Robust Deep Reinforcement Learning for Portfolio Management

Mohammad Amin Masoudi

A thesis submitted in partial fulfillment of the requirements for the Master's degree in

Master of Science in Management

Telfer School of Management

University of Ottawa

Contents

List of Tables

List of Figures

Abstract

In Finance, the use of Automated Trading Systems (ATS) on markets is growing every year and the trades generated by an algorithm now account for most of orders that arrive at stock exchanges (Kissell, 2020). Historically, these systems were based on advanced statistical methods and signal processing designed to extract trading signals from financial data. The recent success of Machine Learning has attracted the interest of the financial community. Reinforcement Learning is a subcategory of machine learning and has been broadly applied by investors and researchers in building trading systems (Kissell, 2020). In this thesis, we address the issue that deep reinforcement learning may be susceptible to sampling errors and over-fitting and propose a robust deep reinforcement learning method that integrates techniques from reinforcement learning and robust optimization. We back-test and compare the performance of the developed algorithm, Robust DDPG, with UBAH (Uniform Buy and Hold) benchmark and other RL algorithms and show that the robust algorithm of this research can reduce the downside risk of an investment strategy significantly and can ensure a safer path for the investor's portfolio value.

1. Introduction

1.1 Background

In Portfolio Management or Financial Investing, a general goal is to dynamically allocate a set of assets to maximize the return of the portfolio over time and minimize risk simultaneously. This research develops an automated trading system that ensures robustness against uncertainty in the parameters estimated from the financial market. The system is built upon combination of modern machine learning techniques and robust optimization methodology.

For investors, it is essential to be able to invest in a portfolio that can satisfy their preset goals by building an optimal portfolio initially and subsequently rebalancing it optimally (Almahdi & Yang, 2017). Recently, Machine Learning and Artificial Intelligence technology, in particular, Deep Reinforcement Learning (DRL) (a branch of machine Learning that seeks to find an optimal strategy for a sequential decision problem by directly interacting with the environment) approaches have shed new light on potential methods to optimize a portfