely used in financial research. The S&P500 Exchange, also known as the Standard and Poor Exchange, was selected as our primary market, encompassing a list of 500 stocks. From the S&P500 Exchange, we carefully selected 25 stocks based on their market capitalization and trading activity to align with the assumptions stated above. The exhaustive list of stocks used in our experiments can be found in Table 1. For these 25 assets, we retrieve the data on a daily basis (open market

days) over 25 years from 06/01/1995 to 29/09/2020. In order to train and test our methods, we split the dataset into two parts. The first part represents the first 6/7 of the dataset, and the testing part represents the last 1/7 of the data. The total period range of the dataset is from 06/01/1995 to 29/09/2020, and the training period extends up to 25/01/2017, Table 2

Ticker	**Complete Name**
ABT	Abbott Laboratories
CMCSA	Comcast Corporation
INTC	Intel Corporation
PFE	Pfizer Inc.
JPM	JPMorgan Chase & Co.
TMO	Thermo Fisher Scientific Inc.
UNH	UnitedHealth Group Incorporated
T	AT&T Inc.
WMT	Walmart Inc.
VZ	Verizon Communications Inc.
CSCO	Cisco Systems, Inc.
DIS	The Walt Disney Company
MSFT	Microsoft Corporation
JNJ	Johnson & Johnson
NKE	Nike, Inc.
PG	Procter & Gamble Co.
CVX	Chevron Corporation
KO	The Coca-Cola Company
BAC	Bank of America Corporation
XOM	Exxon Mobil Corporation
MRK	Merck & Co., Inc.
AAPL	Apple Inc.
PEP	PepsiCo, Inc.
HD	The Home Depot, Inc.
ADBE	Adobe Inc.

Table 1: List of Stocks from S&P500

Dataset	**Start Date**	**End Date**
Total Period	06/01/1995	29/09/2020
Training Period	06/01/1995	25/01/2017
Testing Period	26/01/2017	29/09/2020

Table 2: Period Ranges for Training and Testing

5.3.2 Dataset 1: Closing Prices

Based on the foundation established in the previous subsection, we proceed to construct our first dataset, which is the most straightforward one. For each stock, on every market day, we have exclusively retrieved the closing price. As a result, this dataset can be regarded as a price matrix, as explained in the Scientific Background section. It encompasses the 25 assets selected earlier, covering a time period from 1995 to 2020. This dataset serves as our baseline dataset, given its simplicity and the fact that it solely relies on price information without incorporating any additional data. A visual representation of closing prices is given in Figure 7.

Figure 7: Time series plot displaying the closing prices of two selected stocks, AAPL and MSFT, over the period from 1995 to 2020

5.3.3 Dataset 2: OHLC

The second dataset is similar to the first one, except that instead of solely retrieving the closing price on a daily basis, we now retrieve the complete OHLC (Open, High, Low, Close) data. As a result, the structure of this dataset can be envisioned as an extended price matrix, as described in the Scientific Background section. With the inclusion of OHLC data, the agent gains access to intraday fluctuations, making it possible to capitalize on high and low price information. These additional pieces of information are commonly utilized in technical analysis, and one may anticipate that the agent can leverage them to enhance its strategy. A visual representation of OHLC prices for 1 asset is given in Figure 8.

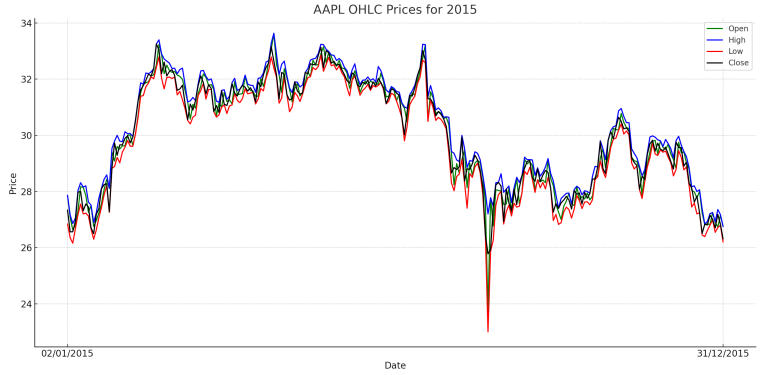

Figure 8: Simplified OHLC Line Plot for AAPL for the year 2015

5.3.4 Dataset 3: Technical Indicators

The third and final dataset is an extension of the second dataset. Similar to the second dataset, it includes the OHLC data. However, in addition to the OHLC data, we augment the dataset with six technical indicators, providing the agent with extended information to aid in strategy development. Technical indicators are widely used in finance to provide insights into the strength of a trend, potential price reversals, volatility, and measure overbought or oversold conditions. The selected technical indicators are as follows:

1. MA (Moving Average): A trend-following indicator that calculates the average price over a specified time period. Used for On-line Moving Average Reversion (OLMAR).

2. MM (Moving Median): The median is considered more robust to outliers than the mean. Used by Robust Median Reversion On-line Portfolio Selection (OLPS) algo.

3. MOMENTUM: A momentum indicator that measures the speed at which a stock's price is changing.
4. ROC (Rate of Change): Measures the percentage change in price over a specified time period.
5. BOLLINGER BANDS: A volatility indicator consisting of a moving average and upper and lower bands.
6. STD (Standard Deviation): Measures the dispersion of stock prices from their mean.

Please note that the time period for each technical indicator corresponds to the window length used by the model, which may vary depending on the experiments. By incorporating these indicators, we aim to investigate whether the inclusion of additional data enhances our models' ability to discover optimal trading strategies. A visual representation of technical indicators is given in Figure 9.

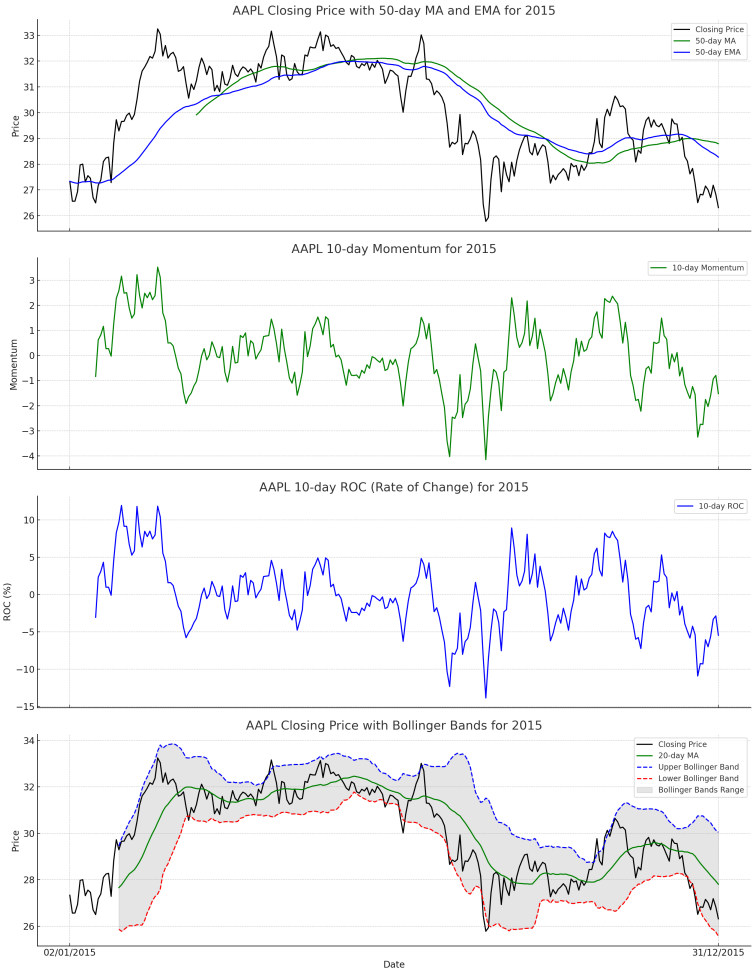

Figure 9: 1. closing price along with the 50-day MA and EMA. 2. 10-day momentum, highlighting the speed of price changes. 3. the 10-day ROC, which measures the percentage change in price. 4. closing price with Bollinger Bands, providing insights into the stock's volatility.

5.4 Benchmarks

In order to assess the validity of our strategy, it is essential to compare its performance against relevant benchmarks. Considering that our study involves stocks from the S&P500, a straightforward and evident benchmark is the overall market performance, which has exhibited substantial

growth at approximately 10.7% per year over the last three decades. However, in the realm of studying DRL algorithms, it is noteworthy that many research papers utilize OLPS methods as benchmarks. [66], [67], [68] OLPS methods represent a class of algorithms designed to dynamically allocate assets within a portfolio over time. Unlike traditional portfolio optimization methods that rebalance periodically, OLPS methods make continuous adjustments to portfolio weights based on incoming market data. These methods aim to optimize the portfolio's performance in real-time, considering the changing market conditions. The interest in using OLPS methods as benchmarks when evaluating Reinforcement Learning algorithms stems from several reasons. First, OLPS methods share similarities with the learning and decision-making process in Reinforcement Learning. Both fields involve the sequential allocation of resources to maximize cumulative rewards or returns while facing uncertain and changing environments. As a result, OLPS methods offer a natural comparison point for studying the effectiveness of RL algorithms in financial decision-making scenarios. Secondly, OLPS methods have been extensively studied and benchmarked themselves, providing a well-established framework for performance evaluation. Researchers have developed various OLPS algorithms with different objectives, such as minimizing regret or maximizing wealth. These methods have been tested under diverse market conditions and have demonstrated their ability to handle practical challenges, such as transaction costs, market impact, and liquidity constraints. Using OLPS techniques as reference points, we can evaluate how our DRL-driven portfolio optimization approach stacks up against renowned and prevalent methods in the online finance domain. Moreover, the use of OLPS methods facilitates fair and objective evaluations, as these benchmarks are not only theoretically sound but also grounded in empirical evidence from real-world financial markets. It is interesting to note that OLPS algortihms are divides into 2 groups: "follow the winner" and "follow the loser" strategies. "Follow the loser" is an investment strategy where investors buy assets that have performed poorly in the hope that they will recover, while "follow the winner" is a strategy where investors buy assets that have performed well, expecting them to continue performing well. In order to explore both group we decided to use 3 benchmarks: a classic one Best Constant Rebalanced Portfolio (BCRP), a follow the loser strategy OLMAR and a follow the winner strategy Universal portfolio (UP). We will briefly explain the different benchmarks in the following subsections. However as we don't implement them ourselves and only integrated them to our framework I encourage you to follow the references for more details.

5.4.1 Best Constant Rebalanced Portfolio (BCRP) Algorithm

BCRP serves as a retrospective optimal strategy, crafted to assess the efficacy of online portfolio selection techniques. As proposed by Li and Hoi in 2014 [69], BCRP aims to determine the best constant portfolio weights retrospectively, maximizing the log cumulative wealth of a CRP over a given time period. The optimal portfolio weights b^* are obtained by solving the following optimization problem:

$$b^* = \arg\max \sum \log(b^\top x_t)$$

where b^* belongs to the valid portfolio simplex Δ_m and x_t represents the price relative vector. The final cumulative portfolio capital obtained by BCRP is computed as follow:

$$S_n(BCRP) = \max\{S_n(CRP(b)) \mid b \in \Delta_m\}$$

where $S_n(BCRP)$ is the cumulative wealth and $S_n(CRP(b))$ denotes the wealth of a CRP with portfolio weights b. BCRP provides a benchmark to assess the performance of various online portfolio selection strategies, including DRL-based methods, in practical scenarios [69].

5.4.2 OLMAR (Online Moving Average Reversion) Algorithm

OLMAR is an online portfolio selection algorithm proposed by Li and Hoi in 2012 [70]. It belongs to the class of Follow-The-Winner strategies, which aim to allocate more capital to assets that have shown positive price movements in the past and less to those with negative movements. Unlike traditional single-period mean-reversion strategies, OLMAR leverages multiple-period mean reversion using Moving Average Reversion. This approach helps to address potential failures in single-period prediction methods. The next price relative in OLMAR is calculated using the following formula:

$$x_{t+1}(w) = \frac{p_t}{w} \odot \left(1 + x_t + \ldots + \frac{1}{w-2} x_{t-w+1}\right)$$

where $x_{t+1}(w)$ is the predicted next price relative, p_t is the price vector corresponding to x_t, w is the window size, and $\odot$ denotes element-wise product. OLMAR's reliance on simple moving averages provides an effective solution to improve predictions and enhance portfolio performance.

5.4.3 Universal Portfolios (UP) Algorithm

Introduced by Cover in 1996 [71], Universal Portfolios stands as a significant algorithm within the realm of online portfolio selection. UP can be viewed as a weighted mean of all consistent rebalanced portfolios, where the weights are determined by historical performance. The strategy aims to maximize cumulative wealth by dynamically allocating capital among different assets over time. The wealth of a particular portfolio manager in UP is given by:

$$S_n(UP) = \frac{S_n(b)s^{\mu}(b)}{\sum S_n(b)s^{\mu}(b)}$$

where $S_n(UP)$ is the cumulative portfolio wealth, $S_n(b)$ represents the wealth of a CRP manager's portfolio, and $s^{\mu}(b)$ denotes the distribution on the space of valid portfolio Δ_m. UP achieves performance by maintaining a weighted average of individual CRP managers' wealth, reflecting their historical performance.

5.5 Agent Environment

As previously discussed, DRL algorithms operate within a specific environment, analogous to the world in which the agent evolves. The significance of the environment stems from the nature of the MDP model. The environment encompasses the conditions in which the agent operates, thereby influencing the states, actions, and rewards, as defined earlier. It is essential to highlight that our environment is designed to allow the model to adjust the allocation of each asset every open market day. Consequently, we define a step as one open market day. Let us now provide a detailed description of the environment.

5.5.1 State Space

Recall that we previously defined the state as "a formal description of the condition of the environment at a specific point in time, forming the basis for the agent's decisions." Hence, it is evident that in our case, the state will be dependent on the dataset used. Consequently, we will provide the state description for each dataset. However, before delving into the specifics, let us revisit some fundamental concepts. Building upon the description of the price vector defined earlier, we will now introduce the relative price vector. The relative price vector at time t is derived from the price vectors at time t and $t-1$ as follows:

$$\text{Relative Price Vector at time } t: \quad \mathbf{r}_t = \frac{\mathbf{p}_t}{\mathbf{p}_{t-1}}$$

with

$$\mathbf{p}_t = \begin{bmatrix} p_{t,1} \\ p_{t,2} \\ \vdots \\ p_{t,M} \end{bmatrix}$$

In line with a similar concept, when leveraging OHLC data, it is important to highlight that we calculate relative high and low vectors by substituting the price at time t with the higher/lower price at the same time point. Once we obtain the relative price/high/low vectors, we apply logarithmic normalization, resulting in log-normalized relative vectors. So we can finally describe the Relative price vector as:

$$R_t = \left(\frac{P_{t1}}{P_{t-1,1}}, \frac{P_{t2}}{P_{t-1,2}}, \ldots, \frac{P_{tM}}{P_{t-1,M}} \right)$$

and finally, for the log-normalized relative vectors:

$$\log \mathrm{R}_t = \left(\log \left(\frac{P_{t1}}{P_{t-1,1}} \right), \log \left(\frac{P_{t2}}{P_{t-1,2}} \right), \ldots, \log \left(\frac{P_{tM}}{P_{t-1,M}} \right) \right)$$

The state representation is influenced by the choice of window lengths, which can be considered as a flexible parameter indicating the number of past time steps deemed relevant at each period and thus impacting the size of each state. For instance, a window length of 3 includes information for timestamps t, t-1, and t-2 instead of just information at timestamp t, akin to extracting a slice of the price matrix discussed earlier. Throughout the project, we explored five different window lengths: 3, 5, 7, 9, and 11 days, which are applicable for daily allocation, enabling us to assess the generality of our results and identify specific window lengths that work more effectively with particular models or datasets. For simplicity, we will consider a window length of 1 for the subsequent state description. Additionally, it is noteworthy that we add an extra entry to each state, denoted as the first entry, which is fixed at 1 (constant). This entry signifies the cash in the portfolio, affording the agent the option not to trade any asset while maintaining the trading currency. We can know illustrate the state space for each dataset:

1. **State 1**: The state at time t is a vector of length $M + 1$, where M defines the number of assets (25 in our case). The values within the vector correspond to the log-normalized RP of each asset at time t, except for the first element, which represents the cash and is therefore fixed at 1. The representation is as follows:

$$\mathrm{State}_{\text{dataset 1}}(t) = [1, \log(\mathrm{RP}_1(t)), \log(\mathrm{RP}_2(t)), \ldots, \log(\mathrm{RP}_M(t))]$$

2. **State 2**: The state at time t is very similar to the state for dataset 1, but with the addition of log-normalized relative H and L prices for each asset at time t. This allows us to leverage the OHLC data provided by dataset 2. The state is now represented as a matrix of size $3 \times (M + 1)$, as we have RP, H, and L information instead of only RP:

$$\mathrm{State}_{\text{dataset 2}}(t) = \begin{bmatrix} 1 & \log(\mathrm{RP}_1(t)) & \log(\mathrm{RP}_2(t)) & \ldots & \log(\mathrm{RP}_M(t)) \\ 1 & \log(\mathrm{H}_1(t)) & \log(\mathrm{H}_2(t)) & \ldots & \log(\mathrm{H}_M(t)) \\ 1 & \log(\mathrm{L}_1(t)) & \log(\mathrm{L}_2(t)) & \ldots & \log(\mathrm{L}_M(t)) \end{bmatrix}$$

3. **State 3**: The state at time t is similar to the state of dataset 2, but with the addition of X Technical Indicator (TI) values to the log-normalized RP of each asset at time t. The

state is now represented as a matrix of size $(3 + X) \times (M + 1)$:

$$\text{State}_{\text{dataset 3}}(t) = \begin{bmatrix} 1 & \log(\text{RP}_1(t)) & \log(\text{RP}_2(t)) & \dots & \log(\text{RP}_M(t)) \\ 1 & \log(\text{H}_1(t)) & \log(\text{H}_2(t)) & \dots & \log(\text{H}_M(t)) \\ 1 & \log(\text{L}_1(t)) & \log(\text{L}_2(t)) & \dots & \log(\text{L}_M(t)) \\ \text{TI 1} & \text{TI 1 value}_1(t) & \text{TI 1 value}_2(t) & \dots & \text{TI 1 value}_M(t) \\ \text{TI 2} & \text{TI 2 value}_1(t) & \text{TI 2 value}_2(t) & \dots & \text{TI 2 value}_M(t) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \text{TI X} & \text{TI X value}_1(t) & \text{TI X value}_2(t) & \dots & \text{TI X value}_M(t) \end{bmatrix}$$

Finally, some remarks on the state space. We apply Z-score normalization to the data in order to ensure that each feature and asset has a similar range of values, which is a common and straightforward method for preparing data as input for DRL models. The Z-score normalization can be described as follows: Let x be a data point, μ be the mean of the data, and σ be the standard deviation. The Z-score z for x is calculated as:

$$z = \frac{x - \mu}{\sigma}$$

This normalization process is applied separately for each feature and asset to ensure consistent scaling. Moreover, when utilizing OHLC data, we compute the relative price using open and closing prices of the same day instead of the closing prices of two consecutive days. We also use the open price to compute the relative low and high price vectors.

5.5.1.1 Action Space

As we have now extensively described the state space, let's now focus on discussing the action space. It is important to recall that the ultimate objective of our agent is to determine the optimal allocation for each asset, thereby maximizing the potential reward. Consequently, it becomes evident that the action an agent needs to take involves assigning a percentage of the portfolio's capital to each asset. As a result, an action is represented as a new weight vector prediction that dictates the allocation of capital to each assets:

$$\text{Action} = w_t = (w_1, w_2, \dots, w_M, w_{M+1}) \tag{32}$$

Here, M represents the number of assets, a value of 25 in our specific case. Notably, the vector's length is $M + 1$ due to the necessity of including cash allocation, a crucial consideration as outlined in the section on the state space. It is worth highlighting that this representation of the action state is widely adopted in related studies [9], [8]. Finally, the action vector representing the allocation of capital to each asset in the portfolio, a constraint is imposed: the sum of allocation weights must equal 1:

$$\sum_{i=1}^{M+1} w_{i,t} = 1$$

5.5.1.2 Reward

As mentioned in the Action section, the ultimate objective of our agent is to determine the optimal allocation for each asset, thereby maximizing the potential reward. Let's describe the reward function. First we need to define the share holding of asset i at time t:
- $h_{i,t}$: The shares holding of stock i at time t

$$h_{i,t} = \left\lfloor \frac{V_t \times w_{i,t}}{R_{i,t}} \right\rfloor$$

where $R_{i,t}$ is taken from the relative price vector as defined in the section about State space.
- $V_t \times w_{i,t}$ is the Portfolio Value at time t:

$$V_t = \sum_{i=1}^{M+1} h_{i,t-1} \times R_{i,t}$$

Using this this definition of the portfolio value it's easy to define the reward at time t as the difference between the portfolio value at time t and $t-1$.
- R_t : Reward at time t, defined as:

$$R_t \triangleq V_t - V_{t-1}$$

5.6 Deep Learning Framework: DDPG

In the Scientific Background section, we introduced the main concepts of DRL. Wdecided to employ the DDPG algorithm for this project. In the following section, we will provide a detailed explanation of the underlying principles of this method. Particularly, we will delve into the concepts of DQN and Actor-Critic methods, as they form the foundation of DDPG. Subsequently, we will explore the implementation details of the DDPG algorithm, including specific aspects such as the replay buffer and the Ornstein-Uhlenbeck method to address the exploration-exploitation trade-off.

5.6.1 Deep Q-Networks (DQN)

In this section we will quikcly review the principle of DQN and how it leaded to DDPG. DQN represent a significant advancement in the field of reinforcement learning, addressing the challenges posed by high-dimensional state spaces that are impractical for traditional Q-table approaches. By leveraging the representational power of deep neural networks, DQN provides a scalable solution for approximating the Q-values associated with state-action pairs in complex environments.

5.6.1.1 Background

Traditional Q-learning relies on a tabular representation, the Q-table, to store the expected future rewards for each state-action pair. However, as the state and action spaces grow, the Q-table becomes infeasible due to the curse of dimensionality. This limitation necessitated the development of function approximation techniques to estimate Q-values, leading to the introduction of DQN.

5.6.1.2 Neural Network Approximation

In DQN, a deep neural network is employed to approximate the Q-function. Mathematically, the Q-function is represented as:

$$Q(s, a; \theta)$$

Where:

- s is the state.
- a is the action.
- θ are the parameters of the neural network.

The network is designed to take the state of the environment as input and produce Q-values for all possible actions as output. This function approximation allows DQN to generalize across states, enabling it to handle environments with large or continuous state spaces.

5.6.1.3 Replay Buffer

A critical innovation in DQN is the concept of replay buffer [72]. Instead of updating the Q-values based on the most recent transition, the agent stores its experiences (state, action, reward, next state) in a replay buffer. During training, mini-batches of experiences are sampled from this buffer to update the network using the following loss function:

$$L(\theta) = \mathbb{E}_{(s,a,r,s')\sim U(D)}\left[\left(r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta)\right)^2\right]$$

Where:

- D is the replay buffer.
- $U(D)$ represents a uniform random sample from the buffer.
- γ is the discount factor.
- θ^- are the parameters of the target network.

5.6.1.4 Target Network

To further stabilize the learning process, DQN introduces the use of a target network [73]. This network is a clone of the primary Q-network but is updated less frequently. The target network is used to compute the target Q-values during the learning phase, preventing rapid oscillations in Q-value estimates.

5.6.1.5 Transition to DDPG

DQN is adept at handling environments characterized by discrete action spaces, but its efficacy diminishes in the context of continuous action domains. This shortcoming paves the way for the emergence of DDPG, an approach that melds the advantages of DQN and actor-critic designs to proficiently manage continuous action spaces.

5.6.2 Actor-Critic Methods

Actor-Critic methods are a foundational approach in RL that combine the benefits of value-based and policy-based methods. By maintaining both a policy function (the actor) and a value function (the critic), these methods aim to harness the strengths of both approaches to achieve more stable and efficient learning.

5.6.2.1 Background

In RL, there are two primary paradigms: value-based methods, which aim to learn a value function that estimates the expected return from each state or state-action pair, and policy-based methods, which directly learn a policy that maps states to actions. Actor-Critic methods bridge these paradigms by maintaining and updating both a policy and a value function.

5.6.2.2 The Actor and the Critic

1. **Actor**: The actor is responsible for selecting actions based on the current policy. Mathematically, the policy, often denoted by π, is a function that maps states to a probability distribution over actions [74]:

   $$\pi(a|s; \theta_{\text{actor}})$$

   Where θ_{actor} are the parameters of the actor network.

2. **Critic:** The critic evaluates the actions taken by the actor by estimating the value function. For Actor-Critic methods, the critic often estimates the state-value function $V(s)$ or the action-value function $Q(s, a)$, depending on the specific implementation. [75]

5.6.2.3 Learning Process

During the training phase, the interaction between the actor and critic unfolds as:

1. The actor determines an action, drawing from its prevailing policy.
2. In response, the environment provides a reward and transitions to a subsequent state.
3. The critic assesses the action taken and calculates the temporal difference (TD) discrepancy.
4. Guided by the feedback from the critic, the actor refines its policy parameters to enhance the anticipated reward.
5. Using the TD discrepancy, the critic refines its value predictions.

The synergy between the actor and the critic ensures that the policy improves in the direction of increasing expected returns, while the value estimates become more accurate over time.

5.6.2.4 Advantages and Transition to DDPG

Actor-Critic methods offer several advantages:

- They combine the strengths of value-based and policy-based methods.
- They can handle both discrete and continuous action spaces.
- The critic's feedback helps reduce the variance of policy updates.

From the description of Actor-Critic methods and DQN, it is quite natural to see that combining the two methods should lead to a solution particularrly adpated for PO problem as on could use continuous action and state space and leverage the characteristics of DQN.

5.6.3 Introduction to Deep Deterministic Policy Gradients (DDPG)

DDPG stands as a cornerstone technique within reinforcement learning, seamlessly merging the strengths of DQN with those of Actor-Critic approaches. Specifically tailored for continuous action spaces, DDPG amalgamates the strengths of both its forerunners to offer a potent solution for environments where actions are represented as real-valued vectors [76].

DDPG was conceived by Google DeepMind to address the challenges posed by continuous action spaces [77, 78]. It is fundamentally a policy gradient-based algorithm. While it employs a stochastic behavior policy for exploration, it estimates a deterministic target policy—a feature that, as highlighted by Lillicrap et al. [77], streamlines the learning process. This deterministic approach was juxtaposed against its stochastic counterpart using standard reinforcement learning benchmark environments, underscoring its efficacy [78].

Essentially, the DDPG method simultaneously updates both a Q-function and a policy. It uses off-policy data combined with the Bellman equation to improve the Q-function. Subsequently, this Q-function is harnessed to optimize the policy. This dual learning mechanism echoes the principles of Actor-Critic methods, where the policy (actor) and value function (critic) are synergistically updated.

In continuous action domains, the action space is delineated by real-valued vectors [76]. Traditional Q-learning methodologies grapple with intractability in such scenarios, especially when the action space is expansive. This is attributed to the "curse of dimensionality", a phenomenon where, given a fixed set of training samples, the predictive prowess of a model escalates with the number of dimensions or features initially but wanes beyond a certain threshold [79, 80]. Moreover, Q-learning is susceptible to overestimation bias, where the maximization of a noisy value estimate culminates in a consistent overestimation of Q-values [81].

DDPG and its derivatives have garnered significant attention in recent literature, especially for problems situated in continuous domains.

5.6.4 The DDPG Algorithm

As mentioned earlier, DDPG is an off-policy algorithm that marries the concepts of value-based and policy-based reinforcement learning, specifically tailored for environments with continuous action spaces. The primary objective of DDPG is to optimize a policy in such a way that expected future rewards are maximized.

The core objective in DDPG is to maximize the expected reward from the environment. Given a policy $\mu(s|\theta^\mu)$ parameterized by θ^μ, the objective can be defined as:

$$J(\theta^\mu) = \mathbb{E}_{s\sim\rho^\beta, a\sim\mu}[Q(s, a|\theta^Q)]$$

Where:

- $Q(s, a|\theta^Q)$ is the action-value function approximated by the critic network.
- ρ^β is the behavior distribution over states.

DDPG employs a critic network to approximate the Q-function. The critic is trained to minimize the mean squared Bellman error. Given a sampled transition (s_t, a_t, r_t, s_{t+1}), the target for the critic is:

$$y_t = r_t + \gamma Q'(s_{t+1}, \mu'(s_{t+1}|\theta^{\mu'})|\theta^{Q'})$$

Where Q' and μ' are the target networks for the critic and actor, respectively.

The actor network in DDPG is trained to maximize the expected Q-values. The gradient used for updating the actor's parameters is:

$$\nabla_{\theta^\mu} J \approx \mathbb{E}_{s\sim\rho^\beta}[\nabla_a Q(s, a|\theta^Q)|_{a=\mu(s)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)]$$

This gradient points in the direction that maximizes the expected reward. A good representation of the DDPG algorithm is given in 1, taken from [77], [78]. Moreover, you can find a schematic representation of how DDPG work within his envronment in Fig 10

Algorithm 1 Deep Deterministic Policy Gradient Algorithm from [77], [78]

Initialize critic network $Q(s, a|\theta^Q)$ and actor network $\mu(s|\theta^\mu)$ with random weights
Initialize target networks Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1 to M **do**
    Initialize a random process for action exploration
    Receive initial observation state s_1
    for t = 1 to T **do**
        Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}$ according to the current policy and exploration noise
        Execute action a_t and observe reward r_t and new state s_{t+1}
        Store transition (s_t, a_t, r_t, s_{t+1}) in R
        Sample a random mini-batch 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 network by minimizing 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:
$\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s_i}$
        Update the target networks:
$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$, $\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$
    end for
end for
return learned policy $\mu(s|\theta^\mu)$

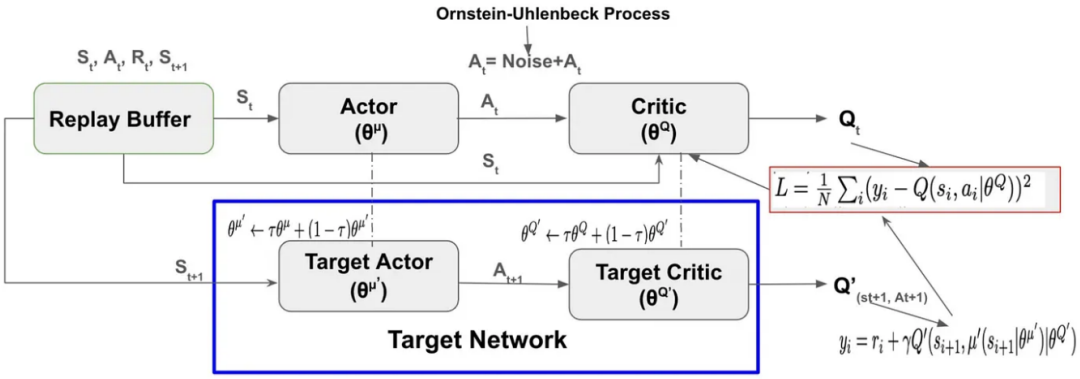

Figure 10: DDPG Architecture [3]

When examining Figure 10 closely, two elements that have not been discussed yet become apparent: the replay buffer of DDPG and the Ornstein-Uhlenbeck method. In the following subsections, we will elaborate on their significance and provide implementation details. We will aslo give more details on Target Networks role in DDPG.

5.6.5 Target Network in DDPG

One of the innovative features of DDPG, borrowed from its predecessor DQN, is the use of Target Networks. These networks are crucial for stabilizing the learning process, especially when function approximators like neural networks are employed. In DDPG, there are two primary networks: the Actor and the Critic. Correspondingly, there are two Target Networks: the Target Actor and the Target Critic. These Target Networks are copies of the primary networks but are updated less frequently, ensuring that the learning targets do not oscillate wildly. From a mathematical perspective, the refresh mechanism for the Target Networks employs a soft

update. This method amalgamates the weights of the main network with the weights of the Target Network. Given the primary network weights θ and the Target Network weights θ', the soft update rule is:

$$\theta' \leftarrow \tau\theta + (1 - \tau)\theta'$$

Where τ is a small constant, typically close to zero. This ensures that the Target Network changes slowly over time, inheriting the stability properties of the primary network while avoiding rapid oscillations.

The primary utility of the Target Networks in DDPG is in the computation of the target Q-values for the Critic's update. Given a transition (S_t, A_t, R_t, S_{t+1}), the target Q-value, computed using the Target Critic and Target Actor, is:

$$y_t = R_t + \gamma Q'(S_{t+1}, \mu'(S_{t+1}|\theta^{\mu'})|\theta^{Q'})$$

Here, Q' represents the Target Critic, and μ' represents the Target Actor. By using the Target Networks to compute this target, DDPG ensures that the update is based on a stable and consistent target, mitigating the risk of divergence.

In summary, Target Networks play a pivotal role in the stability and convergence of DDPG. By providing consistent and slowly changing targets for learning, they prevent harmful oscillations and ensure a smoother learning trajectory.

5.6.6 Replay Buffer

In RL, especially in algorithms like DDPG that operate in continuous action spaces, the sequential nature of experience can introduce strong temporal correlations. These correlations can hinder the learning process, leading to suboptimal policies and even divergence. The Replay Buffer, a fundamental component of DDPG, addresses this challenge by providing a mechanism to store and sample experiences in a manner that breaks these temporal correlations. The Replay Buffer acts as a limited-capacity storage, preserving previous transitions. It captures the state, action, reward, and the following state, represented as (S_t, A_t, R_t, S_{t+1}) [77]. As the agent interacts with the environment, these transitions are stored in the buffer. Once the buffer reaches its maximum capacity, older experiences are overwritten by newer ones, ensuring a continuous influx of recent experiences.By design, the Replay Buffer decouples temporally adjacent experiences. When updating the policy or value function, the DDPG algorithm samples a mini-batch of experiences uniformly from the buffer. This sampling strategy ensures that the experiences used for learning are decorrelated, mitigating the adverse effects of sequential data. DDPG operates as an off-policy method, implying it derives knowledge from experiences not strictly originating from the prevailing policy. The Replay Buffer plays a pivotal role in this context, facilitating the agent's ability to reflect on and assimilate lessons from historical experiences, regardless of the policy under which they were produced. This property is particularly beneficial in environments where valuable experiences are rare, as it ensures that no critical learning opportunity is missed. In many RL scenarios, interacting with the environment can be expensive or time-consuming. TThe Replay Buffer mitigates this issue by preserving historical experiences, enabling the agent to derive insights from them on repeated occasions. This reuse of data significantly boosts the sample efficiency of the algorithm, enabling more effective learning from a limited number of interactions with the environment. The Replay Buffer plays a pivotal role in the stability and convergence properties of DDPG. By breaking temporal correlations, enabling off-policy learning, and enhancing sample efficiency, the Replay Buffer ensures that the learning process is both stable and efficient. In the context of complex tasks, such as portfolio optimization, where the

state and action spaces can be vast and the dynamics intricate, the Replay Buffer's contribution to the robustness of DDPG cannot be overstated [82].

5.6.7 Exploration-Exploitation Trade-off using Ornstein-Uhlenbeck method

When working with RL methods, particularly in algorithms like DDPG that operate within continuous action spaces, the balance between exploration and exploitation is of paramount importance. In order to force the exploration behaviour of our model, noise is added to the actions at training time. Gaussian noise, with its time-independent characteristics, can lead to erratic exploration patterns. In contrast, the Ornstein-Uhlenbeck Action Noise (OU noise) provides temporally correlated fluctuations, ensuring smoother and more consistent exploration trajectories. This autocorrelation in OU noise, as opposed to the time-independent Gaussian noise, facilitates a more coherent exploration of the action space, enhancing the learning process. Adding noise during training, especially using the Ornstein-Uhlenbeck method, has been recognized as an effective strategy for policy exploration, as evidenced in seminal works [77], [62]. Let's describe quickly the OU noise. The OU process is a stochastic process that describes the velocity of a Brownian particle under the influence of friction. Mathematically, it is defined by the stochastic differential equation:

$$dX_t = \theta(\mu - X_t)dt + \sigma dW_t$$

Where:

- X_t is the process value at time t.
- θ is the rate of mean reversion.
- μ is the equilibrium or mean value.
- σ is the volatility.
- dW_t is a Wiener process or Brownian motion.

As mentioned and one can observe, the key property of the OU noise is its temporally correlated nature, which means that its value at a given time is influenced by its previous values. This is in stark contrast to Gaussian noise, which is time-independent.

5.6.7.1 Key parameters of the DDPG Framework

Within this section, we will provide a concise summary of the hyperparameters associated with the implementation of the DDPG framework. It is essential to highlight the sensitivity of DDPG to modifications in hyperparameters, prompting us to adopt values that have been established as effective in pertinent research [77]. Detailed descriptions and values of the crucial parameters for our DDPG Framework can be located in Table 3.

DDPG Framework's parameters	Description	Value
General hyperparameters		
Value function threshold	Reward threshold for early stopping	2
Window length	Window or observation size	$3, 5, 7, 9$, and 11
Number of episode	agent-environment interactions from initial to final states	400
Max Step	Number of steps completed in episode	1000
Replay Buffer		
Batch size	Mini-batch size during training	64
Buffer size	Size of replay buffer	100000
Actor-Critic		
τ (tau)	Target network update ratio	0.001
γ (gamma)	Reward discounting factor	0.99
Critic learning rate (β)	Critic learning rate	0.001
Actor learning rate (α)	Actor learning rate	0.0001
Ornstein-Uhlenbeck Noise		
θ (theta)	rate of mean reversion	0.15
σ (sigma)	volatility - std	0.2
μ (mu)	mean value (vector of 0 for initialization	0

Table 3: Fundamental Hyperparameters of the DDPG Framework implementation

Let's discuss some parameters. Firstly, the Value Function Threshold is utilized as follows: if the reward exceeds the threshold for 10 consecutive episodes, the training is halted, regardless of the number of episodes executed. This mechanism resembles early stopping in Deep Learning, preventing overfitting when specific behavior is observed. Next, let's delve into the window length parameter. This parameter significantly impacts the DDPG process. Notably, as we employ a convolutional network in certain experiments, the minimum window length must be 3 to allow convolution over the time dimension within the convolution layer. We explored window lengths from 3 to 11 open market days, enabling us to analyze both short-term and long-term trends that may benefit our models. Additionally, varied window lengths facilitate evaluating model generalization, among other factors. Moving on to the number of episodes, determining an ideal value proved challenging due to the diverse experimentation involving different DNN architectures and state spaces. To avoid overfitting, a careful selection was crucial. After evaluating values from various literature sources, we decided to set the number of episodes to a constant value of 400 for all experiments. This decision was made to maintain consistency across models, state spaces, and window lengths, although fine-tuning this parameter per scenario remains an avenue for future implementation. Other parameters value were taken as mentioned to original paper introducing this methods within our context or official implementation of the algorithm (ex: Ornstein-Uhlenbeck Noise parameter's value).

5.7 Actor-Critic: Deep Neural Networks Architectures

As described previously, DDPG stands as a sophisticated reinforcement learning technique, merging the merits of actor-critic approaches with deep neural network capabilities. In the realm of financial portfolio optimization, its adaptability and precision have made it an appealing choice. At the heart of DDPG lie four neural networks:

- **Actor Network**: Maps states to actions. This network takes on the critical task of determining the optimal asset allocation strategy for a portfolio at any given state.
- **Target Actor Network**: Acts as a stable baseline for the Actor, assisting in smoother policy updates and mitigating rapid fluctuations in predictions.
- **Critic Network**: Evaluates the value of state-action pairs. By providing feedback on the quality of actions chosen by the Actor, it guides the policy optimization process.
- **Target Critic Network**: Analogous to the Target Actor, this network serves to stabilize the value updates in the Critic network.

While these networks serve distinct roles, their architecture foundation remains consistent. The central hypothesis of our research posits that the underlying network architecture significantly influences the performance of the DDPG method in portfolio optimization tasks.

Both the target networks (actor and critic) mirror the architecture of their original counterparts, so for simplicity, we often refer to them collectively. However, the Actor and Critic networks diverge in certain aspects. The Actor, focusing on policy determination, employs a softmax activation function in its output layer to yield a probability distribution over asset allocations. Conversely, the Critic, evaluating state-action value, uses a linear activation function to predict the expected return or value of a given state-action pair.

It's noteworthy that while the primary structure (be it CNN, LSTM, or CRNN) remains consistent, the input/output layers are adapted to respect each network's objective. Inputs can vary based on the chosen state space—whether it's closing prices, OHLC data, or technical indicators—but the core architecture remains unchanged.

In our comprehensive study, we delve into three primary architectures: CNN, LSTM, and CRNN. Each of these has been explored in its standard form and an optimized variant, aiming to discern the most effective structure for our use case.

In the subsequent sections, we will dissect each architecture in detail, elucidating the key components, their inherent advantages, and potential drawbacks. This deep dive aims to provide clarity on the significance of architecture choice in the DDPG-based portfolio optimization process.

5.7.1 Convolutional Neural Networks (CNN)

5.7.1.1 Key Elements of CNNs

Convolutional Neural Networks have emerged as a cornerstone in the field of deep learning, particularly for tasks related to image processing and computer vision. A CNN typically has three main layers: a convolutional layer, a pooling layer, and a fully connected layer as depicted in Figure 11. Each layer has its distinct role and contributes to the network's ability to automatically and adaptively learn spatial hierarchies of features [83].

1. **Convolutional Layer**: This is the primary building block of a CNN. The layer's name is derived from the "convolution" operation it performs, which is essentially a mathematical combination of two functions, producing a third function. In the context of a CNN, this involves applying a filter or kernel to an input to produce a feature map, capturing the spatial relationships in the image as illustrated in Figure 12. Given an input feature map I of size $W \times H$ (width by height) and a filter K of size $F \times F$ (also known as a kernel), the convolution operation is defined as:

$$O_{(x,y)} = \sum_{i=1}^{F} \sum_{j=1}^{F} I_{(x+i,y+j)} \cdot K_{(i,j)} \tag{33}$$

   Where:

   - O is the output feature map.
   - $O_{(x,y)}$ is the value of the output feature map at position (x, y).
   - The summations run over the spatial dimensions of the filter.

This operation slides the filter K over the input image I in both horizontal and vertical directions, computing the dot product between the entries of K and the corresponding entries of I at each position.

2. **Pooling Layer**: Often placed after the convolutional layer, the pooling layer serves to reduce the spatial dimensions of the feature maps, thereby reducing the number of parameters and computations in the network. This layer helps in making the network invariant to small transformations, rotations, and translations in the input image as illustrated in Figure 13. The most common type of pooling is max pooling, where the maximum value is taken from a set of values in the feature map. For a given input feature map I and a pooling size $P \times P$, the max pooling operation is defined as:

$$O_{(x,y)} = \max\{I_{(a,b)} : x \leq a < x + P, y \leq b < y + P\} \tag{34}$$

Where $O_{(x,y)}$ is the maximum value in the $P \times P$ region of I starting from the top-left corner (x, y).

3. **Fully Connected Layer**: Following multiple convolutional and pooling stages, the neural network's advanced decision-making takes place in the fully connected layer. In this layer, neurons establish connections with all activations from the prior layer, mirroring the behavior in standard artificial neural networks. This segment is tasked with categorizing the features discerned by earlier layers and rendering the ultimate prediction.

The design of CNNs was inspired by the discovery of the visual mechanism of the animal cerebral cortex, and they have since been pivotal in achieving state-of-the-art results in various machine learning challenges [84]. Also CNNs were designed to work with images, they can be used in other context, and remembering that our state space may be seen as a matrix which can itself be seen as an image, it makes sense to use CNN for our task.

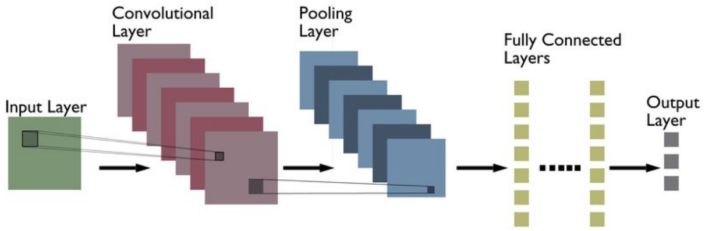

Figure 11: Schema of a CNN architecture (source: vitalflux)

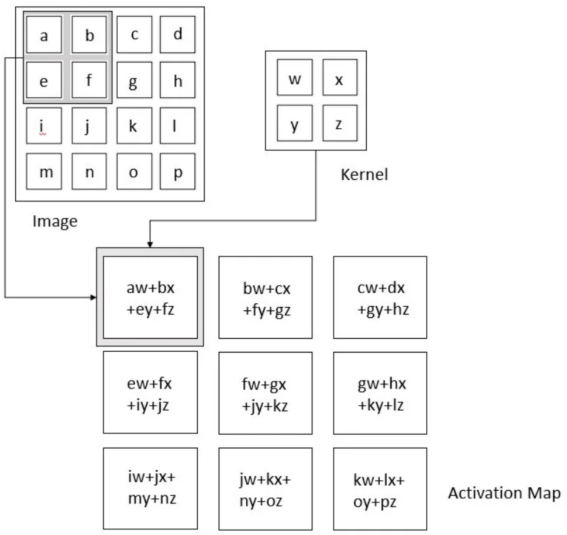

Figure 12: Convolution Operation [4]

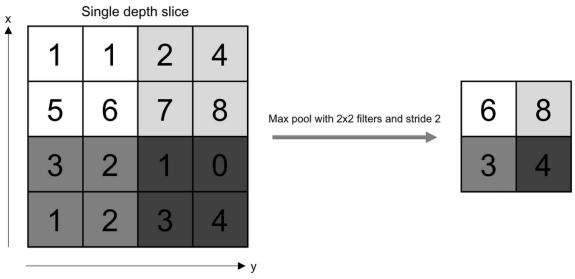

Figure 13: Pooling Operation (source: O'Reilly Media

5.7.1.2 Advantages and Drawbacks

Let's now discuss the main advantages and drawbacks of CNNs in our context.

Advantages

1. **Feature Extraction from Financial Data**: CNNs excel at capturing hierarchical features in images, and this capability extends to financial time series data. They can learn relevant patterns and trends from historical price movements, helping the DDPG agent to discern valuable signals for making allocation decisions.

2. **Spatial Relationships in Data**: Portfolio optimization involves analyzing the interdependencies between different assets over time. CNNs can effectively capture the spatial relationships in these historical trends, enabling the agent to exploit intricate patterns and correlations, which are crucial for informed investment strategies.

3. **Translation Invariance**: Financial markets exhibit complex and dynamic behavior. CNNs' translation invariance property allows them to identify valuable patterns despite small temporal shifts in asset data. This characteristic is particularly advantageous for handling volatile market conditions.

Drawbacks

1. **Limited Contextual Information**: While CNNs excel at capturing local patterns, they might struggle to grasp broader macroeconomic or global factors that influence asset prices. Portfolio optimization benefits from understanding contextual information beyond the individual asset level, which could potentially limit CNNs' effectiveness.

2. **Complexity and Overfitting**: Deep architectures, including CNNs, are prone to overfitting when dealing with relatively small financial datasets. The complexity of CNNs might lead to learning noise instead of signal, which could negatively impact the robustness of the learned allocation policies.

3. **Limited Time Series Exploitation**: CNNs do not fully leverage the sequential nature of time series data. They process each time step independently, potentially missing out on important temporal dependencies that are crucial in financial markets.

4. **Feature Relevance**: CNNs might generate complex features that are difficult to interpret in financial contexts. Understanding the rationale behind an allocation decision is crucial for risk management and regulatory compliance, and overly intricate features could hinder the interpretability of the model.

In conclusion, CNNs offer powerful tools for extracting meaningful features from financial time series data, making them suitable for enhancing portfolio optimization with DDPG. However, careful consideration must be given to their suitability for capturing both local and global patterns in financial markets, as well as addressing challenges related to model complexity, interpretability, and the full exploitation of time series information.

5.7.1.3 Our Architectures

As previously mentioned, our approach involves employing two distinct CNN architectures: a standard configuration and an optimized version. In our discourse, we refrain from distinguishing between the normal and target networks for the sake of clarity. Also, both the critic and actor networks share the same architecture, excluding the input and output layers as explained. Therfore we will describe the actor networks only. This approach allows us to maintain conciseness in our presentation. Furthermore, we have customized the input and output configurations to accommodate varying window lengths and datasets used. These adaptations are crucial due to change of state space according to these parameters. Consequently, the forthcoming description is presented in a generic manner. For specific architectural details, one can refer to the code, which incorporates the parameters outlined above.

Layer Type	**Detail**	**Standard CNN Configuration**	**Optimized CNN Configuration**
Convolutional Layer 1	Filters	32 of size (1, 3)	32 of size (1, 3)
	Activation	ReLU	**ELU**
	Additional	Batch normalization	**Dropout (keep rate: 80%)**, **L2 Regularization**
Convolutional Layer 2	Filters	32 of size (1, window_length - 2)	32 of size (1, window_length - 2)
	Activation	ReLU	**ELU**
	Additional	Batch normalization	**Dropout (keep rate: 80%)**, Batch normalization, **L2 Regularization**
Flattening Layer	Operation	Flattens feature map	Flattens feature map
Fully Connected Layer 1	Neurons	64	64
	Activation	ReLU	ReLU
	Additional	Batch normalization	Batch normalization
Fully Connected Layer 2	Neurons	64	64
	Activation	ReLU	ReLU
	Additional	Batch normalization	Batch normalization
Output Layer	Neurons	Number of assets	Number of assets
	Activation	Softmax	Softmax
	Weight Initialization	Uniform[-3e-3, 3e-3]	Uniform[-3e-3, 3e-3]

Table 4: Detailed comparison between the layers of the Standard CNN and the Optimized CNN.

Both models utilize the Adam Optimizer for the optimization process. The differences between the two models lie mainly in the activation functions, the presence of dropout, and regularization techniques in the convolutional layers of the optimized CNN.

5.7.2 Long-Short-Term-Memory (LSTM)

5.7.2.1 Key Elements of LSTMs

LSTM architectures, a variant of Recurrent Neural Network (RNN), have emerged as a cornerstone in the realm of deep learning, especially for tasks that involve sequential data such as time series forecasting, natural language processing, and, in our context, financial time series for portfolio optimization. An LSTM is designed to remember patterns over long durations and is particularly suited for tasks where the temporal dynamics and dependencies between the steps are crucial. The main components of an LSTM cell include the input gate, forget gate, output gate, and cell state, as illustrated in Figure 14. These components allow the LSTM to regulate the flow of information through the cell [85, 86].

1. **Input Gate**: Decides the extent of new data to be retained in the cell state. A sigmoid activation function is employed to make this decision, where a value of 0 indicates "no passage" and a value of 1 signifies "full passage".

2. **Forget Gate**: Determines which parts of the cell state to retain or discard. A sigmoid activation function aids in this decision-making.

3. **Cell State**: Acts as the LSTM cell's repository. The input and forget gates modulate the addition or removal of information from this state.

4. **Output Gate**: Given the cell state and input, it establishes the subsequent hidden state. This resultant state aids in making predictions and is relayed to the succeeding LSTM cell.

LSTMs have been instrumental in achieving state-of-the-art results in various sequential tasks due to their ability to capture long-term dependencies without suffering from the vanishing gradient problem, which plagues traditional RNNs [4]. Given the sequential nature of financial data, LSTMs are naturally suited for tasks such as portfolio optimization in the context of Deep Reinforcement Learning.

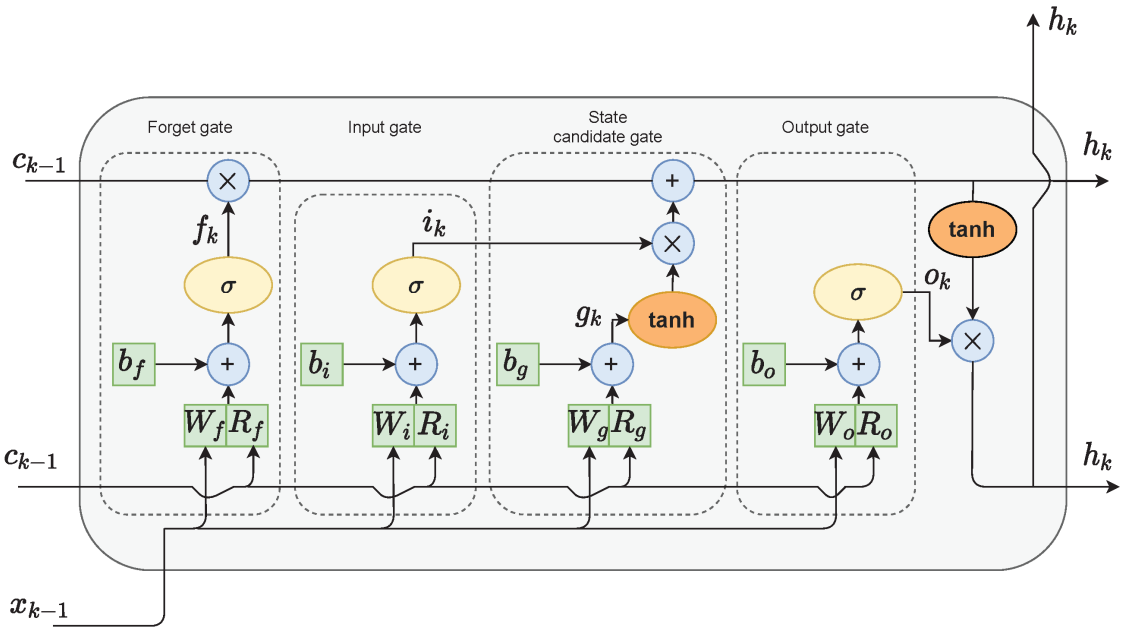

Figure 14: Schema of an LSTM cell architecture [5]

As stated in the Scientific Background section, the LSTM cell regulate the flow of information with the following equation:

$$
\begin{aligned}
i_t &= \sigma(W_{ii}x_t + b_{ii} + W_{hi}h_{(t-1)} + b_{hi}),\\
f_t &= \sigma(W_{if}x_t + b_{if} + W_{hf}h_{(t-1)} + b_{hf}),\\
g_t &= \tanh(W_{ig}x_t + b_{ig} + W_{hg}h_{(t-1)} + b_{hg}),\\
o_t &= \sigma(W_{io}x_t + b_{io} + W_{ho}h_{(t-1)} + b_{ho}),\\
c_t &= f_t * c_{(t-1)} + i_t * g_t,\\
h_t &= o_t * \tanh(c_t),
\end{aligned}
\tag{35}
$$

where i_t, f_t, g_t, o_t, c_t, and h_t denote the input gate, forget gate, cell input, output gate, cell state, and hidden state, respectively, at time t. Note that notations are slighty different than the ones in Figure **??**, don't get confused.

5.7.2.2 Advantages and Drawbacks

Let's delve into the primary advantages and drawbacks of LSTMs in the context of Deep Reinforcement Learning for Portfolio Optimization.

Advantages

1. **Handling of Sequential Data**: LSTMs are explicitly designed to handle time series data, making them adept at capturing temporal dependencies in financial datasets, which is crucial for predicting asset prices and making allocation decisions.

2. **Long-term Dependencies**: LSTMs can remember patterns over extended periods, allowing them to capture long-term trends in financial markets, which can be instrumental for portfolio optimization.

3. **Addressing the Vanishing Gradient Dilemma**: Conventional RNNs are plagued by the vanishing gradient dilemma, leading to a loss of long-term data. LSTMs, owing to their distinct design, counteract this challenge, guaranteeing that pertinent data persists through temporal intervals [85].

Drawbacks

1. **Computational Complexity**: LSTMs, with their multiple gates and operations, can be computationally intensive, especially when dealing with large financial datasets. This can lead to longer training times and increased computational costs.

2. **Overfitting on Small Datasets**: Like other deep learning architectures, LSTMs can overfit, especially when the available financial data is limited. Regularization techniques might be required to prevent this.

3. **Difficulty in Interpretability**: Understanding the decisions made by LSTMs can be challenging, given their complex internal dynamics. This can be a concern when interpretability is crucial for risk management and regulatory compliance in financial applications.

In conclusion, LSTMs present a compelling choice for portfolio optimization tasks in the realm of Deep Reinforcement Learning due to their proficiency in handling sequential data and capturing long-term dependencies. However, their computational complexity and potential for overfitting necessitate careful tuning and consideration when deploying them in real-world financial scenarios.

5.7.2.3 Our Architectures

All assumptions made in Section 5.7.1.3 are also valide here.

Layer Type	**Detail**	**Standard LSTM**	**Optimized LSTM**
		Configuration	**Configuration**
Input Reshape	Shape	[-1, window_length, 1]	[-1, window_length, 1]
LSTM Layer 1	Hidden Dimension	32	32
	Return Sequence	No	**Yes**
	Additional	-	**Dropout (keep rate: 80%)**
LSTM Layer 2	Hidden Dimension	-	**32**
	Additional	-	**Dropout (keep rate: 80%)**
Reshape	Shape	[-1, num_stocks, hidden_dim]	[-1, num_stocks, hidden_dim]
Flatten	Operation	Flattens feature map	Flattens feature map
Fully Connected Layer 1	Neurons	64	64
	Activation	ReLU	ReLU
	Additional	Batch normalization	Batch normalization
Fully Connected Layer 2	Neurons	64	64
	Activation	ReLU	ReLU
	Additional	Batch normalization	Batch normalization
Output Layer	Neurons	Number of assets	Number of assets
	Activation	Softmax	Softmax
	Weight Initialization	Uniform[-3e-3, 3e-3]	Uniform[-3e-3, 3e-3]

Table 5: Detailed comparison between the layers of the Standard LSTM and the Optimized LSTM.

Both models utilize the Adam Optimizer for the optimization process. The differences between the two models are highlighted in bold.

5.7.3 Convolutional Recurrent Neural Network (CRNN) with Gated Reccurent Unit (GRU)

5.7.3.1 Key Element of CRNNs with GRU

Convolutional Recurrent Neural Networks (CRNN) combine the spatial feature extraction capabilities of Convolutional Neural Network (CNN) with the sequential processing strengths of RNN. In the context of Deep Reinforcement Learning for Portfolio Optimization, CRNNs can be particularly effective as they can capture both spatial relationships in financial data and temporal dependencies across time series. A typical CRNN for our context would employ convolutional layers for feature extraction followed by Gated Recurrent Unit (GRU) layers for sequential processing, as depicted in Figure 16.

1. **Convolutional Layers**: As elaborated in the CNN segment, these layers play a pivotal role in discerning hierarchical spatial characteristics from the provided input. They capture the intricate patterns and relationships in financial time series data, preparing it for sequential processing.

2. **GRU Layers**: GRU are a type of RNN architecture that has proven to be effective in

capturing long-term dependencies without being as computationally intensive as LSTMs. A GRU has two gates: a reset gate and an update gate. These gates determine how the unit adapts to the hidden state and the new input [6, 87]. The GRU layers in a CRNN process the features extracted by the convolutional layers in a sequential manner, capturing the temporal dynamics of the financial data. It is illustrated in Figure 15

The fusion of CNNs and GRUs in a CRNN architecture allows for a comprehensive analysis of financial data, making it a promising approach for tasks like portfolio optimization in the realm of Deep Reinforcement Learning.

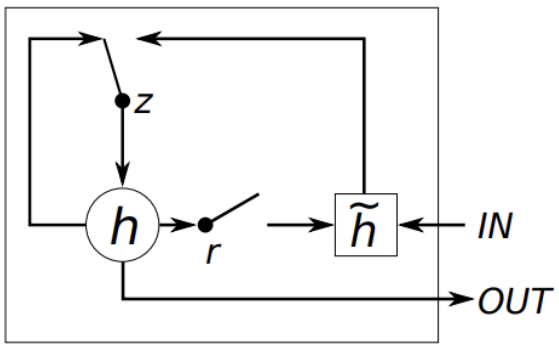

Figure 15: Gated Recurrent Unit (GRU) [6]

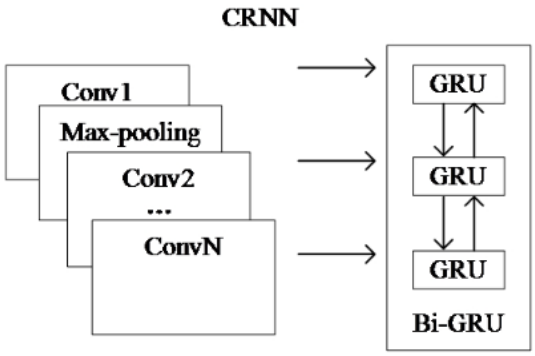

Figure 16: Schema of a basic CRNN architecture [7]

The GRU is a simplified version of the LSTM cell, and it combines the forget and input gates into a single "update gate." Here's the mathematical formulation for the GRU:

$$\begin{aligned} z_t &= \sigma(W_z x_t + U_z h_{(t-1)} + b_z), \\ r_t &= \sigma(W_r x_t + U_r h_{(t-1)} + b_r), \\ \tilde{h}_t &= \tanh(W x_t + U(r_t \odot h_{(t-1)}) + b), \\ h_t &= (1 - z_t) \odot h_{(t-1)} + z_t \odot \tilde{h}_t, \end{aligned} \tag{36}$$

Where:

- z_t is the update gate. It decides how much of the previous hidden state $h_{(t-1)}$ should be updated with the new candidate hidden state $\tilde{h}_t$.
- r_t is the reset gate. It determines how much of the previous hidden state should be forgotten.
- $\tilde{h}_t$ is the candidate hidden state. It's a combination of the input at the current time step x_t and the previous hidden state $h_{(t-1)}$, modulated by the reset gate.
- h_t is the hidden state at time t.
- $\odot$ denotes element-wise multiplication.
- W_z, U_z, b_z, W_r, U_r, b_r, W, U, and b are the weight matrices and bias terms for the update gate, reset gate, and candidate hidden state, respectively.

The GRU uses the update gate z_t to control the extent to which the previous hidden state $h_{(t-1)}$ should be updated to the new candidate hidden state $\tilde{h}_t$. The reset gate r_t allows the model to forget the previously computed state, providing a mechanism to capture shorter-term dependencies in the data.

5.7.3.2 Advantages and Drawbacks

Building upon the advantages and drawbacks discussed for CNNs, let's explore the specific merits and challenges of CRNNs with GRU layers in our context.

Advantages

1. **Holistic Feature Extraction**: CRNNs leverage the strengths of both CNNs and RNNs. They can extract spatial features from financial data and process them sequentially, ensuring a comprehensive understanding of the data.

2. **Efficiency of GRUs**: GRUs, compared to LSTMs, have fewer parameters, making them computationally more efficient while still effectively capturing long-term dependencies [87].

3. **Adaptability**: The combined architecture allows CRNNs to adapt to various patterns in the data, be it spatial hierarchies or temporal sequences, making them versatile for different financial scenarios.

Drawbacks

1. **Model Complexity**: While CRNNs are powerful, they also introduce added complexity due to the fusion of two architectures. This can lead to longer training times and might require more data for effective training.

2. **Potential for Overfitting**: The increased complexity also introduces a higher risk of overfitting, especially when the available financial data is limited.

3. **Interpretability Challenges**: As with CNNs and RNNs individually, understanding the decisions made by a CRNN can be challenging due to the intricacies of both spatial and temporal processing.

In conclusion, CRNNs with GRU present a robust and comprehensive approach for portfolio optimization tasks in Deep Reinforcement Learning. They harness the strengths of both CNNs and RNNs, ensuring a holistic analysis of financial data. However, their complexity necessitates careful tuning and consideration for effective deployment.

5.7.3.3 Our Architectures

All assumptions made in Section 5.7.1.3 are also valide here.

Layer Type	**Detail**	**Standard CRNN**	**Optimized CRNN**
		Configuration	**Configuration**
Convolutional Layer 1	Filters	32 of size (1, 3)	32 of size (1, 3)
	Activation	ReLU	**ELU**
	Additional	Batch normalization	**Dropout (keep rate: 80%)**, **L2 Regularization**
Convolutional Layer 2	Filters	32 of size (1, window_length - 2)	32 of size (1, window_length - 2)
	Activation	ReLU	**ELU**
	Additional	Batch normalization	**Dropout (keep rate: 80%)**, Batch normalization, **L2 Regularization**
Reshape	Shape	[-1, 26, 32]	[-1, 26, 32]
GRU Layer 1	Hidden Dimension	32	32
	Return Sequence	No	**Yes**
	Additional	-	**Dropout (keep rate: 80%)**
GRU Layer 2	Hidden Dimension	-	32
	Additional	-	**keep rate: Dropout (80%)**
Flatten	Operation	Flattens feature map	Flattens feature map
Fully Connected Layer 1	Neurons	64	64
	Activation	ReLU	ReLU
	Additional	Batch normalization	Batch normalization
Fully Connected Layer 2	Neurons	64	64
	Activation	ReLU	ReLU
	Additional	Batch normalization	Batch normalization
Output Layer	Neurons	Number of assets	Number of assets
	Activation	Softmax	Softmax
	Weight Initialization	Uniform[-3e-3, 3e-3]	Uniform[-3e-3, 3e-3]

Table 6: Detailed comparison between the layers of the Standard CRNN and the Optimized CRNN.

Both models utilize the Adam Optimizer for the optimization process. The differences between the two models are highlighted in bold.

5.8 Basic DDPG Framework Experiment (BDDPG)

Our initial experiment aims to develop a straightforward and effective DDPG framework and assess its performance in comparison to the various benchmarks discussed in the dedicated section. This experiment serves as the foundation for the following experiments. Firstly, because the following experiments will be built upon the DDPG framework developed in this initial experiment. Secondly, the results obtained from this experiment will serve as the baseline for evaluating the outcomes of the following experiments. The fundamental aspects of this experiment are as follows:

1. Dataset / State space: We utilize "Dataset 1", comprising solely closing prices of the assets.

2. Actor-Critic DNN architecture: We employ a simple CNN as the Actor-Critic networks architecture.

3. Evaluation Criteria: The performance of the framework is assessed based on the following metrics: Final portfolio value, Sharpe ratio, Sortino ratio, Maximum drawdown, and Average daily yields.

By conducting this preliminary experiment, we aim to establish a solid foundation for the subsequent experiments and gain valuable insights into the DDPG framework's efficacy for portfolio optimization. The performance evaluation based on the selected criteria will provide meaningful guidance for the design and assessment of the following experiments.

5.9 Deep Neural Network for Actor-Critic Networks Experiment (DNN-AC)

The primary objective of our second experiment is to assess the impact of different DNN architectures used within the actor-critic networks on the overall performance of the DDPG framework. The role of actor-critic networks as function approximators is crucial, leading us to suspect that the choice of DNN architecture significantly influences the DDPG framework's outcomes. The experiment's fundamental aspects are as follows:

1. Dataset / State space: We continue to use "Dataset 1," which includes solely the closing prices of the assets.

2. Actor-Critic DNN architecture: We evaluate five new architectures, namely LSTM (standard and optimized), CNN (optimized), and CRNN (standard and optimized).

We anticipate the following observations from this experiment: LSTM-based models are expected to outperform CNN-based models, given LSTM's proven superiority in handling time-series data, while CNNs are typically used for image-related tasks. The CRNN, which combines both CNN and LSTM mechanisms, is expected to perform better than the CNN-based model but fall short of the LSTM-based model's performance. The difference between the standard and optimized versions of each architecture primarily lies in their complexity, among other modifications such as activation function choice or hyperparameters. As a result, we expect optimized models to perform better in specific scenarios. However, we must exercise caution as increased complexity may lead to potential overfitting or reduced generalization of the model. The motivation behind this experiment is twofold. First, we aim to demonstrate the significant impact of DNN architecture choice within the DDPG framework, emphasizing the importance of careful selection and optimization. Second, we intend to highlight that although LSTM architectures are better suited for time-series data than CNNs, adding a recurrent layer to a CNN could enhance its performance when dealing with time-series data. Additionally, as the network constitutes the most substantial part of the DDPG model, it is of great interest to study and fine-tune it to achieve improved performance. While other DRL models such as TD3 are available, we choose to work with DDPG and fine-tune it to our specific needs to demonstrate the significance of architecture choice and optimization within this context. By conducting this experiment, we aim to gain deeper insights into the influence of DNN architectures on the DDPG framework's outcomes and emphasize the importance of selecting the most appropriate model for optimal performance.

5.10 Enhanced State Space with Data Inputs Experiment (DATA-ENH)

The primary objective of our third experiment is to assess the impact of the data used and, consequently, the state space of the DRL environment on the overall performance of the DPPG framework. The state space plays a fundamental role in shaping the environment for all DRL models. We aim to evaluate how the inclusion of OHLC data and Technical Indicators within the state space affects the performance of our DPPG model. As we expand the state space, we recognize that it increases complexity and computational resource requirements. Therefore, in order to justify the incorporation of additional data, we need to achieve improved results with the enhanced state space. To isolate the impact of the state space, we will compare the results with those obtained in BDDPG Experiment, reusing the exact same actor-critic DNN architecture. The experiment's fundamental aspects are as follows:

1. Dataset / State space: We use "Dataset 2" with OHLC data and "Dataset 3" with Technical Indicators to expand the state space.

2. Actor-Critic DNN architecture: We continue using the standard CNN architecture from BDDPG Experiment.

Our models have the potential for various improvements. While the Action, which involves a list of weights assigned to portfolio assets, remains a critical element of the RL framework, the State and Reward components can be adjusted to introduce enhancements. In this experiment, our focus lies on the state part. The current state format, consisting solely of log returns within a window limit, may be considered primitive and might not provide sufficient information for making accurate decisions. To address this limitation, we aim to enrich the state space with more relevant information. Additionally, considering that CNNs are typically used for image-based tasks, the expanded state space with OHLC and Technical Indicators could resemble image structures due to the matrices' shape. Consequently, convolutional layers might effectively leverage this new structure and yield promising results. By conducting this experiment, we seek to gain valuable insights into the significance of the state space in influencing the performance of the DPPG framework. We recognize that the data used within the state space plays a crucial role in shaping the learning process of the agent and, therefore, aim to justify the use of additional data by achieving improved results in comparison to the baseline experiment.

5.11 Synergistic DNN Architecture and Enhanced State Space Experiment (SYNERGY)

The final experiment represents the culmination of all the previous experiments. Having examined the isolated impacts of actor-critic DNN architecture and the enhanced state space, we now aim to explore the combined effects of both factors, anticipating that their SYNERGY will yield even more compelling results. In this experiment, we will employ our DPPG framework using interesting combinations of the new DNN architectures developed in DNN-AC Experiment with the enhanced state space developed in DATA-ENH Experiment. To ensure efficiency and relevance, we will focus on specific combinations based on the results obtained, avoiding unnecessary computations. The fundamental aspects of this experiment are as follows:

1. Dataset / State space: We will utilize "Dataset 2" with OHLC data and/or "Dataset 3" with Technical Indicators to achieve an enhanced state space according to the result obtained within the previous experiments.

2. Actor-Critic DNN architecture: We will explore the utilization of LSTM (standard and optimised) and/or optimised CNN, and/or standard and optimised CRNN architectures within our DPPG framework according to the result obtained wihtin the previous experiments.

By simultaneously addressing both the state space and actor-critic DNN architecture, we are delving into two fundamental aspects of every RL framework, as discussed in DATA-ENH Experiment (the state), and the core components of the DDPG model, namely actor-critic networks. Improving upon this experiment could involve exploring alternative algorithms, such as TD3, or experimenting with the RL framework by designing different reward metrics beyond the standard return. However, such directions are beyond the scope of this project. While it is challenging to make definitive hypotheses regarding the performance of each combination, some insights may prove relevant. Given the increased dimensionality of the state space and the presence of a recurrent layer, the CRNN-based model may outperform the LSTM-based model, benefiting from its ability to capture temporal patterns effectively. Through this final experiment, we aim to unlock valuable insights into the combined impact of actor-critic DNN architecture and enhanced state space on the performance of our DPPG framework. By leveraging the complementary strengths of both factors, we anticipate achieving significant advancements in our portfolio optimization strategy.

6 Results

In the preceding sections, we meticulously designed and executed a series of methods and experiments to delve deep into the DDPG framework and its potential for portfolio optimization. As we transition into the results, we stand at the precipice of understanding the outcomes of our endeavors and the implications they hold for the world of financial portfolio management.

1. **BDDPG:** served as our foundational step, where we implemented a basic DDPG framework and juxtaposed its performance against established benchmarks. This experiment not only set the stage for the subsequent ones but also provided a baseline against which the enhancements of Experiments 2 and 3 could be measured.

2. **DNN-AC:** our focus shifted to the neural architectures underpinning the actor-critic networks. We anticipated discerning the influence of different Deep Neural Network (DNN) architectures on the DDPG framework's performance, with a particular interest in the comparative capabilities of LSTM, CNN, and CRNN models.

3. **DATA-ENH:** pivoted our attention to the state space of the DRL environment. By introducing OHLC data and Technical Indicators, we aimed to gauge the impact of an enriched state space on the DPPG model's efficacy. The underlying hypothesis was that a more information-rich state space could potentially enhance the agent's decision-making strategy.

4. **SYNERGY:** represented the endpoint of our research journey. Here, we sought to harness the synergistic potential of combining the insights from Experiments 2 and 3. By evaluating various combinations of DNN architectures and state spaces, we hoped to identify an optimal configuration that maximizes the DDPG framework's performance.

As we delve into the results, our objective is twofold: to validate our hypotheses and to extract actionable insights that can guide future research and practical applications in portfolio optimization. Note that in order to obtain insighful plot we sometimes use rolling method with a rolling window of value 100 (days) in order to smooth the curve. It is the case in our representation of Sharpe, Sortino and Maximum Drawdown ratio as well as Daily Average Yield. We also used the same method to illustrate the evolution of rewards over each episode in order to smooth the curve and make it easier to read.

Finally, we would like to emphasize the following point. Our results are heavily embellished and far from being representative of a real market situation, as we rely on strong assumptions about the market and utilize closing prices for entry and exit point, which is in practice not possible. Consequently, our primary focus within this research is not on the precise returns or final portfolio values. Instead, we aim to compare these embellished results among themselves, considering the various network architectures and state space representations that we employ.

6.1 BDDPG

In BDDPG, we introduce and evaluate a foundational DDPG framework using a simple CNN with closing prices as input, and compare its performance against benchmark strategies, namely UP, OLMAR, and BCRP. The outcomes of this experiment provide a reference point for subsequent investigations and offer insights into the DDPG model's potential in portfolio optimization.

6.1.1 Benchmarks Results

In this section, we will analyze the performance of the benchmark methods introduced earlier in the project. Specifically, we will focus on metrics such as portfolio value, rolling yields, Sharpe

and Sortino ratios, and maximum drawdown. We will also use that section to make some racall about each metrics and how to anylze them as we will used them exstensively in the following experiments. It's important to note that these benchmarks were tested exclusively during the period from early 2017 to late 2020, which is the same time frame we will use to evaluate our DDPG framework in the subsequent sections. One may find in Table 7 a summary of the benchmar's results.

Portfolio Value across Test Dates: Figure 17 showcases the portfolio value (PV) for different benchmarks across various test dates. The variation in PV indicates the performance variability of the benchmarks. By analyzing the graph, it is evident that all three benchmarks achieve a positive PV at the conclusion of the testing period. Notably, the OLMAR method stands out for its superior performance compared to the other two methods, BCRP and UP, over time. Moreover, it managed to achieve most of its performance gains during the final year. To be precise, the BCRP method, represented by the red line, reaches a portfolio value of 1.671. Although this achievement is commendable, it is overshadowed by the remarkable ascent of the OLMAR method, indicated by the yellow line, which reaches an impressive value of 3.483. In contrast, the UP method, denoted by the pink line, adopts a more conservative approach and culminates in a portfolio value of 1.151. It's worth noting that the last year of the test set coincides with the global COVID-19 pandemic. This pandemic had a significant impact on the stock market, resulting in substantial losses during the initial month of the outbreak, followed by a robust and swift recovery over the subsequent year. Examining the graph, one might suspect that both OLMAR and BCRP managed to navigate most of the losses while capitalizing on the subsequent market rebound. However, it's essential to remember that our analysis focuses on a restricted set of 25 assets. Therefore, it's plausible that the selection of these assets could have introduced a bias into the results, potentially avoiding the worst-performing companies during the pandemic period. This is especially highly possible noting that the BCRP method performs well during this period while being unable to adjust his portfolio weights at anytime by definition.

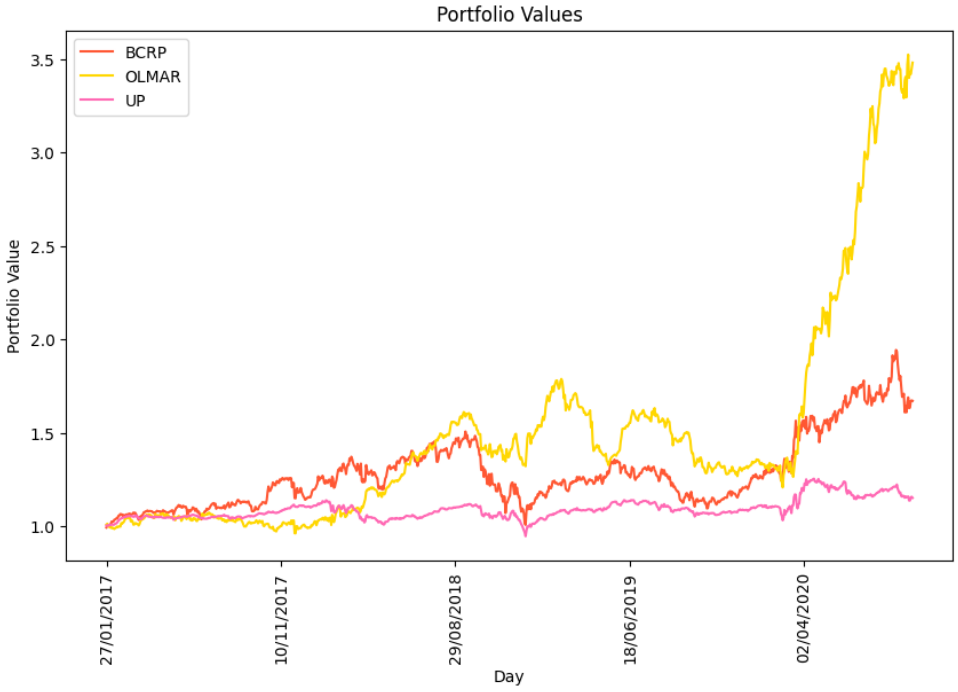

Figure 17: Portfolio Value across Test Dates (OLPS)

Rolling Maximum Drawdown: Remember that Maximum drawdown is an essential metric

for risk evaluation. A higher MDD means a higher potential loss, which investors generally want to avoid. By examining this graph, we can infer which strategies have a more consistent performance and which ones might pose higher risks during specific periods.

The rolling Maximum Drawdown (MDD) for the benchmarks, depicted in Figure 18, provides insights into the risk associated with each strategy over time. OLMAR and BCRP have similar maximum drawdowns of 32.4908% and 33.2060% respectively, indicating periods of significant value loss. However, UP, with a max drawdown of 16.9325%, underscores its risk-averse strategy, ensuring limited potential losses even if it compromises on higher returns. Also, the MMD values fluctuate across the time frame, indicating periods of increased risk or losses for certain strategies at specific points in time. We can observe that between late 2018 and mid-2019, all methods were subject to high MDD, especially BCRP and UP which are having their MMD over all the entire period during this interval. Note that we don't observe the real MDD of OlMAR method wich indicates that it happens during a period of time greater than 100 days as this is the window value used for the rolling method.

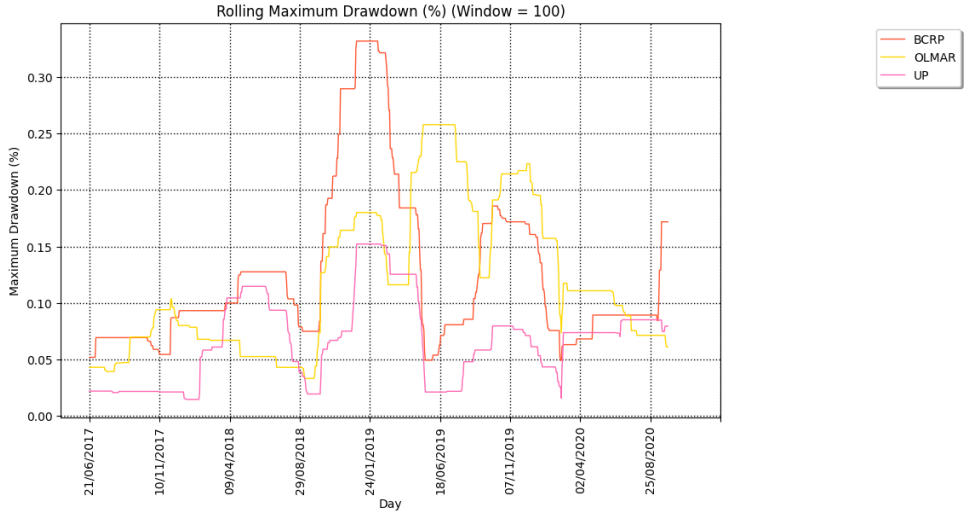

Figure 18: Benchmark's Rolling Maximum Drawdown (OLPS)

Rolling Sharpe Ratio: Remember that the Sharpe ratio measures the risk-adjusted return of an investment. A higher Sharpe ratio suggests that the investment has either a higher return for a given level of risk or a lower risk for a given return. Observing the variations in the Sharpe ratio allows investors to evaluate the risk and return trade-off for each strategy.

Figure 19 represents the rolling Sharpe ratio, reflecting the risk-adjusted return of the benchmarks over time. The OLMAR method boasts an impressive Sharpe ratio of 9.7054, indicating its superior return for the risk undertaken. BCRP, with a ratio of 4.2116, offers a balanced risk-reward profile. UP, with its 2.4416 Sharpe ratio, might provide lower returns but does so with a reduced risk profile. Furthermore, if we examine the substantial gap in the Sharpe ratio of each model, we notice that it aligns with the period characterized by high MDD. This correlation is logical, given that the risk-return ratio is significantly affected when our strategy experiences substantial drawdowns. Also, even if the UP strategy is having a better management of risk according to the result of MDD, one may observe that the sharpe ratio of this strategy is still lower than the other one, most likely because of the extreme returns made by these last ones.

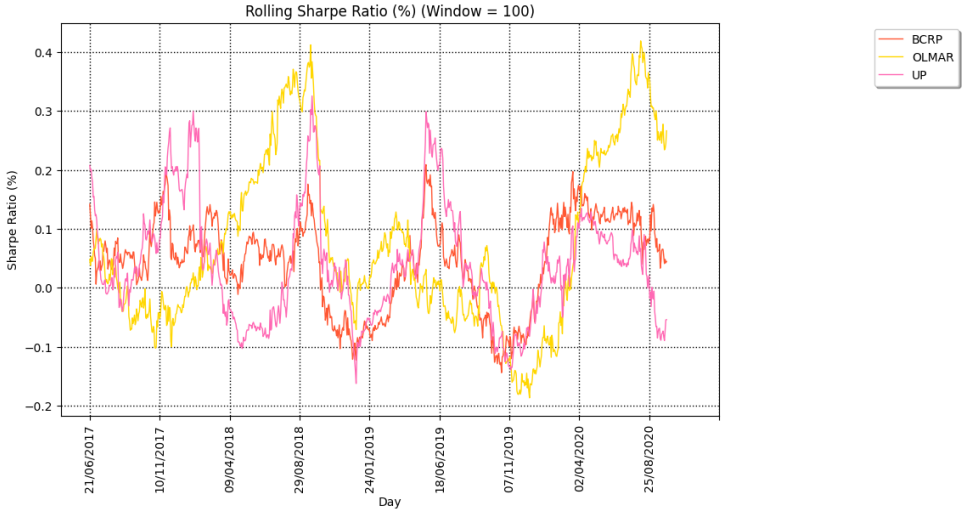

Figure 19: Benchmark's Rolling Sharpe Ratio (OLPS)

Rolling Sortino Ratio: Remember that the Sortino ratio is akin to the Sharpe ratio but focuses on negative volatility or downside risk. A higher Sortino ratio indicates a better risk-adjusted performance, especially when considering the undesirable downside fluctuations.

The rolling Sortino ratio, highlighting the downside risk-adjusted performance, is illustrated in Figure 20.OLMAR, with its 13.8401 Sortino ratio, demonstrates that it not only generates high returns but does so with minimal harmful volatility. BCRP's 5.62485 and UP's 2.9688 further reiterate their respective risk-return stances, with UP being the most conservative in terms of downside fluctuations. We also observe the same trend as the one observe over the sharp ratio. Confirming that the Sortino ratio is closely related to the MMD.

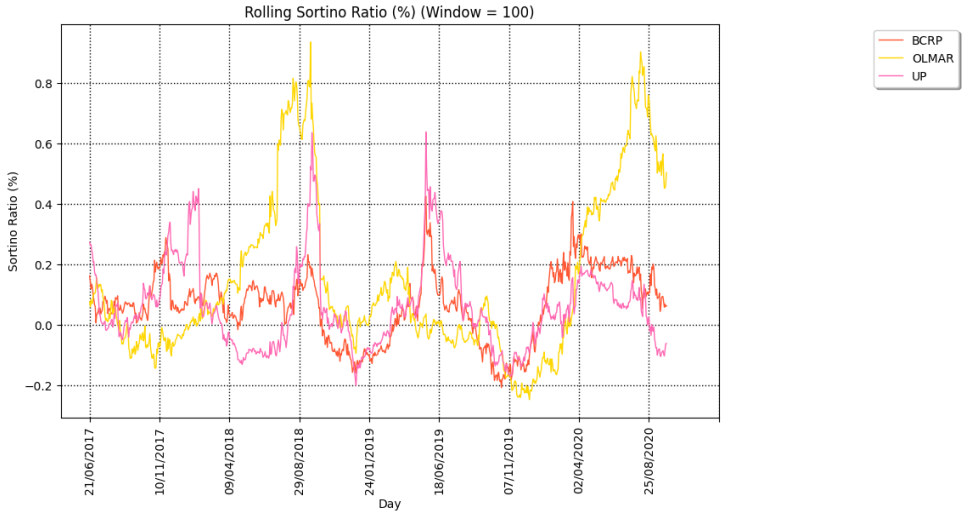

Figure 20: Benchmark's Rolling Sortino Ratio (OLPS)

Rolling Yield: Remember that the rolling yield/return provides insight into the income generated by an investment over specific periods. By examining this graph, we can deduce which strategies consistently provide higher yields and which ones might be more volatile in terms of returns.

Figure 21 demonstrates the rolling yield for the benchmarks, offering insights into the returns

generated over specific periods. The average daily returns further delineate the benchmarks' performance disparities. OLMAR once again leads the pack with a 0.1454% return, demonstrating its consistent profitability during the test period. BCRP follows with a respectable 0.0704%, whereas UP trails with a modest 0.0175%, reflecting its conservative stance.

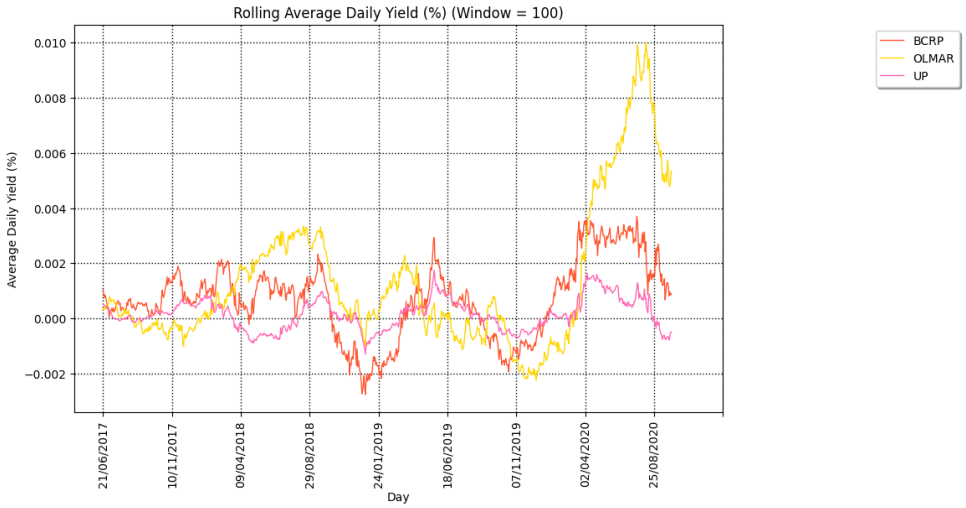

Figure 21: Benchmark's Rolling Average Daily Yield in %

Benchmark	Portfolio Value	Avg Daily Return %	Sharpe Ratio	Sortino Ratio	Max Drawdown
BCRP	1.671	0.0704	4.2116	5.62485	33.2060
OLMAR	3.483	0.1454	9.7054	13.8401	32.4908
UP	1.151	0.0175	2.4416	2.9688	16.9325

Table 7: Summary of Key Metrics for Benchmarks

6.1.2 DDPG Framework using CNN and Closing Prices

In this segment of our study, we explore the implementation of the DDPG framework using a CNN architecture. Our focus is on evaluating this framework's performance when solely utilizing the closing prices of assets. We'll assess the outcomes across various window lengths to comprehend the influence of the input's temporal extent on our results.

6.1.2.1 Training Analysis

The training process involves feeding the agent with the temporal sequences of the closing prices of the assets, with each sequence's length matching the specified window length. Given the sequential nature of our input data, a CNN architecture is employed to extract crucial temporal features, which are then utilized by the DDPG agent to take portfolio allocation actions.

Two key metrics during the training process, indicative of our agent's learning progress, are the Q-values and the rewards. The evolution of Q-values (Figure 22) is indicative of the agent's approximation of the state-action value function. A steadily increasing Q-value trend suggests that the agent is progressively getting better at gauging the potential value of its actions. The rewards evolution (Figure 23), on the other hand, offers insights into the actual returns that the agent is achieving during its training episodes. Note that on rewards plot, we also plot the rolling mean curve using a window of length 10 (episodes) in order top smooth the curves.

Upon examining Figure 22, we note that models with window lengths of 11 and 9 converge in fewer than 100 episodes towards their final values. These final values are also among the lowest Q-values obtained after training. Furthermore, we observe the convergence of the models with window lengths of 3 and 7, with the latter achieving the highest Q-value after 400 episodes. However, there is a concern regarding the model with a window length of 5, as it does not appear to have converged even after 400 episodes. Nonetheless, on the whole, we are satisfied with the evolution of the Q-values, and one might expect the best results from models with window lengths of 5 and 7. Turning to Figure 23, we also observe convergence in the reward curves. More significantly, there is a clear distinction between the reward curves from models with window lengths of 9 and 11 compared to the others. This observation aligns with the results of the Q-value evolution during training. Note that models with window length and 11 does not seems to have any trends compared to the other following a distinct upward trend. Also the volatility of the rewards of the latter is clearly evident compared to the model with window length 9 and 11. High volatility may be the sign of good exploration and therefore could let us assume that these models would perform better.

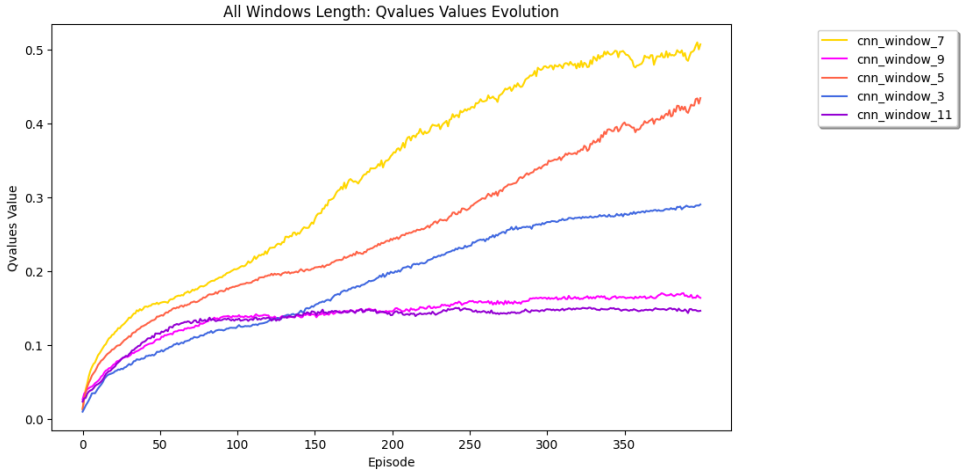

Figure 22: BDDPG: Evolution of Q-Values during Training

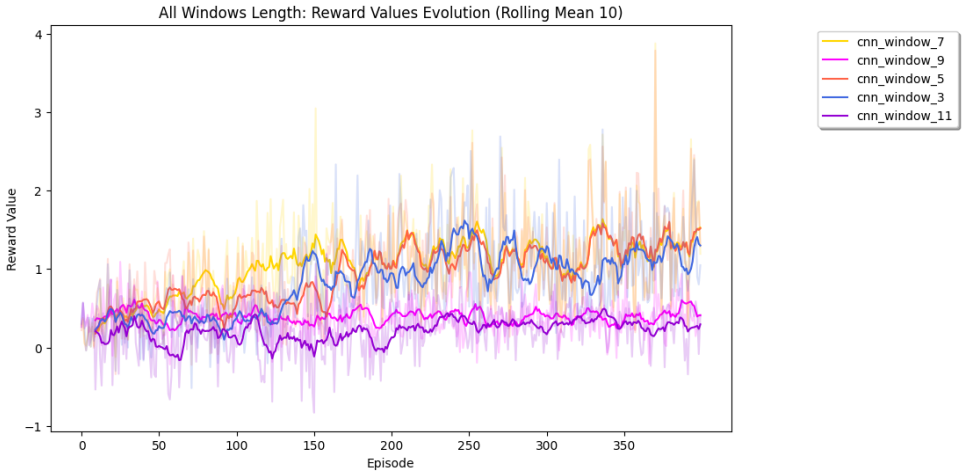

Figure 23: BDDPG: Evolution of Rewards during Training

6.1.2.2 Testing Analysis

After the training phase, we assess the performance of our DDPG framework on the test set. We evaluate this performance based on various metrics, comparing the outcomes across different window lengths.

Portfolio Values: Figure 24 presents the portfolio value across the testing period for different window lengths. By observing the graph, it is evident that the window lengths of 5 and 7 days consistently outperform the other configurations, with final portfolio values of 9.650 and 8.515 respectively. The 3-day window, while achieving a commendable portfolio value of 4.773, is overshadowed by the 5 and 7-day windows. Interestingly, the 9-day and 11-day windows demonstrate diminished performance with values of 1.785 and 1.627, respectively, as expected after the analysis of the training phase.

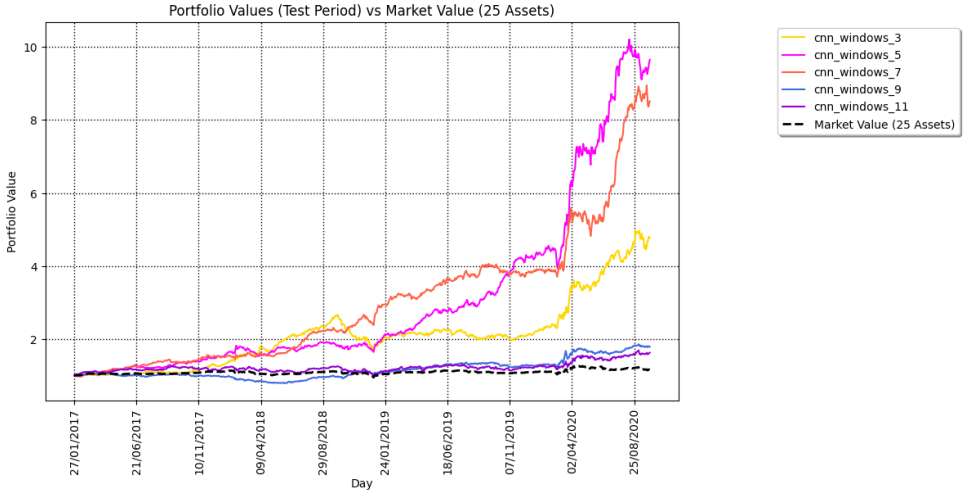

Figure 24: BDDPG: Portfolio Value across Test Dates for Different Window Lengths

Metrics: To achieve a comprehensive understanding of our models' performance, it's paramount to examine the average rolling metrics, including the Sharpe ratio, Sortino ratio, yields, and drawdown. Presented in Figures 25, 26, 27, and 28, these metrics offer valuable insights. The models with 5 and 7-day windows consistently dominate the rolling Sharpe and Sortino ratios, highlighting their superior risk-adjusted returns and resilience to negative volatility. This superior performance is especially significant when considering the tumultuous market conditions during the test period. The rolling yields further accentuate the efficiency of the 5 and 7-day windows, as they recurrently deliver higher returns than the other configurations, ensuring profitability over extended durations. When considering the drawdown, the minimized values observed for these windows underline their ability to effectively manage risk and limit potential losses. These metrics, in tandem, validate the prominence of the 5 and 7-day windows in the DDPG framework, affirming their potential in real-world portfolio management scenarios.

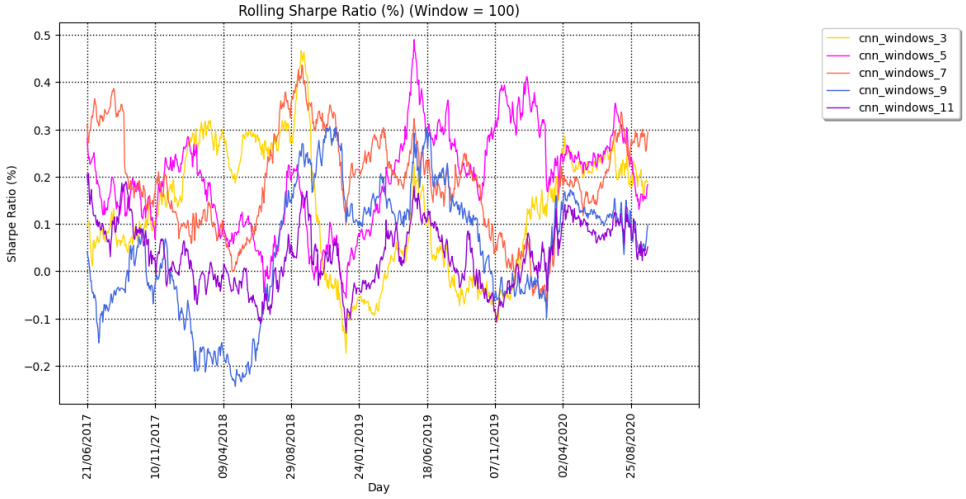

Figure 25: BDDPG: Rolling Sharpe Ratio for Different Window Lengths

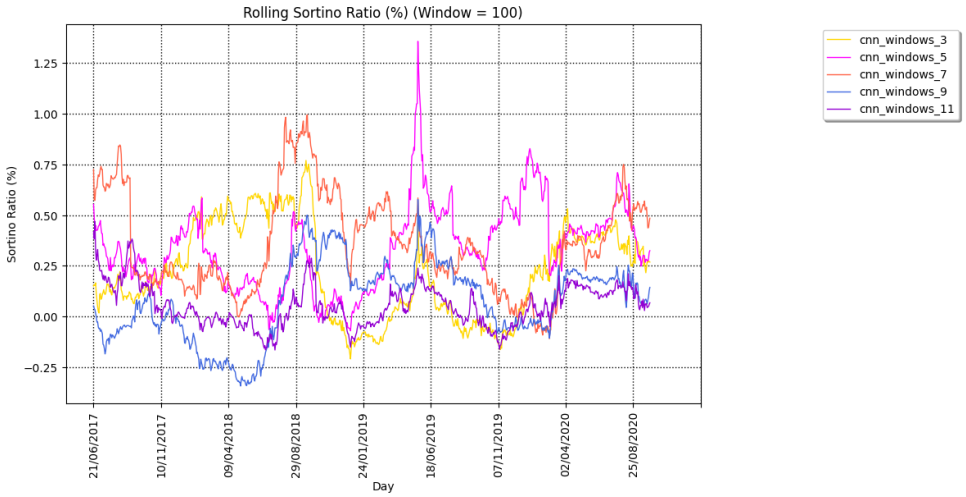

Figure 26: BDDPG: Rolling Sortino Ratio for Different Window Lengths

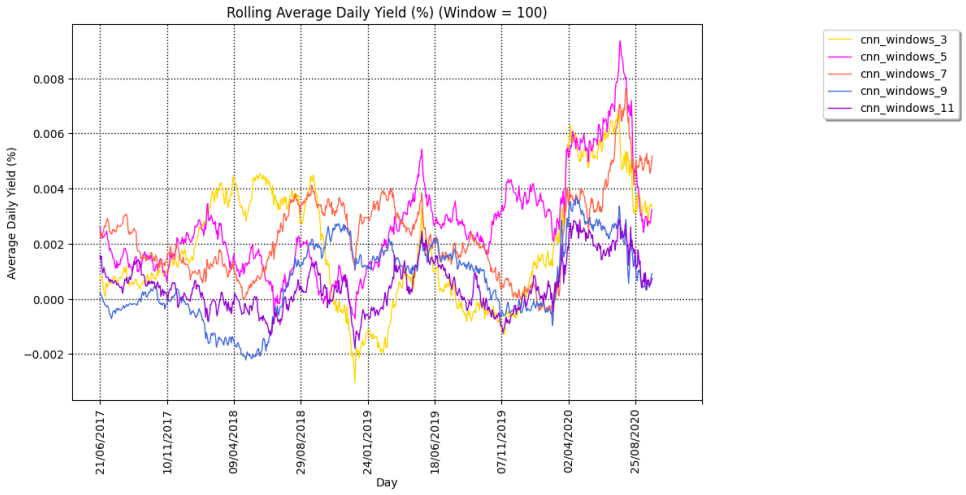

Figure 27: BDDPG: Rolling Yield for Different Window Lengths

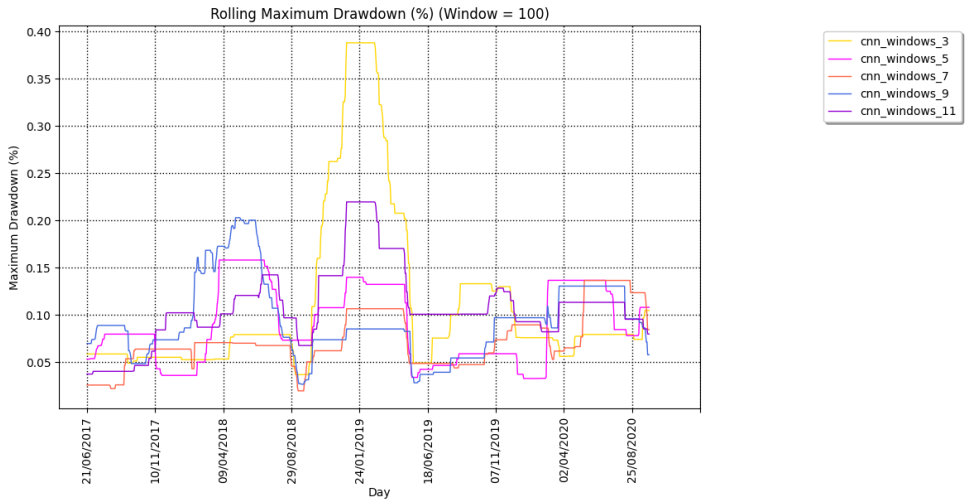

Figure 28: BDDPG: Rolling Drawdown for Different Window Lengths

In conclusion, the DDPG framework's implementation using CNNs and closing prices proves to be a promising approach, especially with window lengths of 5 and 7 days. When juxtaposing the DDPG results with the earlier benchmarks, it's evident that the DDPG approach, especially with the optimal window lengths, consistently outperforms the benchmarks in various metrics. For instance, the Portfolio Value and the Sharpe Ratio for the 5 and 7-day windows are significantly higher than those of the benchmarks. This indicates not only the viability of the DDPG framework but also its capability to deliver superior results in the domain of portfolio management using similar data and a basic NN architecture.

Now, upon juxtaposing the training process observations with the testing outcomes, intriguing insights emerge. The models with Window Length 11 (W11) and Window Length 9 (W9), despite their rapid convergence during training, did not translate into top-tier performance during testing. Their swift convergence and lower Q-values may have resulted from overfitting or a lack of adaptability to diverse market scenarios. Contrastingly, the models with Window Length 5 (W5) and Window Length 7 (W7) exhibited outstanding results in the testing phase. The model with W5 showcased a remarkable portfolio value and other metrics, despite its non-converging Q-value evolution. This discrepancy emphasizes that while Q-value convergence provides an insight into the model's learning, it doesn't necessarily guarantee superior performance in real-world testing. The model with W7, on the other hand, aligns well with our expectations from training, cementing its robustness. The distinct reward curves for models with W11 and W9 further cement their divergence from the top-performing models. The reward curves, coupled with the Q-value evolution, offer a holistic view of model behavior, suggesting that a balance between quick convergence and adaptability is crucial for optimizing performance.

Window length	**Portfolio Value**	**Avg Daily Return %**	**Sharpe Ratio**	**Sortino Ratio**	**Max Drawdown**
3	4.773	0.1813	11.9864	17.8458	38.7976
5	9.650	0.2551	17.9600	29.4021	15.8132
7	8.515	0.2405	18.8610	32.3762	13.6592
9	1.785	0.0691	6.4011	8.7301	25.4936
11	1.627	0.0609	4.6126	5.9906	22.1282
Market Value (25 Assets)	1.160	0.0183	3.1334	2.5726	16.7101

Table 8: Summary of Key Metrics from BDDPG

6.1.2.3 In-Depth Analysis of CNN based model with W5

The model with a W5 stood out in our tests, performing better than the others in many areas. In this section, we'll take a closer look at how this model works. We'll explore its strategies, the stocks it preferred, and try to understand why it did so well. This deep dive will help us get a clearer picture of its strengths and might offer hints for improving other models in the future.

Asset Distribution with Portfolio Value: We provide a plot illustrating the primary chosen assets for each step are presented, overlaid onto the portfolio value graph. The continuous action space necessitated the creation of data points within the plot by selecting the asset with the highest weight at each step. It is accompanied by a graph displaying the normalized values of the corresponding assets. Refer to Figure 29. A couple of observations can be made:

- *Diversification:* The model seems to have a preference for a mitigate diversification, using mainly 5 stocks: BAC, CMCSA, JNJ, PG, UNH.
- *Concentration in Selected Assets:* As time progresses, we observe certain assets becoming more dominant in the portfolio. This could reflect the model's identification of these particular assets as potential high-performers or as stable assets during volatile periods.
- *Portfolio Value Correlation:* The portfolio's value increases as the model starts concentrating on specific assets, suggesting the asset selection strategy's effectiveness.
- *Loss aversion:* while observing a huge gap of all assets at the begining of the COVID-19 pandemic, we see that the agent is able to mitigate that loss and still take advantage of the strong recovery period.

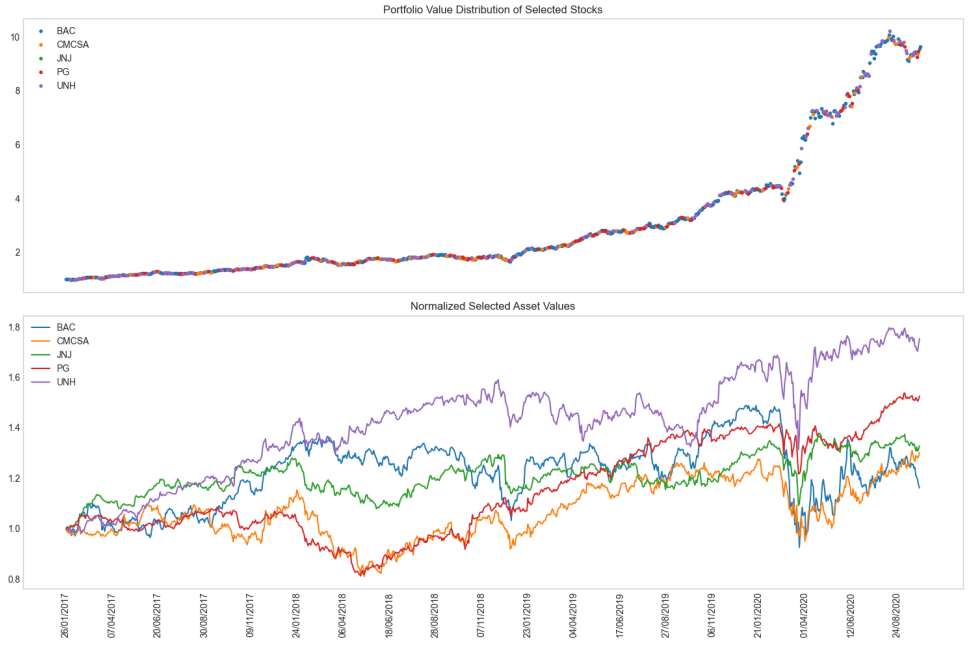

Figure 29: BDDPG with W5: Asset Distribution with Portfolio Value Over Testing Time

Frequency Distribution of Assets Selected by the Agent: We also add a graph showing the number of step an asset has been selected (with limiting ourself to highly weighted assets only). Refer to Figure 30, one may observe that the top asset (BAC) is not the one performing the best over the testing period. However the agent managed to make return out of it, meaning that he was able to understand correctly and to predict the behavior of this asset.

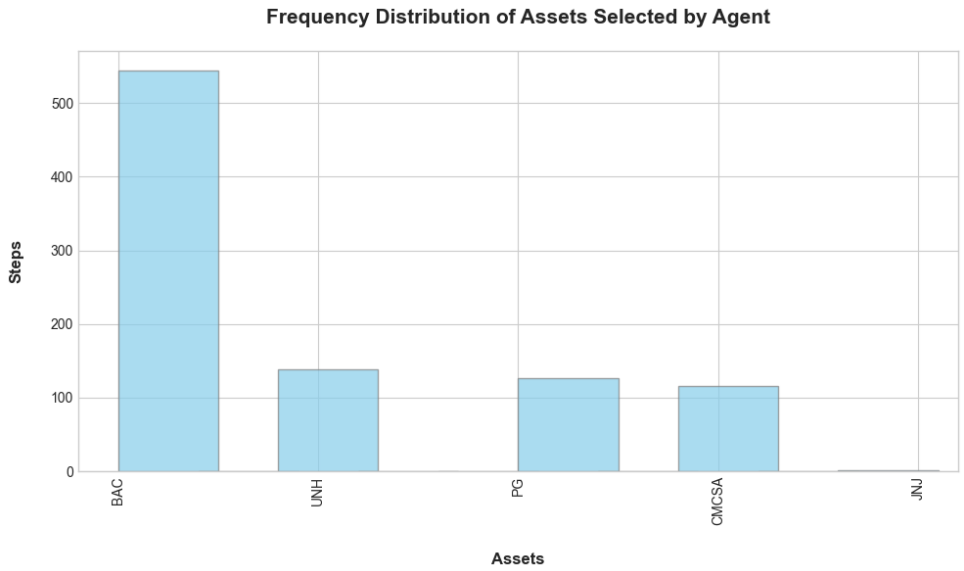

Figure 30: BDDPG with W5: Frequency Distribution of Assets Selected by the Agent

Detailed Portfolio Allocation over a selected time period: We ultimately present a composite Figure that combines various plots. Within a specified time period, we graph the normalized asset values of selected assets, their log returns, the weights assigned to each asset by our agent, and the normalized value of our portfolio. Referring to Figure 31, we can infer:

- *Consistent Core:* While the model modifies its asset selection, there's a consistent set of assets it frequently selects.
- *Occasional Diversification:* There are periods the model diversifies more than usual, likely in response to market volatility.
- *All or nothing:* The model tends to allocate the entire weight to a single asset instead of achieving diversification across multiple assets.

Figure 31: BDDPG with W5: Portfolio Analysis over a chosen time period

6.2 DNN-AC

In DNN-AC Experiment, we assess the influence of different Deep Neural Network (DNN) architectures within the actor-critic framework of the DDPG model, specifically comparing LSTM, CNN, and CRNN structures using only closing prices as input. The goal is to discern the efficacy of each architecture in handling time-series data, emphasizing the crucial decision-making involved in model selection for optimal performance.

In the forthcoming sections detailing the results of DNN-AC Experiment, our approach will be concise, focusing primarily on the most pertinent outcomes. Particularly in the testing phase, emphasis will be placed on portfolio values, while other metric results will be tabulated for clarity, sidestepping the inclusion of extensive graphical representations. Deep-dive analyses will be reserved for models deemed most relevant based on their performance and characteristics.

Additionally, any figures or data deemed valuable but not central to the main discussion will be allocated to the project's appendix for those interested in a more granular exploration.

6.2.1 Training Analysis

Upon examining the training results across different architectures, we observed the following trends:

LSTM Architecture: From Figure 32, we can make the following observations:

- Models with Window Length 3 (W3) (blue) and W11 (purple) appear to achieve the highest Q-values among the configurations.
- The model with a window length of 9 (pink) exhibits a steady increase in Q-value, suggesting a consistent learning process. However, it doesn't surpass the values of W3 and W11.
- Models with W7 (yellow) and W5 (red) present relatively moderate Q-values, with a fast convergence towards their maximum Qvalue.

On rewards evolution, from Figure 33 include:

- The model with a W3 (blue) demonstrates a clear upward trend during its training. Despite encountering high volatility, it achieves the highest rewards by the end of the training phase.
- Models with W5 (red) and W7 (yellow) are notably similar in their performance. Both lack any distinctive trends, consistently yielding rewards below 0.5 and exhibiting low volatility.
- The model with W9 (pink) follows a clear upward trajectory throughout its training, marked by high volatility, and secures commendable rewards by its conclusion.
- Similarly, the model with a W11 (purple) showcases a consistent upward trend during training, resulting in impressive reward growth by the end.

Considering our observations from BDDPG Experiment and the patterns observed in the training graphs above, the LSTM model with W3 and seem to be the most promising candidates for optimal performance. However, the behavior of the model with W5 suggests it might be more adaptable and consistent across various market scenarios. Given that it had a strong showing in BDDPG Experiment, it's plausible to expect it to perform well in testing scenarios too.

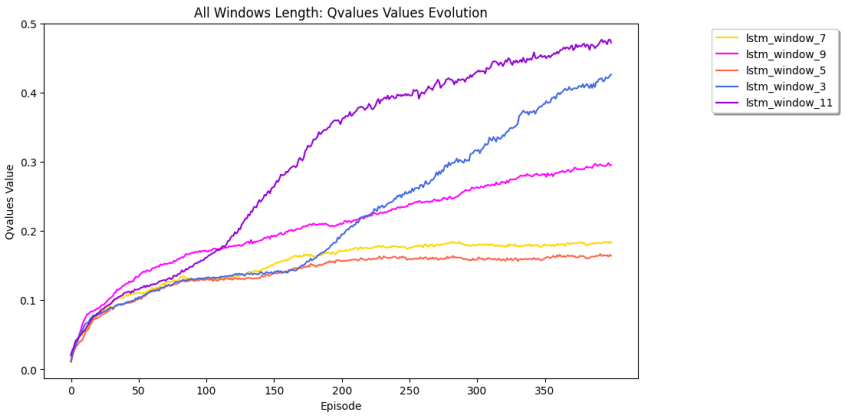

Figure 32: Q-values Evolution for LSTM Architecture from DNN-AC

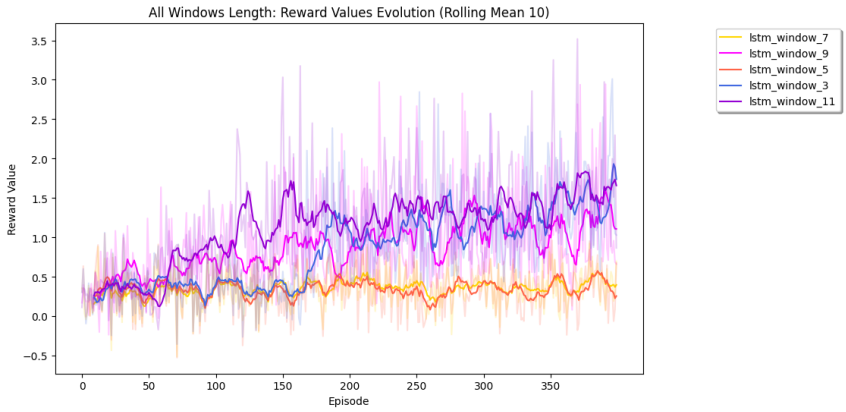

Figure 33: Rewards Evolution for LSTM Architecture from DNN-AC

CRNN Architecture: From the Q-values presented in Figure 34, and the corresponding rewards (Figure 35) we could make the following analysis. The Q-values and rewards observed during the training process of the CRNN architecture are notably different from the previously discussed architectures.

- *Magnitude of Q-values and Rewards*: The magnitude of both the Q-values and rewards is roughly half compared to those of the LSTM model. This reduced magnitude might be indicative of the model's potential difficulties in capturing and predicting the underlying dynamics of the market data. A lower Q-value suggests that the model anticipates lesser future rewards, which might be a cause for concern regarding its efficacy in portfolio optimization.

- *Trends Across Window Lengths*: For all window lengths, we notice that the Q-values are steadily increasing, albeit with some fluctuations. This indicates that the model is continuously learning and improving its policy. However, the rewards seem to be converging more slowly, with the 3-day window model showing a slight upward trend, while the other window lengths are relatively stable.

In conclusion, while the CRNN model exhibits consistent learning, the reduced magnitude in Q-values and rewards, coupled with the slower convergence of rewards, raises questions about its performance compared to the LSTM and CNN models . Its real efficacy can be further judged by analyzing its performance on testing data.

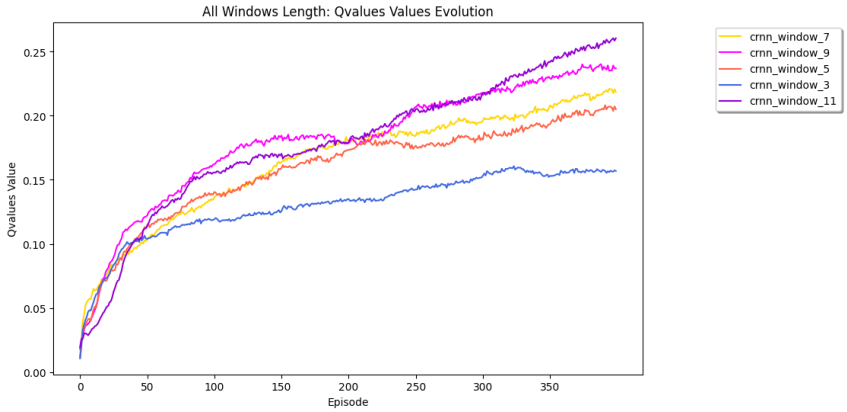

Figure 34: Q-values Evolution for CRNN Architecture from DNN-AC

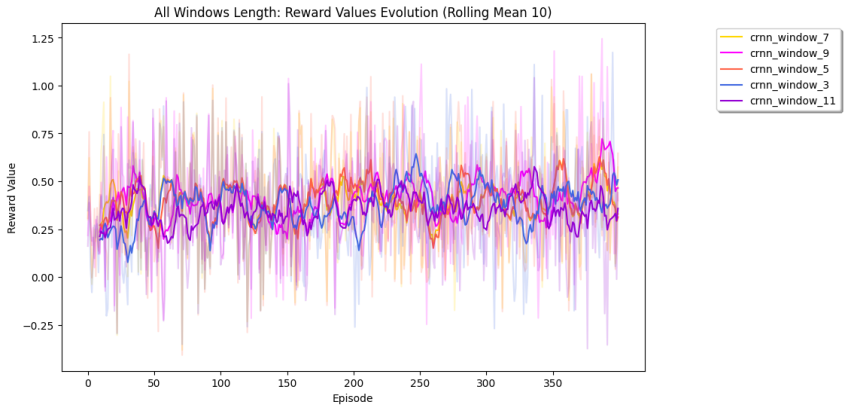

Figure 35: Rewards Evolution for CRNN Architecture from DNN-AC

CNN Optimized Architecture: Upon examining the Q-value evolution (Figure 36), we can make the following observations:

- The model with a window length of 5 is showing promising signs. It seems to be converging towards higher Q-values as the episodes progress and still indicates potential for further increase.
- Models with W3 and W7 exhibit steady convergence patterns and achieve Q-values that are quite comparable to each other. However the early convergence might indicate a lack of exploration during training which could lead to bad results.
- The model with W11 is similar to the models with W3 and W7 but the fact that it does not converge early could be a good sign of exploration and might lead to better results in the testing phase.
- Interestingly, the model with W9 converges rapidly to a relatively low Q-value and demonstrates limited improvement as episodes progress.

Turning our attention to the reward evolution (Figure 37), the following patterns emerge:

- The reward curves for W3, W5, and W7 show a rising trend, suggesting an improving performance over time.
- However, the model with window length 11 demonstrates an erratic behavior in rewards, which is consistent with its Q-value evolution in Figure 36

Comparing the CNN Optimized architecture with the standard CNN architecture from our previous analysis, we notice the optimized models with W9 and W11 seem to struggle more compared to the standard models of similar window lengths. The continuous oscillations and the lack of clear convergence may indicate challenges in the training process for these particular configurations. In conclusion, the CNN Optimized architecture has exhibited improvements for certain window lengths, underscoring the importance of architectural refinements. However, not all optimizations lead to consistent improvements across all configurations, emphasizing the need for a careful selection of hyperparameters and training setups.

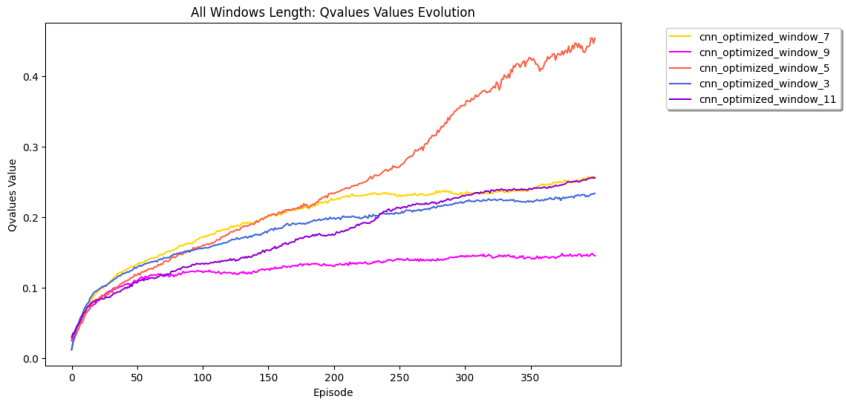

Figure 36: Q-values Evolution for CNN Optimized Architecture from DNN-AC

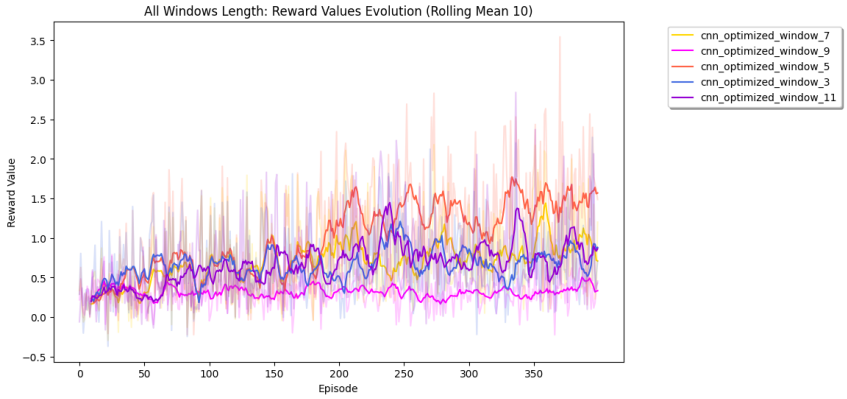

Figure 37: Rewards Evolution for CNN Optimized Architecture from DNN-AC

LSTM Optimized Architecture: The Q-value evolution for the optimized LSTM (Figure 38) showcases a few discernible patterns:

- The blue line trends upward (window length 3) remains relatively stable throughout, suggesting a consistent learning process and stabilizes after 200 episodes.
- The red line (window length 5) sees an initial spike before stabilizing, suggesting initial rapid learning which then converges. However, we can spot a new increase after 250 episodes suggesting a new exploration phase.
- Yellow (window length 7) and pink (window length 9) lines display similar behavior, with rapid convergence and stabilization at low Q-values suggesting low exploration and probably a lack of generalization of the model.
- The purple line (window length 11) is similar to the 7 and 9-windows lines but seems to trends downward after reaching its peak qvalue. It will be interesting to observe the performance of this model later in order to understand this behavior.

For the rewards (Figure 39):

- The blue line (window length 3) experiences slight fluctuations but remains stable and achieve highest rewards overtime
- Red (window length 5) and yellow (window length 7) lines converge relatively quickly and maintain stable rewards. Except at the end of the training where, as for the Q-value, the rewards from the 5-window lentgh model seems to be increasing again.

- The pink line (window length 9) has a slight dip in the initial phase but eventually stabilizes.
- The purple line (window length 11) does not seem to have any trends and stay low during the entire training process.

In conclusion, the optimized LSTM appears to force rapid convergence and stability, which might indicate overfitting and/or lack of exploration during the training phase. However, both variants have their unique strengths, and their performance might differ based on the specific market scenario. We will compare the final result in the testing section.

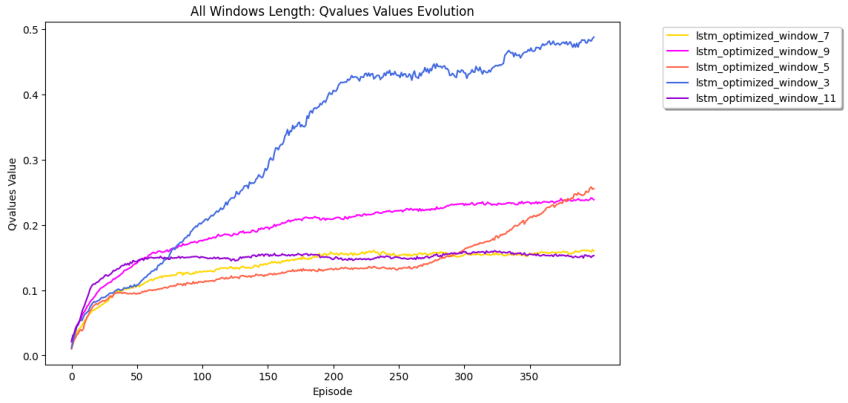

Figure 38: Q-values Evolution for LSTM Optimized Architecture from DNN-AC

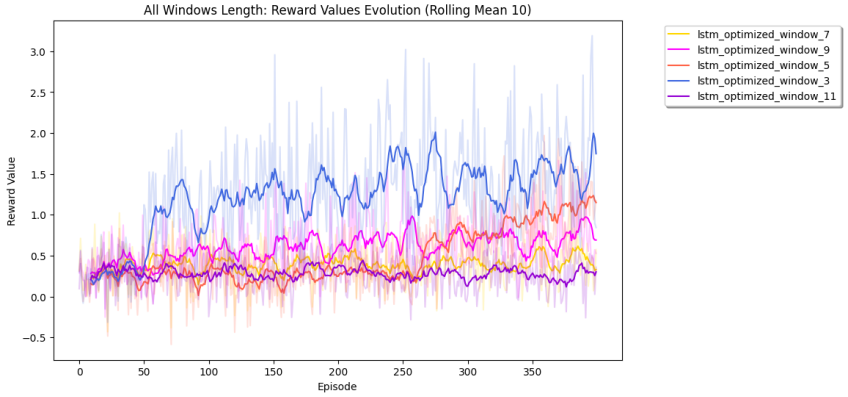

Figure 39: Rewards Evolution for LSTM Optimized Architecture from DNN-AC

CRNN Optimized Architecture: The CRNN Optimized architecture presents a rather chaotic and non-convergent behavior during its training phase, as evident from Figures 40 and 41. Particularly, the reward curves are predominantly declining over episodes for almost all window lengths, suggesting that the model is struggling to optimize its policy. The Q-value evaluations further substantiate this claim; some window lengths exhibit an exponential surge or even plummet to zero after initial learning phases. This erratic behavior, especially visible in the yellow (W7), red (W5), pink (

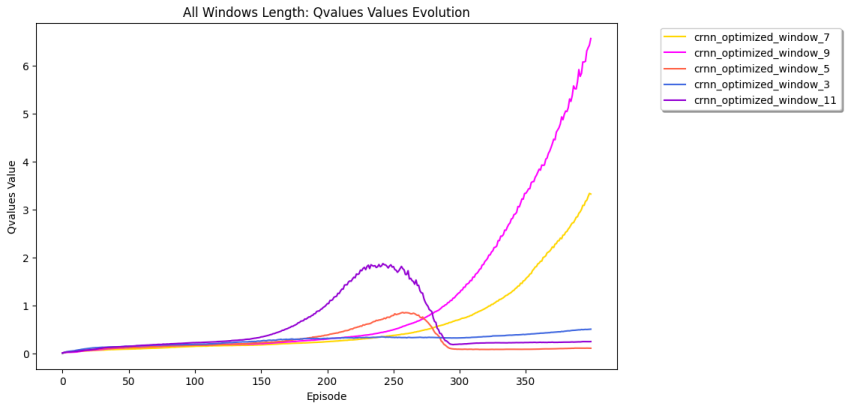

Figure 40: Q-values Evolution for CRNN Optimized Architecture from DNN-AC

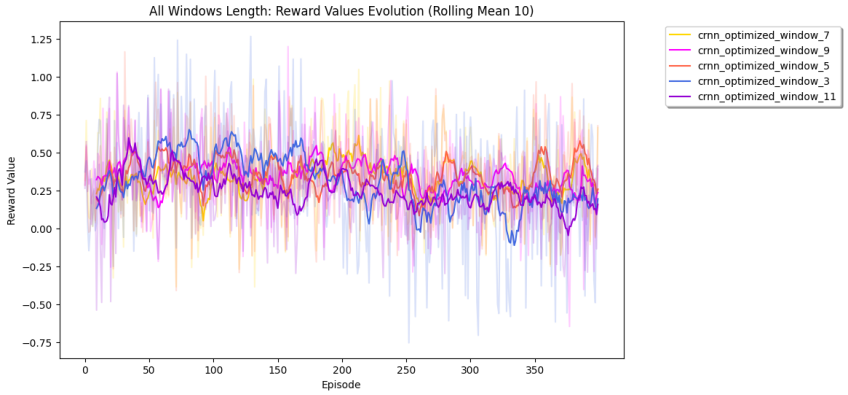

Figure 41: Rewards Evolution for CRNN Optimized Architecture from DNN-AC

6.2.2 Testing Analysis

Upon delving into the testing results, we immediately recognize some noteworthy patterns that provide insights into the performance of different neural network architectures and their respective variations. The results can be viewed in Table 9, and the portfolio value (PV) evolution during the testing phase for different architectures is illustrated in Figures 42, 43, 44, 45, and 46.

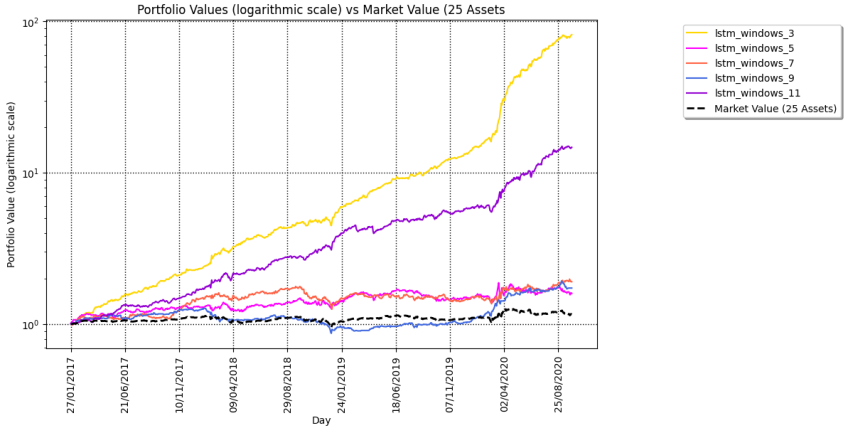

Figure 42: Portfolio Value Evolution for LSTM Architecture from DNN-AC

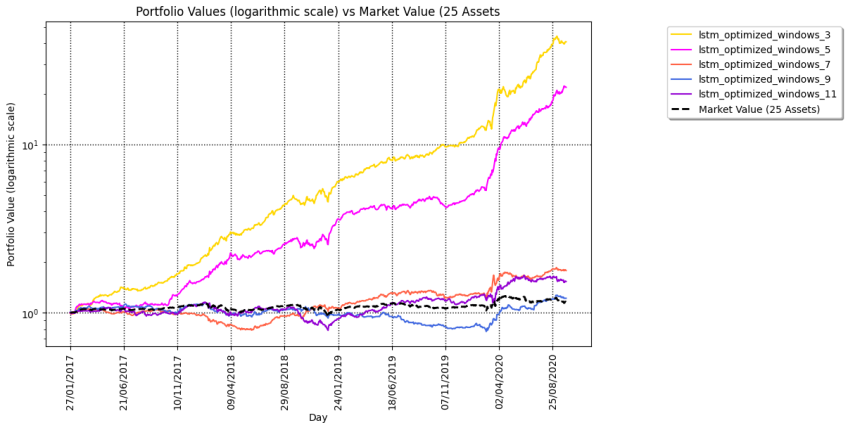

Figure 43: Portfolio Value Evolution for LSTM Optimized Architecture from DNN-AC

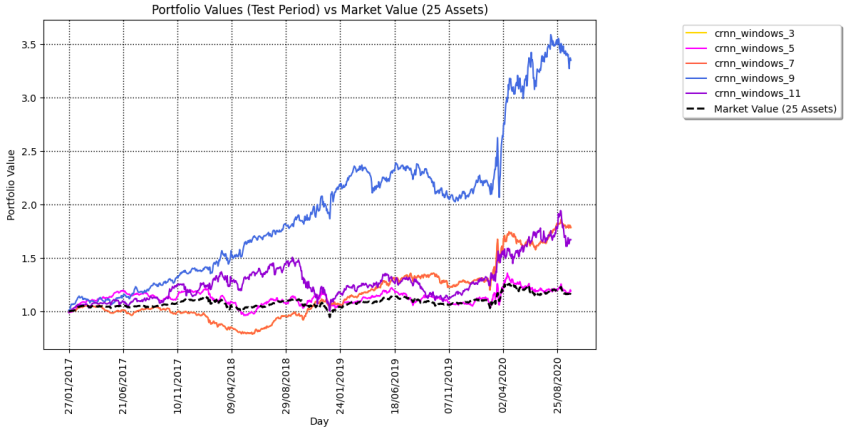

Figure 44: Portfolio Value Evolution for CRNN Architecture from DNN-AC

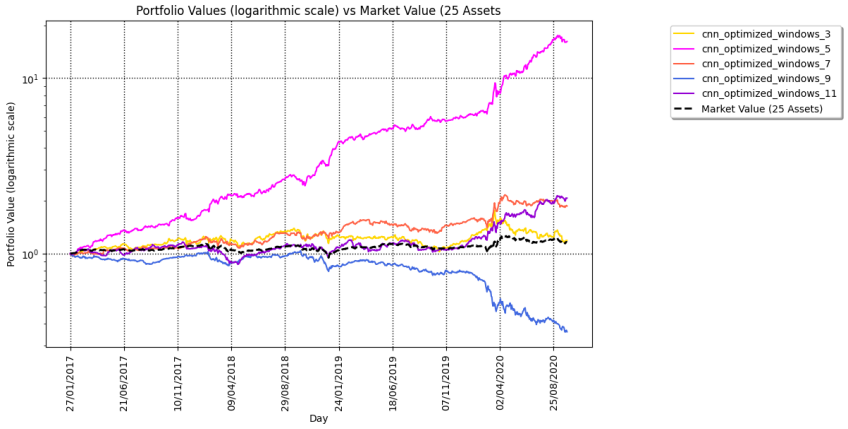

Figure 45: Portfolio Value Evolution for CNN Optimized Architecture from DNN-AC

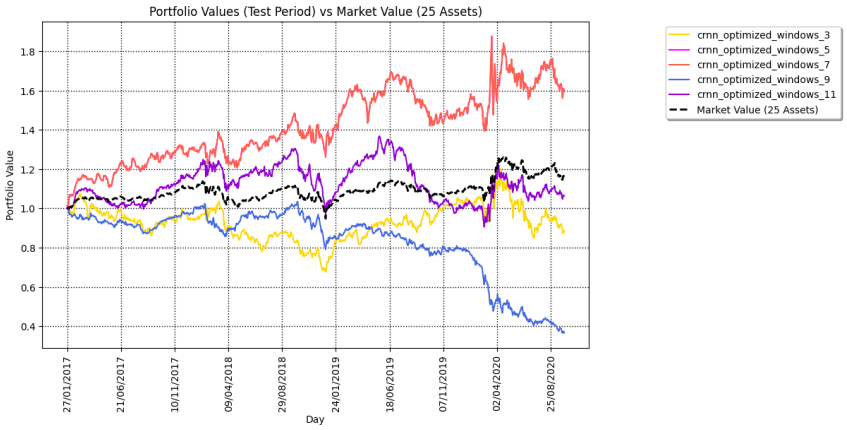

Figure 46: Portfolio Value Evolution for CRNN Optimized Architecture from DNN-AC

LSTM vs. LSTM Optimized: The LSTM model with a W3 stands out, achieving a remarkable portfolio value of 81.905, surpassing other configurations. In contrast, the LSTM Optimized models exhibit a more consistent performance across different window lengths, but none of them reach the heights of the aforementioned LSTM model. Notably, the LSTM Optimized with a W3 achieves a portfolio value of 40.793, which, while being impressive, is approximately half of its non-optimized counterpart. This disparity underscores the significance of the underlying architecture and training process, even before optimization.

CRNN vs. CRNN Optimized: The CRNN models generally underperform in comparison to the LSTM and CNN architectures. The best-performing CRNN model, with a window length of 9, achieves a portfolio value of 3.346, which pales when juxtaposed with the top LSTM results. The CRNN Optimized models further intensify our concerns. As anticipated from their tumultuous training phase, they yield unsatisfactory results.

CNN Optimized vs. CNN Classic: The CNN Optimized models show a mixed bag of results. While the model with a window length of 5 impresses with a portfolio value of 16.176, others like the one with window length 9 drastically underperform, depreciating to a value of 0.359. Comparing these with the CNN Classic results from BDDPG Experiment, we observe that the optimized models don't consistently outperform the classic models. For instance, the classic CNN model with a window length of 5 achieved a portfolio value of 9.650, which, though commendable, is surpassed by the optimized variant for the same window length.

In summary, while the LSTM architecture, particularly with W3 and W11, establishes itself as a frontrunner, the other architectures and their optimized variants present a more nuanced picture. Optimization doesn't always guarantee superior performance, as evident from the LSTM Optimized results. The CRNN models, both classic and optimized, raise concerns regarding their applicability in this domain. The mixed results from the CNN Optimized models emphasize the importance of careful model selection and hyperparameter tuning. *Remarks:* It's paramount to remember that the interpretability of these results is more pertinent when comparing amongst each other rather than against real-world market scenarios. As previously emphasized, our results are heavily embellished and far from being representative of a real market situation, due to strong assumptions about the market and the utilization of closing prices for entry and exit points, which is not feasible in practice. Our primary focus within this research is not on the precise returns or final portfolio values. Instead, we aim to compare these embellished results among themselves, considering the various network architectures and state space representations

that we employ.

Model	Portfolio Value	Avg Daily Return %	Sharpe Ratio	Sortino Ratio	Max Drawdown
LSTM_window3	**81.905**	**0.4866**	**35.2214**	**65.5129**	**12.2596**
LSTM_window5	1.597	0.0589	4.2595	5.5230	21.2412
LSTM_window7	1.905	0.0787	5.7817	8.4996	29.2550
LSTM_window9	1.731	0.0648	6.0503	7.8227	31.7150
LSTM_window11	14.743	0.2999	21.3370	31.6539	**11.7489**
CRNN_window3	1.785	0.0691	6.4011	8.7301	25.4936
CRNN_window5	1.191	0.0259	2.2562	2.8173	20.9996
CRNN_window7	1.785	0.0691	6.4011	8.7301	25.4936
CRNN_window9	3.346	0.1386	10.1912	13.9712	21.2413
CRNN_window11	1.671	0.0704	4.2119	5.6252	32.2058
CNNOpti_window3	1.189	0.0267	2.0530	2.6736	34.3079
CNNOpti_window5	**16.176**	**0.3139**	**20.4878**	**30.5430**	16.6238
CNNOpti_window7	1.896	0.0765	5.9570	8.3966	16.5274
CNNOpti_window9	0.359	-0.1013	-7.4492	-9.6488	**65.4803**
CNNOpti_window11	2.072	0.0877	6.3396	9.1815	25.4564
LSTMOpti_window3	**40.793**	**0.4146**	**26.0964**	**38.2921**	16.5498
LSTMOpti_window5	21.905	0.3452	22.9143	37.7208	14.2269
LSTMOpti_window7	1.785	0.0691	6.4011	8.7301	25.4936
LSTMOpti_window9	1.221	0.0255	2.6233	3.8497	32.8436
LSTMOpti_window11	1.537	0.0545	4.2262	5.7176	31.9794
CRNNOpti_window3	0.887	-0.0010	-0.0621	-0.0870	37.1948
CRNNOpti_window5	1.597	0.0589	4.2594	5.5228	21.2413
CRNNOpti_window7	1.597	0.0589	4.2597	5.5232	21.2412
CRNNOpti_window9	0.366	-0.0995	-7.4573	-9.6574	**64.6943**
CRNNOpti_window11	1.065	0.0160	1.1810	1.6020	33.7867
Market Value (25 Assets)	1.160	0.0183	3.1334	2.5726	16.7101

Table 9: Summary of Key Metrics from DNN-AC

6.2.2.1 In-Depth Analysis of the LSTM Model

The LSTM model with a window length of 3 consistently stood out in our evaluations, showing impressive results across various metrics. In this section, we delve deeper into this model's behaviors and strategies. Through an intricate examination, we hope to unveil the underlying mechanisms that led to its success. Such insights are pivotal as they not only provide clarity regarding the model's strengths but also offer direction for future refinements.

Asset Distribution with Portfolio Value: In Figure 47, we overlay the primary assets chosen by the model at each step onto the portfolio value graph. This choice is made by selecting the asset with the highest weight at each decision point. Concurrently, we also present a graph showcasing the normalized values of these major assets. From this, several observations emerge:

- *Diversification Strategy:* The model exhibits a penchant for a measured diversification approach. It seems to oscillate among a selected few stocks, suggesting a deliberate strategy. The stock being: APPLE, BAC, HD, JPM, MSFT. One may observe that we find again BAC's stock, as it was the case in the In-Depth Analysis of the CNN architecture with window length 5.
- *Asset Concentration:* Over time, certain assets gain prominence in the portfolio. This could be attributed to the model's capability in discerning these assets' potential for higher returns or stability during tumultuous market phases.
- *Correlation with Portfolio Value:* An increase in portfolio value is discernible when the model gravitates towards specific assets. This hints at the efficacy of the model's asset selection strategy.

- *Loss Mitigation:* The model adeptly navigates through market downturns, as evidenced during turbulent periods like the onset of the COVID-19 pandemic. It not only cushions the portfolio from severe losses but also capitalizes on subsequent recovery phases.

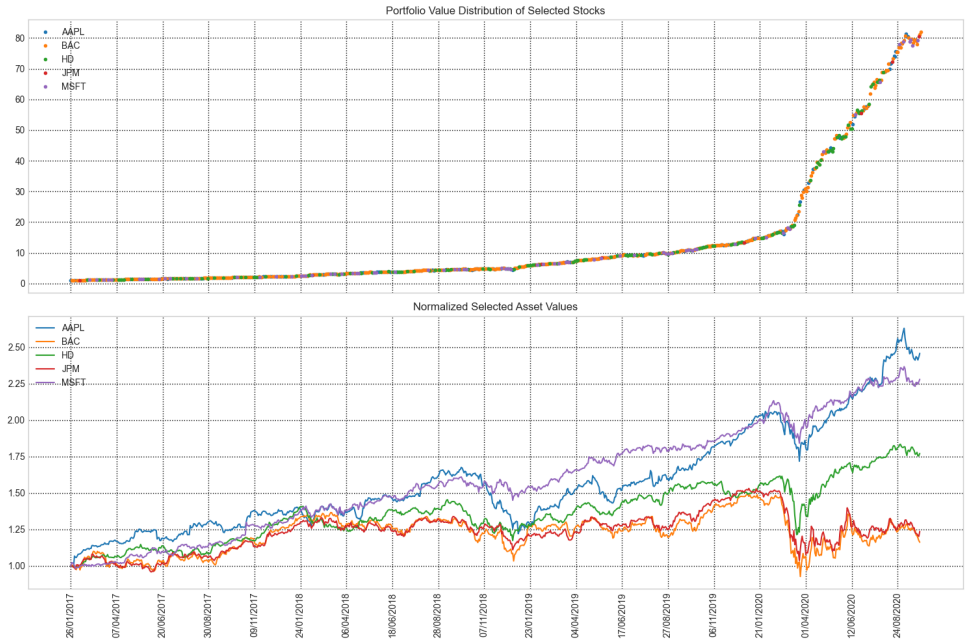

Figure 47: LSTM W3 & Closing Prices: Asset Distribution with Portfolio Value Over Testing Time

Frequency Distribution of Assets Selected by the Agent: Figure 48 depicts the frequency distribution of assets chosen by the agent. It's noteworthy that the top assets aren't necessarily the best performers over the testing phase. This underscores the agent's prowess in leveraging these assets' behavioral patterns to yield returns. However, the model also took advantage of outperforming assets such as APPL and MSFT.

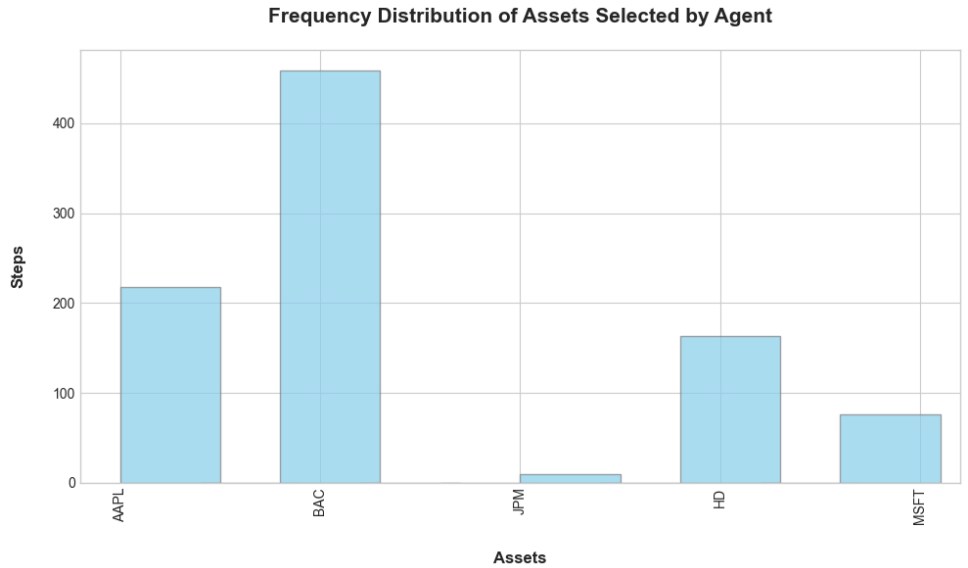

Figure 48: LSTM W3 & Closing Prices:: Frequency Distribution of Assets Selected by the Agent

Detailed Portfolio Allocation over a selected time period: Figure 49 amalgamates var-

ious plots within a specified interval. It details the normalized asset values, their log returns, agent-assigned asset weights, and the normalized portfolio value. The insights derived include:

- *Core Asset Selection:* The model consistently veers towards a set of assets, indicating a strong preference or strategy rather than taking advantage of the 25 available assets.
- *Diversification Pulses:* The agent occasionally opts for broader diversification, possibly as a hedge against market volatility.
- *Focused Allocation:* A distinct pattern emerges where the model allocates maximum weight to singular assets, possibly aiming for maximized returns from specific market movements. Indeed, in the subset period, we always allocate the entire portfolio value to one asset only.

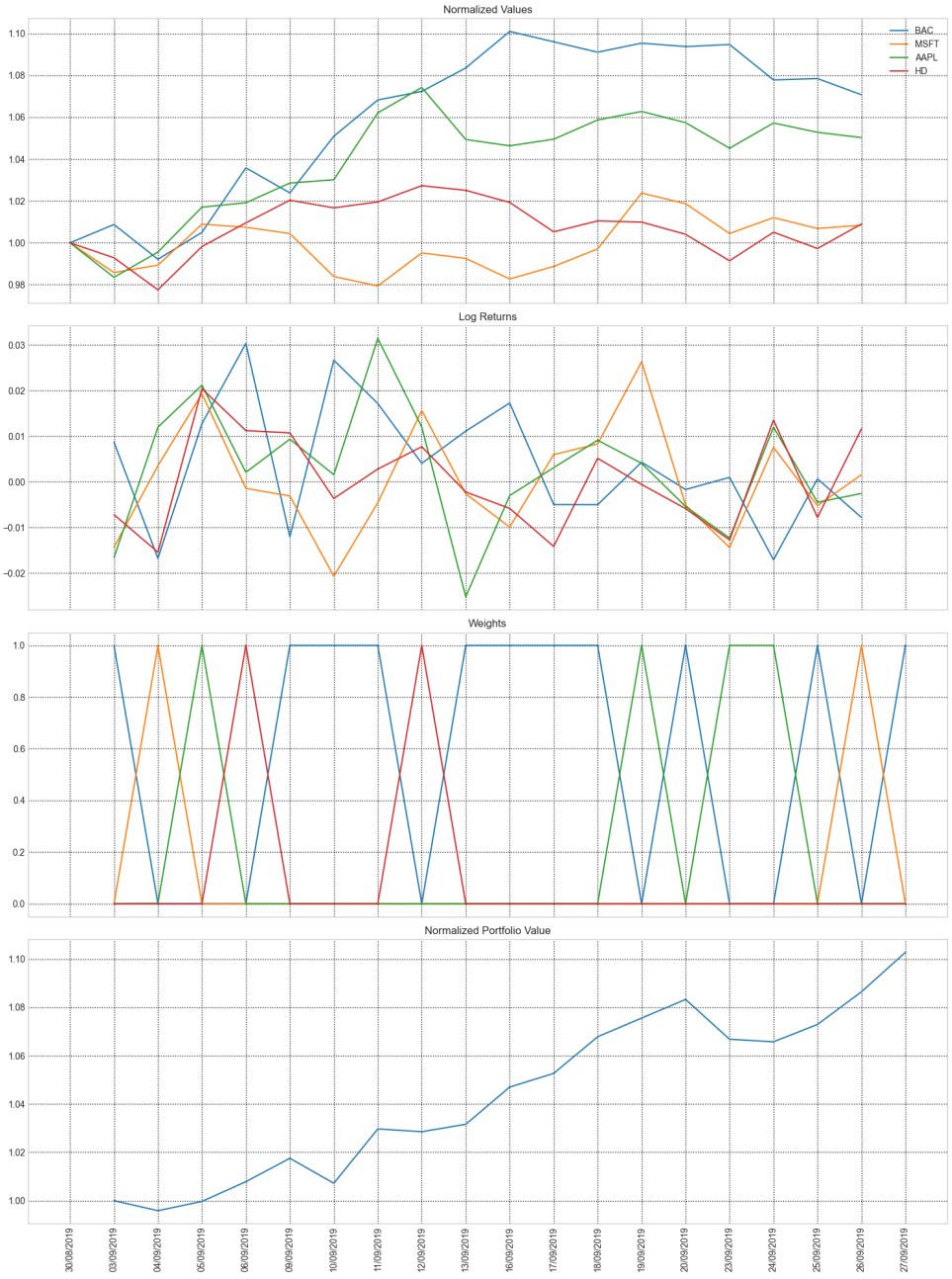

Figure 49: LSTM W3 & Closing Prices: Portfolio Analysis over a selected time period

6.3 DATA-ENH

In DATA-ENH Experiment, we pivot our focus towards understanding the profound influence of state space on the performance of the DPPG framework. The state space serves as the linchpin, defining the environment for all DRL models and setting the stage for the agent's learning journey. By introducing OHLC data and Technical Indicators into our state space, we venture into a richer informational landscape, seeking to provide our agent with a more holistic view of the market dynamics. However, as we broaden the horizons of our state space, we grapple with the augmented computational intricacies and the challenges of effectively processing this multi-faceted data. The quintessential question we aim to answer is: Does the enhanced state space, despite its complexities, lead to improved model performance? To ensure a focused evaluation, we retain the actor-critic DNN architecture from BDDPG Experiment as our constant, thereby primarily spotlighting the effects of our state space modifications.

6.3.1 Enhanced State Space using OHLC Data

In the following sections we will anaylse the results obtained when using OHLC data to enhance to state space and the basic CNN architecture for the actor-critic networs within the DPPG framework.

6.3.1.1 Training Analysis using OHLC Data

Upon close inspection of Figure 50, it's evident that models with W9 and W11 converge quite quickly, similar to our observations in BDDPG Experiment. However, as seen is DNN-AC Experiment, one may observe that the model W11 seems to start a new learning phase after 150 epoch as the evolution of Qvalue start increasing again which might indicates a good exploration phase. Interestingly, the models W3 and W5 to achieve higher Q-values in this training than when using closing prices only and also to converge in a more distinct way. This could indicate potentially improved performances during testing. The evolution of the Qvalue for the model using a W7 is very similar to the one obtained during the BDDPG Experiment. Overall, all models from DNN-AC Experiment seems to have a better training phase than the one obtain in the BDDPG Experiment, so one may expect to obtain better results during the testing phase.

Shifting our attention to Figure 51, the rewards seem to converge for most models and to have an upward trends during learning which is a good sign. But there's a significant distinction in the reward curves for models W9 and W11 when compared to the others. This behavior aligns with the Q-value observations, suggesting that these models might not perform as well during testing as models W3, W5 and W7. However, compared to the result obtain during BDDPG Experiment, we once again observe an improvement for each model, which could let us think that using OHLC data may have impact positively the training phase.

Taking into consideration the results of BDDPG Experiment (Table 8), and based on the Q-values and rewards during training, one might expect models W3, W5, and possibly W7 to outperform the other models during testing when using OHLC data. Particularly, the model W5, which demonstrated strong results in BDDPG Experiment, is anticipated to perform well, given its noticeable learning progression during the training process.

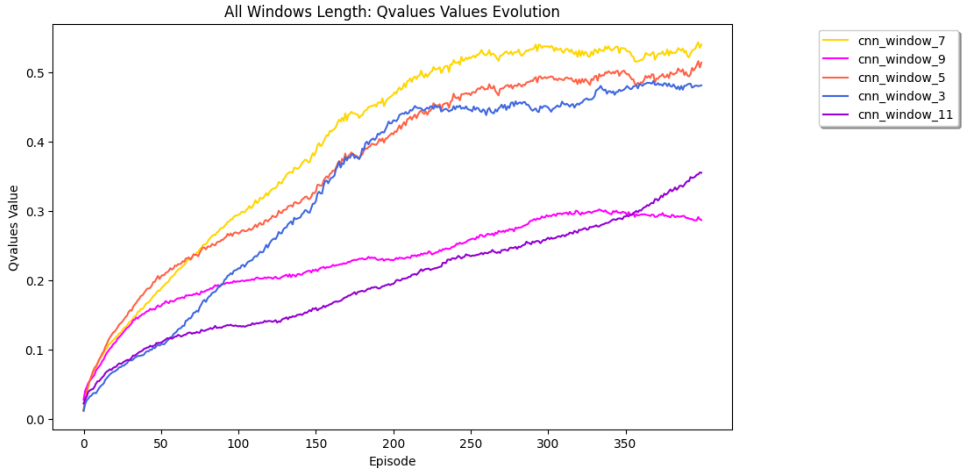

Figure 50: Q-Value Evolution during Training using OHLC Data

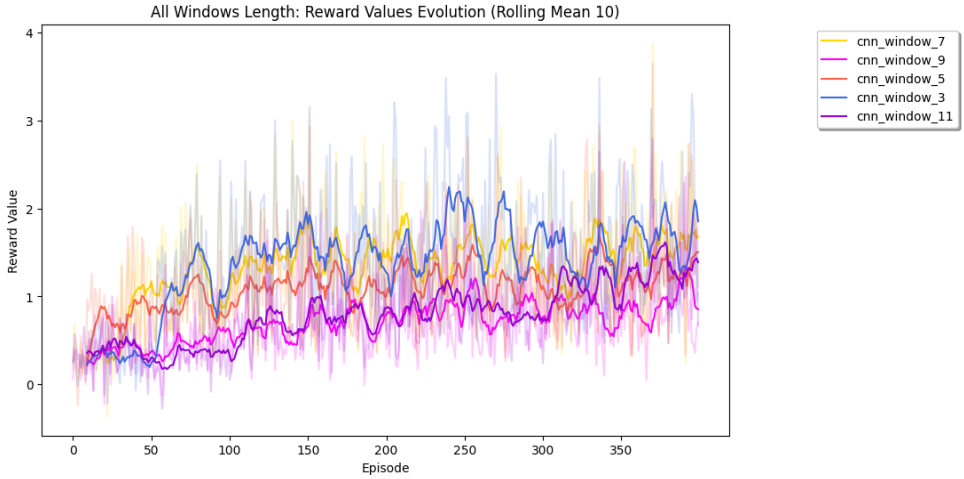

Figure 51: Rewards Evolution during Training using OHLC Data

6.3.1.2 Testing Analysis using OHLC Data

Upon completing the training phase, we transitioned to the testing phase, wherein the true capabilities of our models are revealed. Here, we gauge how well the trained agents adapt to unseen data and the resultant portfolio values they generate.

From Figure 52, we can draw several observations:

- The model with a window length of 3 appears to be the standout performer, consistently achieving a high portfolio value, surpassing its performance in BDDPG Experiment.
- Models with window lengths of 7 and 11 also perform reasonably well, with the latter slightly outdoing its performance from BDDPG Experiment.
- The model with a window length of 5, although demonstrating good learning during the training phase, couldn't replicate its outstanding performance from BDDPG Experiment.
- As anticipated from the training analysis, the model with a window length of 9 struggles significantly, underlining the concerns raised during its training phase.

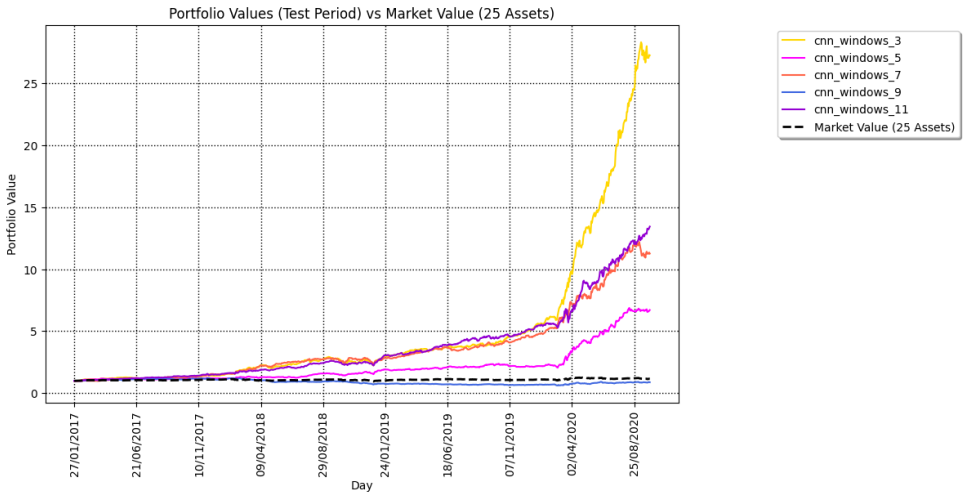

Figure 52: Portfolio Value Evolution during Testing using OHLC Data

Comparing the results from Table 10 for DATA-ENH Experiment with those from BDDPG Experiment, we notice:

- The model with a window length of 3 has shown a remarkable improvement, moving from a portfolio value of 4.773 in BDDPG Experiment to a whopping 27.232 in DATA-ENH Experiment. This validates our assumption from the training analysis that this model would be a standout performer.
- While the model with a window length of 5 was the star in BDDPG Experiment, it doesn't reach the same heights in DATA-ENH Experiment but still maintains a respectable portfolio value of 6.709.
- The models with window lengths of 7 and 11 also show improvements, underlining the potential benefits of incorporating OHLC data into the state space.
- As expected, the model with a window length of 9 underperformed significantly, which was consistent with its non-convergent behavior during the training phase.

Besides the Portfolio Value, other key metrics like Max Drawdown and Sharpe Ratio also play a pivotal role in gauging performance. A closer look reveals that models with window lengths of 3, 5, 7, and 11 maintain a Sharpe Ratio above 15, demonstrating their capability in generating good returns relative to the risk undertaken. Moreover, the Max Drawdowns remain relatively contained for these models, indicating resilience during downturns.

In conclusion, expanding the state space to include OHLC data has proven beneficial for most models, especially for the model with a window length of 3. The inclusion of this data likely provided the agent with a richer context, enabling it to make more informed decisions. The results reiterate the importance of the state space in the DRL environment and how crucial choices related to data can heavily influence performance outcomes.

Window length	Portfolio Value	Avg Daily Return %	Sharpe Ratio	Sortino Ratio	Max Drawdown
3	27.232	0.3711	23.5358	36.9303	23.6082
5	6.709	0.2155	15.2576	25.5064	13.4160
7	11.256	0.2729	18.9163	29.3575	18.5757
9	0.888	-0.0037	-0.2686	-0.3885	48.5872
11	13.433	0.2908	20.6676	28.6744	16.0710
Market Value (25 Assets)	1.160	0.0183	3.1334	2.5726	16.7101

Table 10: Summary of Key Metrics from DATA-ENH (OHLC Data)

6.3.2 Enhanced State Space using Technical Indicators

As for the other experiment, we will start by analyzing the training process, focusing on the evolution of Q-values and rewards. Then, we will move on to the testing analysis, where we will compare the portfolio values, Sharpe ratios, and other key metrics to draw conclusions about the model's performance using Technical Indicators compare to the one using closing price and OHLC data.

6.3.3 Training Analysis using Technical Indicators

Given the significance of the state space in shaping the learning environment, expanding the state space to include technical indicators is expected to provide the DRL agent with more actionable insights. However, this comes at the cost of increased complexity and potential overfitting risks.

By observing the Q-value evolution (Figure 53), we can gauge the agent's progress in estimating the potential value of its actions. A steadily increasing Q-value trend indicates that the agent is progressively optimizing its policy.

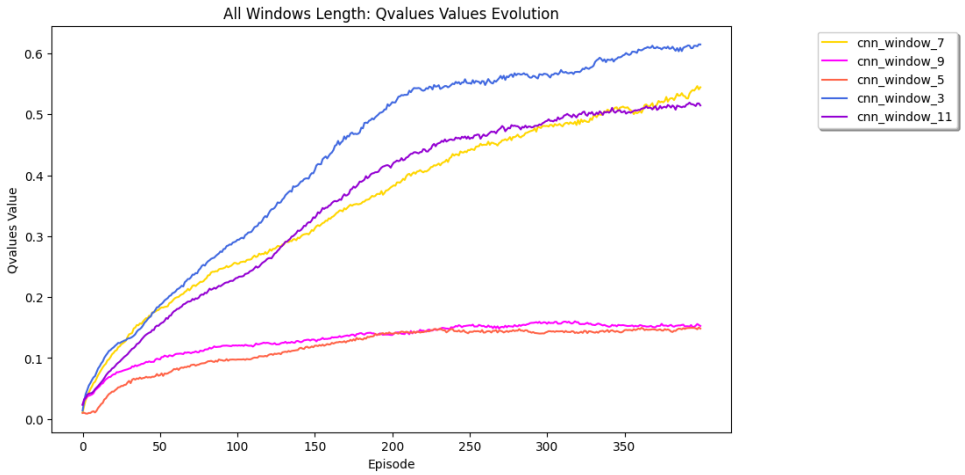

Figure 53: Q-Value Evolution during Training with Technical Indicators Data from

From the given Q-value evolution plot, the following observations can be made:

- Models with window lengths of 9 and 5 seem to converge rapidly, suggesting early stabilization in their learning. This might hint at a potential risk of overfitting, especially if the training performance doesn't translate similarly during testing. It also indicates a lack of exploration, despite the use of OU noise within our framework.
- The model with a window length of 7 displays a gradual increase in Q-values, indicating steady learning, which might be promising when applied to unseen data.
- The models with window lengths of 3 and 11 also show a promising upward trend. However, their learning curves also show a clear convergence at then end of the training phase which could indicate overfitting or the end of the learning process early because of the lack of exploration.

The rewards evolution (Figure 54) provides insights into the actual returns the agent is achieving during its training episodes.

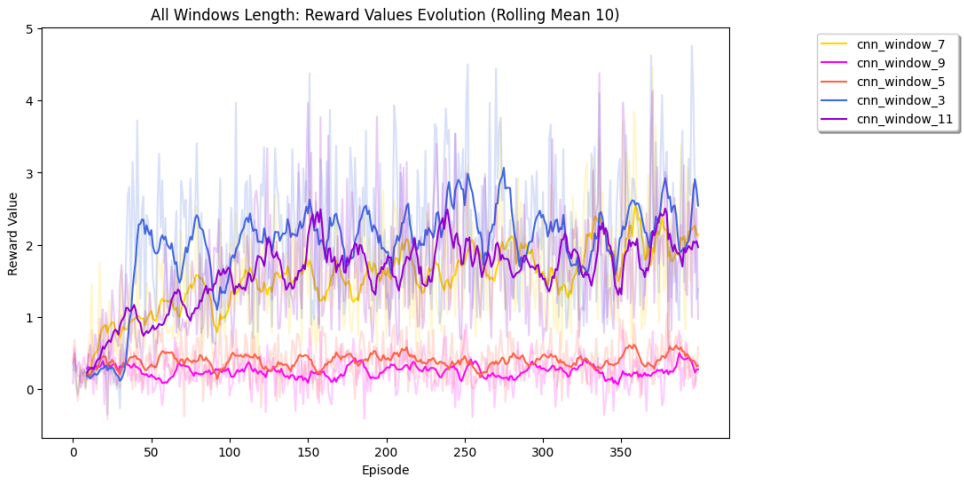

Figure 54: Rewards Evolution during Training with Technical Indicators Data from DATA-ENH

A few observations from the rewards plot:

- Models with window lengths of 9 and 5 demonstrate low volatility in their reward trajectories. This might suggest stability in their learning, but the consistent reward magnitude, limited to around 1, without any clear upward or downward trend, may raise concerns about potential overfitting or insensitivity to important patterns in the training data.
- On the other hand, models with window lengths of 11 and 3 exhibit high volatility, hinting at their sensitivity and adaptability to the nuances and complexities of the training data. Their clear upward trend in rewards is an encouraging sign, indicating that despite the high variance in rewards, they are progressively making better decisions and learning effectively from the environment.
- The model with a window length of 7 seems to offer an optimal blend of the characteristics observed in the other models. With its relatively low volatility, it suggests stable learning and potentially good generalization capabilities. Furthermore, its consistent upward trend in rewards indicates effective learning, making it a strong candidate for achieving good performance on unseen data.

Comparing these observations with the results from BDDPG Experiment and the first part of DATA-ENH Experiment (using OHLC data), we notice:

- The models trained on Technical Indicators appear to have a more pronounced learning curve, especially for window lengths of 3, 11, and 7. This suggests that the inclusion of technical indicators might be offering additional insights that aid the learning process.
- The rapid convergence observed in models with window lengths of 9 and 5 in the Q-values plot, combined with their volatility in the rewards plot, raises concerns about their ability to generalize. This observation is similar to the one made for the CNN model using closing prices in BDDPG Experiment for the window length 9, however it differs for the model with window length 5 which used to be the best one and does not seems to be able to replicate his performance this time using technical inidcators.

In conclusion, while expanding the state space with technical indicators shows promise, especially for window lengths of 3, 11, and 7, it's essential to validate these observations with testing results. The testing analysis will help in determining the model's capability to generalize its learning to new, unseen data and offer a more comprehensive evaluation of its performance.

6.3.3.1 Testing Analysis using Technical Indicators

Post-training, our focus shifts to the testing phase. This is the stage where our models truly demonstrate their learning by adapting to unseen data. We gauge their efficiency based on the resultant portfolio values and other key metrics. The results are juxtaposed with those from BDDPG Experiment (using closing prices) and the OHLC data from the first part of DATA-ENH Experiment to draw comprehensive conclusions.

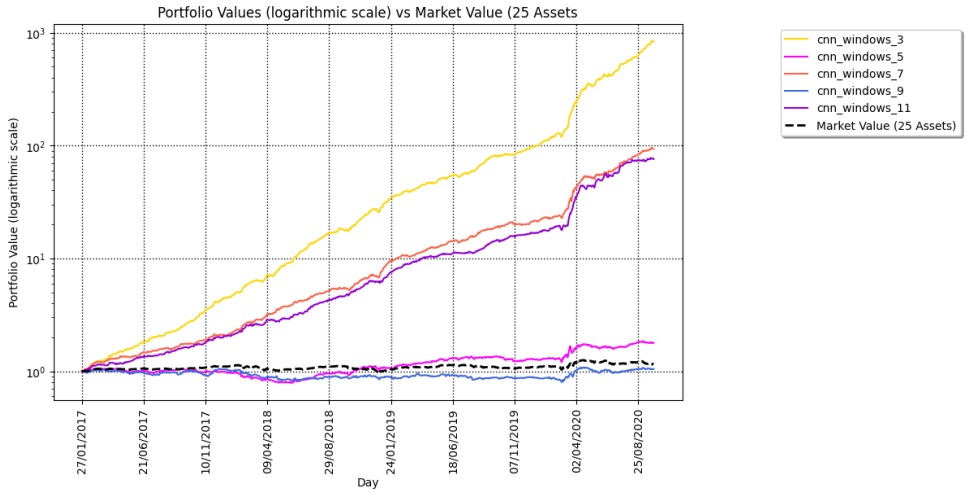

Figure 55: Portfolio Value Evolution during Testing with Technical Indicators Data from DATA-ENH

A few observations from Figure 55:

- The model with a window length of 3 displays an exceptionally high portfolio value. However, this value appears to be anomalously high and is currently under in-depth review to determine if an error might have occurred. Preliminary investigations have not identified any issues, but the result remains unusually high.
- Models with window lengths of 7 and 11 perform commendably, closely followed by the model with a window length of 5. This aligns with our training analysis where these models showed promise.
- The model with a window length of 9, as expected from the training analysis, struggles to achieve a substantial portfolio value, reinforcing the concerns raised during its training phase.

Comparing the metrics from Table 11 for DATA-ENH Experiment with those from BDDPG Experiment and the OHLC data of DATA-ENH Experiment, we observe:

- The model with a window length of 3, despite the outstanding results, is still under scrutiny. If the results stand after our in-depth analysis, this model would have achieved a remarkable improvement.
- The models with window lengths of 7 and 11 show enhancements in their performance metrics, especially when compared to the same window lengths from BDDPG Experiment. This indicates that the inclusion of technical indicators in the state space might be furnishing the agent with valuable insights that enhance its decision-making.
- The model with a window length of 5, while not matching its performance in BDDPG Experiment, still yields a respectable outcome.

- As anticipated from the training analysis, the model with a window length of 9 underperforms, further underlining its struggles during the training phase.

Diving deeper into other essential metrics like Max Drawdown and Sharpe Ratio:

- Models with window lengths of 3, 5, 7, and 11 consistently maintain commendable Sharpe Ratios, showcasing their potential in delivering good returns relative to the risk undertaken.
- Max Drawdowns are relatively contained for most models, indicating resilience during potential market downturns. The model with a window length of 9 is a notable exception, which might raise concerns about its ability to manage losses.

In conclusion, the inclusion of technical indicators in the state space has brought about varied results. While some models, particularly with window lengths of 7 and 11, demonstrate clear enhancements, others, like the one with window length 9, continue to struggle. The anomalously high performance of the model with a window length of 3 remains a topic of ongoing investigation.

Window length	Portfolio Value	Avg Daily Return %	Sharpe Ratio	Sortino Ratio	Max Drawdown
3	837.352	0.7354	55.4174	111.6698	7.8412
5	1.785	0.0691	6.4011	8.7301	25.4936
7	93.598	0.5010	38.2605	67.2571	6.9427
9	1.047	0.0096	0.9196	1.4093	23.5714
11	76.089	0.4779	35.9858	70.9207	8.9639
Market Value (25 Assets)	1.160	0.0183	3.1334	2.5726	16.7101

Table 11: Summary of Key Metrics from DATA-ENH (Technical Indicators)

6.4 Impact of Neural Networks Architecture and Enhanced State Space on The Results

In the rapidly evolving domain of portfolio management, the efficacy of reinforcement learning models can be influenced by several factors. Among these factors, the choice of the neural network architecture and the richness of the input data play pivotal roles. This section delves into the comparative analysis of different neural network architectures, from traditional CNN to more intricate designs like LSTM and CRNN. Furthermore, we also explore the impact of enhancing our state space by incorporating diverse financial indicators such as OHLC data and technical indicators. By juxtaposing these variables, we aim to discern the optimal configurations that yield the most promising results in portfolio optimization using the DDPG framework.

6.4.1 Observation of Key Metrics Between Each Group

To provide a granular insight into the performance of our models, we embark on a meticulous examination of key metrics across various configurations. This sub-section zeros in on metrics such as Portfolio Value, Sharpe Ratio, and Max Drawdown, as illustrated in Tables 12 and 13. By tabulating these metrics and further visualizing them through the Spider Charts in Figures 56 and 57, we offer an unambiguous representation of how each model performs in real-world scenarios. The data showcased not only elucidates the absolute performance but also offers insights into the consistency and volatility of returns, thereby facilitating a comprehensive assessment of the employed strategies.

Neural Network Architectures:

- **LSTM**: The LSTM model, especially its optimized version (*LSTMOpti*), stands out in terms of performance. Notably, its high average Portfolio Value and Sharpe Ratio are commendable. However, a closer look at the standard deviations reveals considerable variability, especially in Portfolio Value. This suggests that while the LSTM models,

on average, perform well, there might be inconsistencies in their results across different window lengths or datasets.

- **CNN**: The basic CNN model yields moderate results on average. However, its optimized counterpart (*CNNOpti*) appears to falter. Interestingly, the standard deviation for the Portfolio Value for *CNNOpti* is significantly high, indicating a wide spread of results. This suggests that while there might be instances where *CNNOpti* performs exceptionally well, there are also scenarios where it might underperform.
- **CRNN**: Both the basic and optimized CRNN models generally depict lower mean values across all metrics. However, their relatively low standard deviations, especially in Portfolio Value, indicate consistent (though subpar) performance. The consistency suggests that the CRNN models might be stable but need refinement to compete with the likes of LSTM in terms of returns.

Enhanced State Space:

- **Technical Indicators**: The remarkable performance of models using Technical Indicators as state space is undeniable. However, the high standard deviation, especially in Portfolio Value, indicates a broad range of outcomes. While on average, the results are impressive, there might be instances where the performance isn't as stellar, which warrants caution.
- **OHLC Data**: While OHLC data doesn't reach the heights of the Technical Indicators, its performance is still commendable. The standard deviations are relatively lower than those of the Technical Indicators, suggesting more consistent outcomes when using OHLC data.
- **Closing Prices**: Being the simplest input, the Closing Prices unsurprisingly offer the least impressive mean metrics. However, their consistency, as indicated by the low standard deviations, provides a reliable baseline for our experiments. It's a testament to the importance of having a foundational benchmark against which enhancements can be measured.

In summary, delving into both the average performances and their respective standard deviations provides a multi-faceted perspective on the strategies employed. When evaluating the results holistically, combining both mean and consistency (as gauged by the standard deviation):

- **Neural Network Architectures**: The *LSTM* models, especially the optimized version (*LSTMOpti*), emerge as the frontrunners. Their superior average performance combined with a reasonable spread of results suggests a blend of high returns and relative consistency. In contrast, the *CRNN* models, both basic and optimized, depict a consistent yet underwhelming performance. This outcome is somewhat unexpected, and while the results from this experiment are not promising for CRNN, we remain hopeful that SYNERGY Experiment might showcase its potential in a different light.
- **Enhanced State Space**: The *Technical Indicators* reign supreme, with the highest mean values across the board. However, its high standard deviation, especially in Portfolio Value, indicates some variability in outcomes. *OHLC Data*, though not as robust in terms of mean performance as the Technical Indicators, offers a good balance of performance and consistency, making it a reliable choice for portfolio management tasks.
- **Optimized Architectures**: The results suggest a mixed bag when it comes to the efficacy of optimized architectures. For instance, while *LSTMOpti* exhibits commendable results, *CNNOpti* and *CRNNOpti* don't consistently outshine their basic counterparts. This raises questions about the universal applicability of complexity optimizations, indicating that

enhancements need to be carefully tailored to the specific requirements of the task at hand.

Model	Portfolio Value		Sharpe Ratio		Max Drawdown (%)	
	Avg	Std Dev	Avg	Std Dev	Avg	Std Dev
CNN	5.27	3.72	11.96	6.49	23.18	9.94
CNNOpti	4.34	6.65	5.48	10.06	31.68	20.28
CRNN	1.96	0.82	5.89	2.96	25.09	4.54
CRNNOpti	1.10	0.52	0.44	4.80	35.63	17.78
LSTM	20.38	34.85	14.53	13.49	21.24	9.28
LSTMOpti	13.45	17.65	12.45	11.14	24.22	8.59

Table 12: Average and Standard Deviation of Key Metrics for Different Neural Network Architectures using Closing Prices

State Space	Portfolio Value		Sharpe Ratio		Max Drawdown (%)	
	Avg	Std Dev	Avg	Std Dev	Avg	Std Dev
Closing Prices	5.27	3.72	11.96	6.49	23.18	9.94
OHLC Data	11.90	9.82	15.62	9.38	24.05	14.22
Technical Indicators	43.13	48.70	20.39	19.47	16.24	9.64

Table 13: Average and Standard Deviation of Key Metrics for Different Input Data Using Basic CNN Architecture

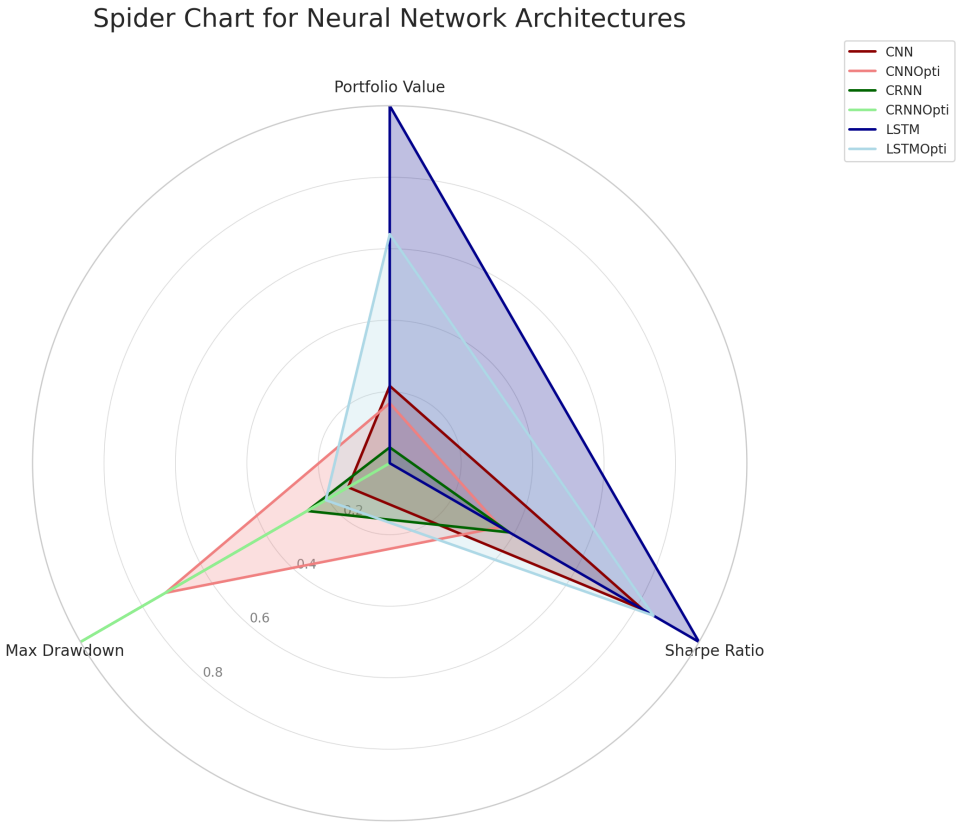

Figure 56: Spider Chart of the Key Metrics obtained from Different Neural Network Architecture

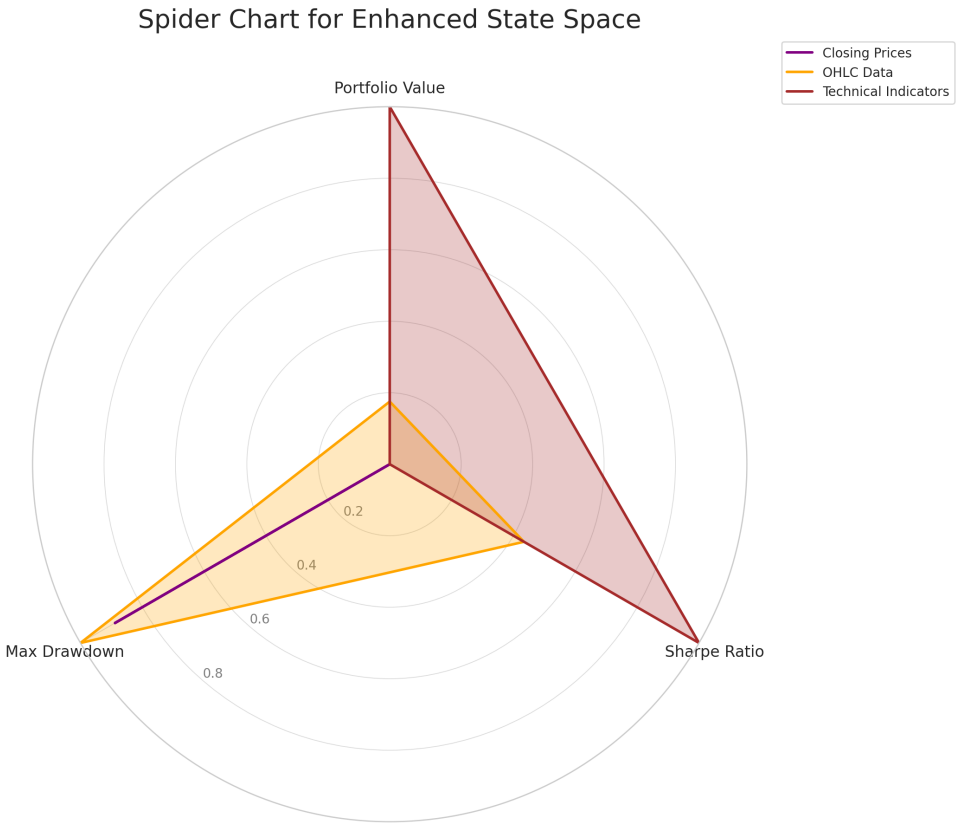

Figure 57: Spider Chart of the Key Metrics obtained using Different State Space

6.4.2 ANOVA Test

The Analysis of Variance (ANOVA) is a statistical test used to determine whether there are any statistically significant differences between the means of three or more independent (unrelated) groups. While the t-test is used to compare two means, ANOVA is used for comparisons involving more than two groups. The primary objective of ANOVA is to test for differences in group means by examining the variance of data points within groups compared to the variance between groups.

In our case, we utilize the ANOVA test to evaluate the performances of different Neural Network Architectures and various Enhanced State Spaces in portfolio optimization. By examining the key performance metrics across these categories, the ANOVA test helps determine if observed differences in metric values are statistically significant or can be attributed to random chance. This offers a more rigorous foundation for our evaluations, ensuring that our conclusions are not based merely on observed means but are statistically validated.

Metric	F-value	P-value
Portfolio Value	1.0878	0.3924
Sharpe Ratio	1.8061	0.1497
Max Drawdown	0.9262	0.4814

Table 14: ANOVA Test Results for Neural Network Architectures

Metric	F-value	P-value
Portfolio Value	2.5619	0.1221
Sharpe Ratio	0.5238	0.6063
Max Drawdown	0.5829	0.5746

Table 15: ANOVA Test Results for Enhanced State Space

The ANOVA tests conducted for both Neural Network Architectures and Enhanced State Space across different metrics (Portfolio Value, Sharpe Ratio, MDD) consistently showed that there aren't statistically significant differences in performance metrics as depicted in Table 14 and 15. This indicates that while there may be observed differences in the metrics' values, they are not statistically significant when considering the variability within each group. However, it's crucial to approach these results with caution, as the number of samples for each group was limited to the number of models per group, with only five samples corresponding to each of the window lengths tested.

6.5 SYNERGY

In SYNERGY Experiment, we integrate key findings from our previous studies to further optimize our DDPG framework. Based on the results from DNN-AC Experiment, the LSTM architecture demonstrated superior performance, positioning itself as a primary candidate for further evaluation. Additionally, the enhanced state space using Technical Indicators (TI) from DATA-ENH Experiment showed significant potential in improving the learning environment. Therefore, this experiment focuses on combining the LSTM model with TIs to evaluate the combined efficacy. Although the CRNN architecture did not meet expectations in earlier experiments, we hypothesize that with a larger feature space provided by TIs, CRNN might effectively capture temporal patterns and select relevant features. The primary objective of SYNERGY Experiment is to assess the performance of these combined approaches and derive actionable insights for portfolio optimization.

6.5.1 Training Analysis

For the fourth experiment, the focus lies on understanding the training dynamics when integrating complex architectures like LSTM and CRNN with an enhanced state space using technical indicators. The plots for Q-value and rewards evolution serve as diagnostic tools, revealing how the model progresses over episodes.

LSTM Model Analysis Starting with the LSTM model, the Q-value evolution is illustrated in Figure 58.

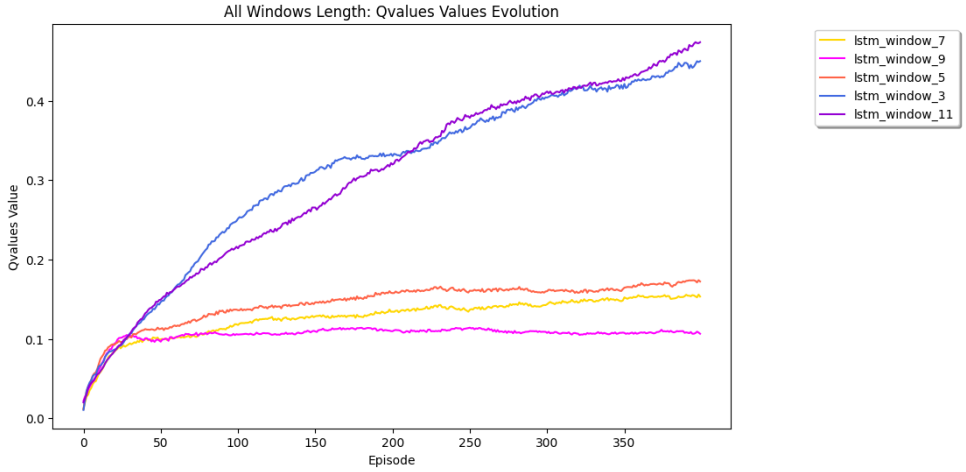

Figure 58: Q-Value Evolution during Training using LSTM with Technical Indicators Data

Key observations include:

- Models with window lengths of 3 and 11 exhibit steady upward trends and tend towards the highest Q-values, with no clear signs of convergence towards the end of the training.
- Models with window lengths of 7 and 5 converge rapidly within the first 50 episodes. Their Q-values plateau around a magnitude of 0.12, and this level is maintained from episode 50 to episode 400.
- The model with window length 9 initially converges in a manner similar to the models with window lengths of 7 and 5. However, post-convergence, it displays a downward trend persisting until the end of the training phase.

The rewards evolution for the LSTM model is presented in Figure 59.

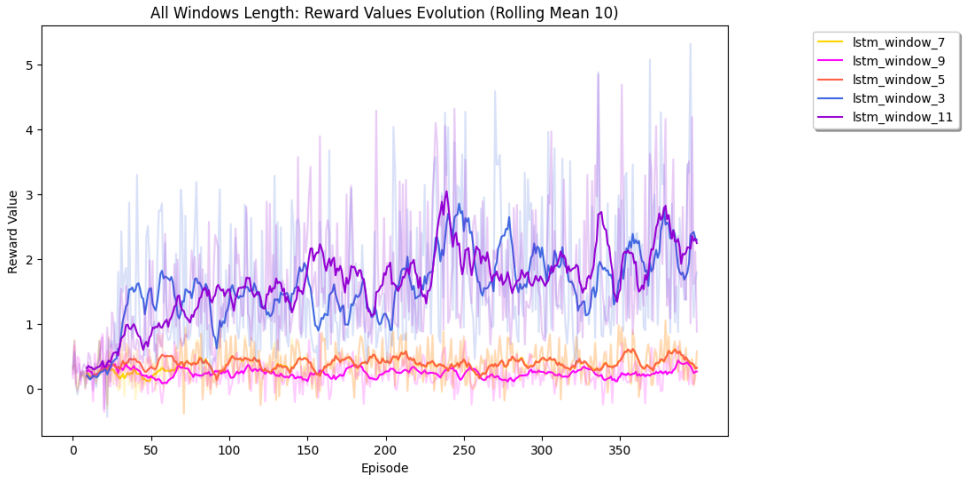

Figure 59: Rewards Evolution during Training using LSTM with Technical Indicators Data

Key observations from the rewards plot are:

- Models with window lengths of 3 and 11 exhibit similar behaviors, characterized by high volatility and an upward trend throughout the training phase. Both models culminate in reward values around 2 by the end of training.

- For the models with window lengths of 5, 7, and 9, there is no discernible upward or downward trend. They exhibit low volatility and maintain a subdued reward trajectory, with reward values not exceeding 1 at any point during training.

CRNN Model Analysis

For the CRNN model, the Q-value evolution is depicted in Figure 60.

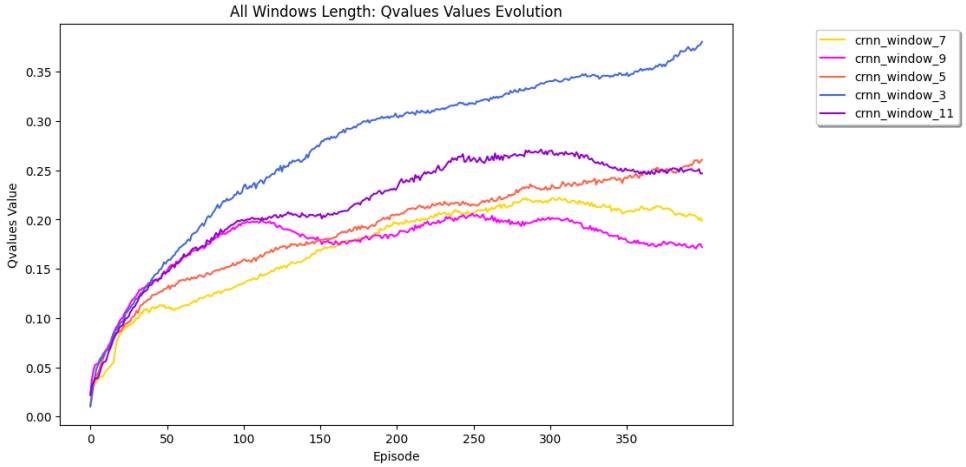

Figure 60: Q-Value Evolution during Training using CRNN with Technical Indicators Data

- The model with a window length of 3 displays a steady upward trend, culminating in the highest Q-value around 0.4. However, there's no clear sign of convergence, particularly in the last 10 episodes where the Q-value continues to rise.
- The model with window length 5 exhibits a behavior akin to the one with window length 3, albeit with a reduced magnitude, achieving Q-values around 0.25.
- Models with window lengths of 7, 9, and 11 show more irregular trends, with periods of both ascent and descent. By the end of the training, they achieve mid-range Q-values. Such erratic behavior could be a potential concern when it comes to the testing phase.

The rewards evolution for the CRNN model is presented in Figure 61.

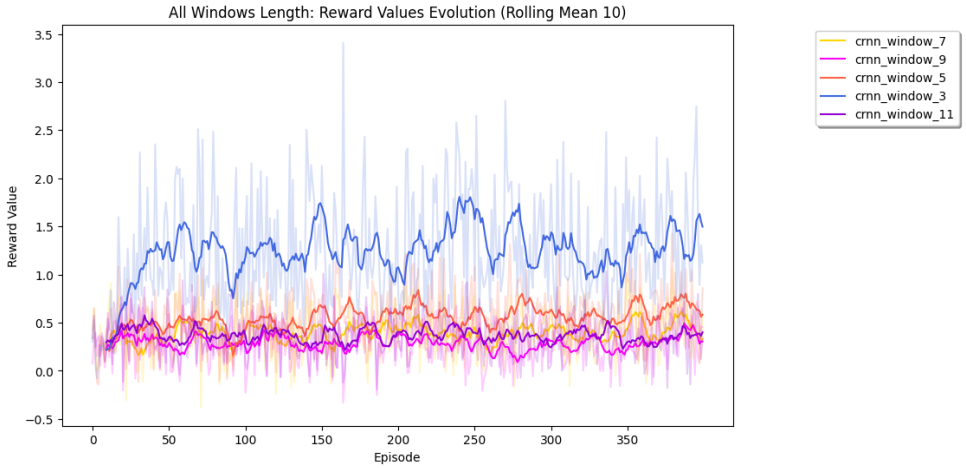

Figure 61: Rewards Evolution during Training using CRNN with Technical Indicators Data

Insights from the plot include:

- The model with window length 3 is distinguished by a clear upward trend, signifying effective learning. Morevoer it achieve the highest rewards value by the end of the learning phase and show some volatility during training
- Window lengths of 5 and 9 have subdued reward trajectories, with the latter being the least rewarding.
- The models with window lengths of 7 and 11 display some volatility, especially the latter. Both have upward trends, but the fluctuations may indicate sensitivity to training data nuances.

In conclusion, the training results from the LSTM model, particularly with window lengths of 3 and 11, indicate a potential for better performance compared to the CRNN model. The CRNN model with a window length of 3 still offers promising prospects, hinting at commendable outcomes during the testing phase. However, other configurations do not exhibit any specific trends, suggesting they might yield satisfactory but not exceptional results in the testing phase.

6.5.2 Testing Analysis

Upon transitioning to the testing phase, the main objective is to evaluate the adaptability of the trained agents to unseen data and measure the resultant portfolio values they produce. This step allows us to validate whether the patterns observed during training are consistent when the models are exposed to new data.

From Figure 62, we can infer the following about the LSTM model's performance:

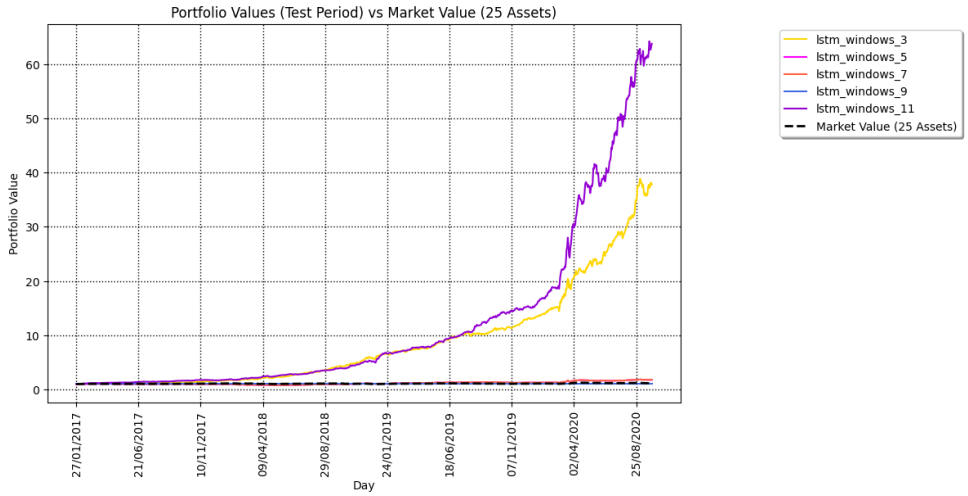

Figure 62: Portfolio Value Evolution during Testing using LSTM and Technical Indicators

- The model with a window length of 3 delivers a remarkably high portfolio value, consistent with our training observations, highlighting its adeptness at generalizing and adapting to new scenarios.
- The model with window length 11 also exhibits commendable performance, although it doesn't reach the peaks achieved by the model with window length 3. Still, it considerably surpasses the market's performance.
- Models with window lengths of 5 and 7, intriguingly, appear to adopt similar investment strategies. This overlap in strategy is curious but not implausible given the dynamic nature of markets.

- The model with window length 9 is an anomaly. During the entire testing period, it refrains from making any investment decisions, opting to retain the cash asset exclusively. Despite thorough investigation, we have yet to pinpoint the cause of this behavior.

Considering Figure 63, the CRNN model displays the subsequent patterns:

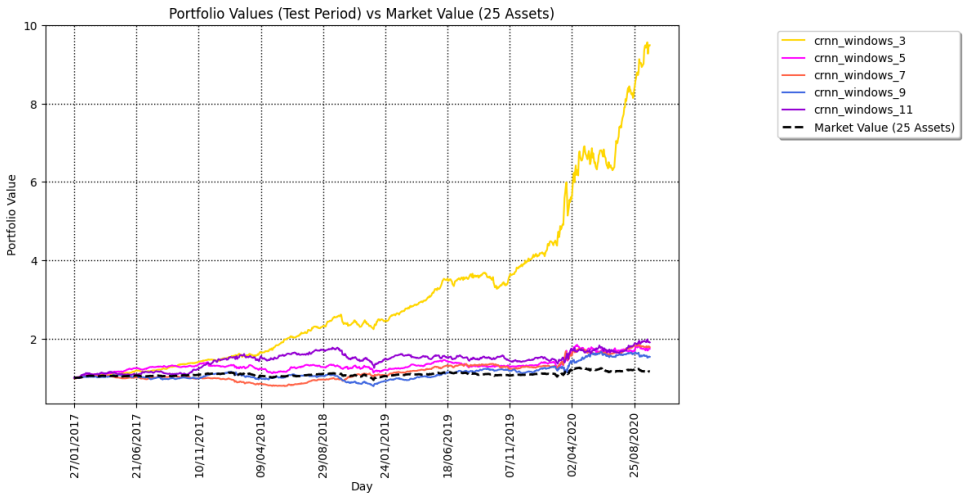

Figure 63: Portfolio Value Evolution during Testing using CRNN and Technical Indicators

- The standout model, in terms of performance, is the one with window length 3. It achieves a portfolio value significantly higher than the other configurations, albeit not matching its LSTM counterpart.
- The remaining configurations, including window lengths 5, 7, 9, and 11, manifest modest growth. Underscoring the notion that the CRNN architecture may require more fine-tuning or an alternative state representation for this specific problem when using the current state space.

Comparing the outcomes from Tables 16 and 17:

- The LSTM models, particularly with window lengths of 3 and 11, distinctly overshadow their CRNN equivalents in metrics like Portfolio Value, Sharpe Ratio, among others.
- The CRNN model with window length 3, although it exhibited promise during the training phase, couldn't maintain the momentum during testing. However, it still retains a respectable position among the CRNN variants.
- A majority of the CRNN configurations do not yield outstanding results, which is in line with our training observations.

In conclusion, when combined with the enhanced state space using Technical Indicators, the LSTM architecture emerges as the most potent setup, particularly for window lengths 3 and 11. Although the CRNN model harbors potential, its true capability may be unlocked through additional optimization or a different state representation. The outcomes underscore the criticality of selecting the appropriate model architecture and its SYNERGY with the chosen state space for achieving optimal portfolio performance.

Now, when comparing the architectures using Technical Indicators as showed in Table 18 against the previous results:

1. **CRNN with TI vs. Other Architectures**

   - The CRNN model, when enhanced with Technical Indicators, shows a noticeable improvement compared to when it was used with only closing prices. The average portfolio value increased from 1.96 (with closing prices) to 3.092 with TI. This suggests that the CRNN model, which initially seemed not to be the best performing architecture, is able to benefit from an enhanced state space, possibly due to its ability to process both spatial and temporal features more effectively.
   - However, even with this improvement, the CRNN model's performance with TI doesn't surpass other architectures like the basic CNN or LSTM when they are used with closing prices or OHLC data. This might indicate that while TI provides more comprehensive data, the architecture of CRNN might not be as effective in leveraging it as other architectures, or that other factors might be influencing its performance.

2. **LSTM with TI vs. Other Architectures**

   - The LSTM model with TI outperforms its counterparts from the previous experiments by a significant margin. With an average portfolio value of 21.211, it surpasses the LSTM model that used only closing prices. This showcases the potential of the LSTM architecture in effectively leveraging the rich information provided by Technical Indicators.
   - The Sharpe Ratio, which measures the risk-adjusted return, is also commendably high for the LSTM with TI, indicating that the model not only achieved higher returns but did so with a level of risk that was commensurate or even lower than other models.
   - When considering the maximum drawdown, the LSTM model with TI also shows resilience, with values that are competitive, if not better, than other models.

In summary, the utilization of Technical Indicators seems to be highly beneficial for the LSTM model, enhancing its performance and solidifying its position as the most promising architecture among those tested. On the other hand, while the CRNN model does show improvement with TI compared to using just closing prices, it still lags behind in overall performance. This suggests that while an enhanced state space is beneficial, the choice of architecture plays a pivotal role in how effectively this data can be leveraged.

Window length	Portfolio Value	Avg Daily Return %	Sharpe Ratio	Sortino Ratio	Max Drawdown
3	37.816	0.7354	55.4174	111.6698	7.8412
5	1.785	0.0691	6.4011	8.7301	25.4936
7	1.785	0.0691	6.4011	8.7301	25.4936
9	1.000	x	x	x	x
11	63.763	0.4595	33.2013	53.8885	13.2093
Market Value (25 Assets)	1.160	0.0183	3.1334	2.5726	16.7101

Table 16: Summary of Key Metrics using LSTM and Technical Indicators

Window length	Portfolio Value	Avg Daily Return %	Sharpe Ratio	Sortino Ratio	Max Drawdown
3	9.490	0.2518	20.4368	28.1099	14.2430
5	1.744	0.0688	5.3438	6.6706	20.0036
7	1.785	0.0691	6.4011	8.7301	25.4936
9	1.537	0.0545	4.2263	5.7177	31.9792
11	1.905	0.0787	5.7817	8.4995	29.2550
Market Value (25 Assets)	1.160	0.0183	3.1334	2.5726	16.7101

Table 17: Summary of Key Metrics using CRNN and Technical Indicators

Model with TI	**Portfolio Value**		**Sharpe Ratio**		**Max Drawdown (%)**	
	Avg	**Std Dev**	**Avg**	**Std Dev**	**Avg**	**Std Dev**
LSTM	21.211	25.173	22.291	21.296	18.588	8.795
CRNN	3.092	3.171	7.447	6.269	24.193	7.245

Table 18: Average and Standard Deviation of Key Metrics for LSTM and CRNN with Technical Indicators

7 Conclusion and Discussion

7.1 Conclusion

The pursuit of optimal financial portfolios has been a long-standing challenge in the realm of finance, and this research sought to explore new frontiers by integrating the power of DRL into portfolio optimization. The main research question of this thesis revolved around the efficacy of the DDPG framework, especially when enhanced by different NN architectures used for the actor-critic networks and state spaces using OHLC and Technical Indicators.

Our initial experiments using the DDPG framework with CNNs set a promising baseline, with certain window lengths outperforming traditional benchmarks. As we delved into the influence of different NN architectures, the LSTM model consistently emerged as a frontrunner. This architecture's prowess was further accentuated with the inclusion of Technical Indicators in the state space, solidifying its superiority among the evaluated architectures.

Contrastingly, the CRNN models presented a more nuanced outcome. While they showed signs of improvement with the inclusion of Technical Indicators, they did not reach the performance zenith achieved by their LSTM counterparts. This highlighted the pivotal role of architecture selection in harnessing the potential of an enhanced state space.

The research journey, punctuated by systematic experiments, has contributed new insights to the field:

- The DDPG framework, especially when combined with LSTM, holds significant promise for portfolio optimization.
- The choice of NN architecture is paramount, with LSTMs showcasing consistent superiority. Although CRNN theoretically promised significant results, we couldn't validate its effectiveness in practice.
- Enhancing the state space, particularly with Technical Indicators, can be beneficial, but its efficacy is intertwined with the chosen architecture. OHLC enhancement also show promising result and offered better stability over the different settings.

Reflecting on the research process, the challenges faced, from selecting appropriate architectures to integrating diverse state spaces, have been instrumental in refining our understanding of DRL in finance. The meticulous experimental design and rigorous evaluation have ensured that the insights gleaned are both robust and replicable.

However, it's imperative to reiterate that while the results are promising, they are embellished and should be interpreted in the context of the research setting. Real-world financial markets, with their myriad complexities, may present additional challenges. Yet, the findings of this thesis lay a robust foundation for future explorations, potentially paving the way for more sophisticated and adaptable DRL-based financial models.

In conclusion, this thesis has bridged the realms of finance and DRL, unveiling promising avenues for future research and applications. As technology continues to revolutionize the financial landscape, the integration of advanced DRL techniques, as explored in this research, could herald a new era of portfolio optimization. This project was truly enlightening for me. It let me use what I've learned from my years studying Computer Science, especially in the exciting area of DRL. It was also my first deep dive into financial research, and it has sparked my interest to continue studying in this direction.

7.2 Discussion and Future Work

Our exploration into the intersection of DRL and portfolio optimization has opened up various potential directions for further research. Here are some of the promising avenues we believe could significantly advance the field:

7.2.1 Alternative Algorithms and Frameworks

With the continuous advancements in DRL, sticking to just one method like DDPG might be limiting. Also, other algorithms could beneficiate our results as they leverage NN within ther architecture too. Some other algorithms could offer novel perspectives or better performance for the PO problem:

- *Trust Region Policy Optimization (TRPO):* TRPO ensures that policy updates don't deviate too much, thus providing stability in training [56]. Exploring its application in portfolio optimization might yield improved results.
- *Twin Delayed Deep Deterministic (TD3):* TD3 addresses some of the limitations of DDPG, especially the overestimation of Q-values [58]. Its application could refine our solutions further.

7.2.2 Multi-Agent Systems

The financial market is a complex system with multiple interacting participants. This complexity suggests that a single-agent model might not capture the full picture:

- *Multi-Agent Reinforcement Learning (MARL):* Adopting a MARL approach can allow us to simulate different trading strategies or specific market participants more effectively [10]. This method can help in understanding and navigating the intricacies of real-world financial systems.

7.2.3 Integration of New Data

As we proved, enhancing the state space using relevant data can drasticly increase the performance of our agent. Therefore, expanding the data sources used in our models can offer richer insights and potentially better predictions:

- *Incorporating Alternative Data:* Using non-traditional data like satellite imagery or social media trends can offer unique market insights [88]. For instance, satellite images can give us a glimpse into real-world activities, like store footfalls or crop yields, which can influence stock prices.
- *News and Sentiment Analysis:* News events have a significant impact on market movements. By integrating sentiment analysis derived from news articles or financial reports, our model might pick up on vital market signals [89].
- *Macroeconomic Indicators:* Stock prices often react to broad economic trends. Incorporating indicators like interest rates, inflation rates, or unemployment rates can give our model a more comprehensive view of the market landscape [90].

In conclusion, while our research has made significant strides in the realm of portfolio optimization using DRL, there's always room for improvement. We believe that by going into the above directions, future research can push the boundaries even further, paving the way for more effective and adaptive financial models.

8 Appendix

Algorithm	Description	Policy	Action Space	State Space	Operator
Monte Carlo	Every visit to Monte Carlo	Either	Discrete	Discrete	Sample-means
Q-learning	State–action–reward–state	Off-policy	Discrete	Discrete	Q-value
SARSA	State–action–reward–state–action	On-policy	Discrete	Discrete	Q-value
Q-learning - Lambda	State–action–reward–state with eligibility traces	Off-policy	Discrete	Discrete	Q-value
SARSA - Lambda	State–action–reward–state–action with eligibility traces	On-policy	Discrete	Discrete	Q-value
DQN	Deep Q Network	Off-policy	Discrete	Continuous	Q-value
DDPG	Deep Deterministic Policy Gradient	Off-policy	Continuous	Continuous	Q-value
A3C	Asynchronous Advantage Actor-Critic Algorithm	On-policy	Continuous	Continuous	Advantage
NAF	Q-Learning with Normalized Advantage Functions	Off-policy	Continuous	Continuous	Advantage
TRPO	Trust Region Policy Optimization	On-policy	Continuous or Discrete	Continuous	Advantage
PPO	Proximal Policy Optimization	On-policy	Continuous or Discrete	Continuous	Advantage
TD3	Twin Delayed Deep Deterministic Policy Gradient	Off-policy	Continuous	Continuous	Q-value
SAC	Soft Actor-Critic	Off-policy	Continuous	Continuous	Advantage

Table 19: Comparison Reinforcement Learning Algorithms. Source @wikipedia

Library	Version	Description
numpy	1.18.1	Numerical computing library for arrays and matrices
pandas	0.25.1	Data manipulation and analysis library
tflearn	0.3.2	Deep learning library built on top of TensorFlow
tensorflow	1.15.0	Deep learning framework developed by Google
keras	2.3.1	High-level neural networks API
yfinance	0.1.54	Yahoo Finance API to retrieve financial data
cvxopt	1.2.5	Library for convex optimization problems
gym	0.17.1	OpenAI Gym, a toolkit for developing and comparing RL algorithms
matplotlib	3.1.3	Data visualization library
seaborn	0.10.0	Data visualization library based on matplotlib
scikit-learn	0.23.2	Machine learning library
h5py	2.10.0	Interface for working with HDF5 files
theano	1.0.4	Numerical computation library for deep learning
pyqt	5.9.2	Python bindings for Qt, a GUI framework
bokeh	1.4.0	Interactive data visualization library
plotly	4.9.0	Interactive graphing library

Table 20: List of Relevant Libraries, Versions, and Descriptions

References

[1] Yunan Ye, Hengzhi Pei, Boxin Wang, Pin-Yu Chen, Yada Zhu, Jun Xiao, and Bo Li. Reinforcement-learning based portfolio management with augmented asset movement prediction states, 2020.

[2] Gang Huang, Xiaohua Zhou, and Qingyang Song. Deep reinforcement learning for portfolio management, 2022.

[3] @arshren. Step-by-step guide to implementing ddpg reinforcement learning in pytorch. *Medium*, 2020.

[4] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[5] Krzysztof Zarzycki and Maciej Ławryńczuk. Lstm and gru neural networks as models of dynamical processes used in predictive control: A comparison of models developed for two chemical reactors. *Sensors*, 21:5625, 08 2021.

[6] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[7] Kunpeng Liu, Lihua Gong, Nuo Tian, Feixiang Gong, and Qi Wang. Feature extraction method of power grid load data based on stft-crnn. In *Proceedings of the 6th International Conference on Big Data and Computing*, ICBDC '21, page 55–60, New York, NY, USA, 2021. Association for Computing Machinery.

[8] Shashank Hegde, Vishal Kumar, and Atul Singh. Risk aware portfolio construction using deep deterministic policy gradients. In *2018 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1861–1867, 2018.

[9] Yifan Zhang, Peilin Zhao, Qingyao Wu, Bin Li, Junzhou Huang, and Mingkui Tan. Cost-sensitive portfolio selection via deep reinforcement learning. *CoRR*, abs/2003.03051, 2020.

[10] Jinho Lee, Raehyun Kim, Seok-Won Yi, and Jaewoo Kang. MAPS: Multi-agent reinforcement learning-based portfolio management system. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*. International Joint Conferences on Artificial Intelligence Organization, jul 2020.

[11] Harry Markowitz. Portfolio selection. *The Journal of Finance*, 7(1):77–91, 1952.

[12] H. M. Markowitz. *Portfolio Selection: Efficient Diversification of Investments*. John Wiley & Sons, New York, 1959.

[13] Harry M. Markowitz. *Portfolio Selection: Efficient Diversification of Investments*. Yale University Press, 1959.

[14] E. J. Elton, M. J. Gruber, S. J. Brown, and W. N. Goetzmann. *Modern Portfolio Theory and Investment Analysis*. John Wiley & Sons, 2007.

[15] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[16] Gerard Cornuejols and Reha Tütüncü. *Optimization Methods in Finance*. Cambridge University Press, 2006.

[17] David G Luenberger. *Investment science*. Oxford University Press, 1997.

[18] Shihao Gu, Bryan T Kelly, and Dacheng Xiu. Empirical asset pricing via machine learning. *The Review of Financial Studies*, 33(5):2223–2273, 2020.

[19] Sewon Baek and Sanjeev K Mohanty. Deep learning in asset pricing. *Available at SSRN 3528774*, 2020.

[20] Alexandros Tsantekidis, Nikolaos Passalis, Anastasios Tefas, Juho Kanniainen, Moncef Gabbouj, and Alexandros Iosifidis. Forecasting stock prices from the limit order book using convolutional neural networks. In *2017 IEEE 19th Conference on Business Informatics (CBI)*, volume 1, pages 7–12. IEEE, 2017.

[21] Hyung-Jeong Jang and Kyong Joo Kim. Applying deep learning to enhance momentum trading strategies in stocks. *Journal of Big Data*, 7(1):1–15, 2020.

[22] Vladimir Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag New York, 1995.

[23] Wei Huang, Yoshiteru Nakamori, and Shouyang Wang. Forecasting stock market movement direction with support vector machine. *Computers & Operations Research*, 32(10):2513–2522, 2005.

[24] J.B. Heaton, N.G. Polson, and J.H. Witte. Deep learning for finance: Deep portfolios. *Applied Stochastic Models in Business and Industry*, 2017.

[25] Wei Bao, Jun Yue, and Yulei Rao. Deep learning framework for financial time series using stacked autoencoders and long-short term memory. In *Proceedings of the 2019 11th International Conference on Machine Learning and Computing*, pages 57–61. ACM, 2019.

[26] Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.

[27] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[28] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[29] Dipti Ravi, Carol Wong, Fani Deligianni, Mickael Berthelot, Javier Andreu-Perez, Benny Lo, and Guang-Zhong Yang. The 'black box' problem and lack of transparency with ai in medicine. *Healthcare Technology Letters*, 7(2):27–35, 2020.

[30] Robert Almgren and Neil Chriss. Optimal execution of portfolio transactions. *Journal of Risk*, 3:5–40, 2000.

[31] Tianyang Chang, Yunjie Yang, and Zhaojun Zhang. Portfolio optimization under constraints. *European Journal of Operational Research*, 258(2):664–672, 2017.

[32] Rama Cont. Empirical properties of asset returns: stylized facts and statistical issues. *Quantitative Finance*, 1(2):223–236, 2001.

[33] John Y Campbell, Andrew W Lo, and A Craig MacKinlay. *The econometrics of financial markets*. Princeton university press, 1997.

[34] Geert Bekaert, Campbell R Harvey, Christian T Lundblad, and Stephan Siegel. What segments equity markets? *The Review of Financial Studies*, 24(12):3841–3890, 2011.

[35] Financial assets: All you need to leverage effectively (+ best tips). https://businessyield.com/finance-accounting/financial-assets/.

[36] Trading indicators. https://capex.com/en/academy/trading-indicators.

[37] Unlockcing the secretof sortino ratio: The ultimate guide. https://wealthyeducation.com/sortino-ratio/.

[38] Martin L. Puterman. Markov decision processes: Discrete stochastic dynamic programming. In *Wiley Series in Probability and Statistics*, 1994.

[39] Richard S Sutton and Andrew G Barto. *Reinforcement Learning: An Introduction*. MIT press, 2018.

[40] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., 1994.

[41] Thomas T. Hills, Peter M. Todd, and Robert L. Goldstone. Search in external and internal spaces: Evidence for generalized cognitive search processes. *Psychological Science*, 26(7):802–808, 2015.

[42] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, USA, 2 edition, 2018.

[43] Yarin Gal. *Uncertainty in Deep Learning*. PhD thesis, University of Cambridge, 2016.

[44] Richard Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA, 1958.

[45] Ryszard S Michalski, Jaime G Carbonell, and Tom M Mitchell. *Machine learning: An artificial intelligence approach*. Springer Science & Business Media, 2013.

[46] Dimitri P Bertsekas. *Dynamic Programming and Optimal Control*, volume 1. Belmont, MA: Athena Scientific, 2005.

[47] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.

[48] Franco Bertoluzzo, Marco Corazza, and Walter Montagna. Forecasting industrial production and the early detection of turning points. *Economies*, 1(1):26–46, 2012.

[49] John Moody, Lizhen Wu, Yufeng Liao, and Matthew Saffell. Learning to trade via direct reinforcement. In *International Conference on Artificial Intelligence*, volume 1, pages 1626–1633. Citeseer, 2001.

[50] Beakcheol Jang, Myeonghwi Kim, Gaspard Harerimana, and Jong Wook Kim. Q-learning algorithms: A comprehensive classification and applications. *IEEE Access*, 7:133653–133667, 2019. Publisher Copyright: © 2013 IEEE.

[51] Mohit Sewak. *Policy-Based Reinforcement Learning Approaches: Stochastic Policy Gradient and the REINFORCE Algorithm*, pages 127–140. 06 2019.

[52] Kevin Dabeacute;rius, Elvin Granat, and Patrik Karlsson. Deep execution - value and policy based reinforcement learning for trading and beating market benchmarks. *SSRN Electronic Journal*, 01 2019.

[53] Marco Corazza and Andrea Sangalli. Q-Learning and SARSA: a comparison between two intelligent stochastic control approaches for financial trading. Technical report, 2015.

[54] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015.

[55] Longxin Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8:293–321, 2004.

[56] Boyi Liu, Qi Cai, Zhuoran Yang, and Zhaoran Wang. Neural proximal/trust region policy optimization attains globally optimal policy. *Advances in Neural Information Processing Systems*, 32, 2019. Publisher Copyright: © 2019 Neural information processing systems foundation. All rights reserved.; 33rd Annual Conference on Neural Information Processing Systems, NeurIPS 2019 ; Conference date: 08-12-2019 Through 14-12-2019.

[57] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning, 2015.

[58] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. *International Conference on Machine Learning*, 2018.

[59] R.S. Sutton and A.G. Barto. Reinforcement learning: An introduction. *IEEE Transactions on Neural Networks*, 9(5):1054–1054, 1998.

[60] Hyungjun Park, Min Kyu Sim, and Dong Gu Choi. An intelligent financial portfolio trading strategy using deep q-learning, 2019.

[61] Xiao-Yang Liu, Zhuoran Xiong, Shan Zhong, Hongyang Yang, and Anwar Walid. Practical deep reinforcement learning approach for stock trading, 2018.

[62] Zhengyao Jiang, Dixing Xu, and Jinjun Liang. A deep reinforcement learning framework for the financial portfolio management problem, 2017.

[63] Pengqian Yu, Joon Sern Lee, Ilya Kulyatin, Zekun Shi, and Sakyasingha Dasgupta. Model-based deep reinforcement learning for dynamic portfolio optimization, 2019.

[64] Ben Hambly, Renyuan Xu, and Huining Yang. Recent Advances in Reinforcement Learning in Finance. Papers 2112.04553, arXiv.org, December 2021.

[65] Paul Wilmott. *Paul Wilmott Introduces Quantitative Finance*. Wiley-Interscience, USA, 2 edition, 2007.

[66] Siyi Chen, Bin Li, and Bin Zhou. Portfolio selection with transaction costs and factor models. *Management Science*, 64(4):1806–1823, 2018.

[67] Bin Li and Steven C.H. Hoi. Online portfolio selection: Principles and algorithms. In *Advances in Financial Machine Learning*, pages 337–368. Wiley, 2014.

[68] Bin Li, Steven C.H. Hoi, and Vivek Gopalkrishnan. Online portfolio selection: A survey. *ACM Computing Surveys (CSUR)*, 46(3):1–36, 2011.

[69] Bin Li and Steven CH Hoi. Best constant rebalanced portfolio. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, volume 32, pages 1618–1626, 2014.

[70] Bin Li and Steven CH Hoi. On-line portfolio selection with moving average reversion. *Journal of Machine Learning Research*, 15(1):2059–2079, 2014.

[71] Thomas M Cover. Universal portfolios. *Mathematical Finance*, 1(1):1–29, 1996.

[72] J. Kirkpatrick, R. Pascanu, N. C. Rabinowitz, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114:3521–3526, 2017.

[73] V. Mnih, K. Kavukcuoglu, D. Silver, et al. Human-level control through deep reinforcement learning. *Nature*, 518:529–533, 2015.

[74] Zuyue Fu, Zhuoran Yang, and Zhaoran Wang. Single-timescale actor-critic provably finds globally optimal policy. *CoRR*, abs/2008.00483, 2020.

[75] Edi Muskardin, Martin Tappler, Bernhard K. Aichernig, and Ingo Pill. Reinforcement learning under partial observability guided by learned environment models, 2022.

[76] X. Kanwar. Title of kanwar's paper. *Journal Name*, 2019.

[77] T. P. Lillicrap et al. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2016.

[78] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller. Deterministic policy gradient algorithms. In *ICML*, 2014.

[79] J. Liang et al. Title of liang's paper. *Journal Name*, 2018.

[80] G. Trunk. A problem of dimensionality: A simple example. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-1:306–307, 1979.

[81] S. Thrun and A. Schwartz. Issues in using function approximation for reinforcement learning. pages 255–262, 1993.

[82] Christoph Graf, Viktor Zobernig, Johannes Schmidt, and Claude Klöckl. Computational performance of deep reinforcement learning to find nash equilibria. *CoRR*, abs/2104.12895, 2021.

[83] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[84] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[85] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[86] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with lstm. *Neural computation*, 12(10):2451–2471, 1999.

[87] Junyoung Chung, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[88] Anita S Harsoor and Anushree Patil. Forecast of sales of walmart store using big data applications. *International Journal of Research in Engineering and Technology*, 4(6):51–59, 2015.

[89] Frank Z. Xing, Zhaoxia Wang, and Yue Zhang. Sentiment-aware volatility forecasting. *Knowledge-Based Systems*, 2019.

[90] Tarika Singh, Seema Mehta, and MS Varsha. Macroeconomic factors and stock returns: Evidence from taiwan. *Journal of economics and international finance*, 3(4):217, 2011.

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

Authored by (in block letters):
For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**

With my signature I confirm that

- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.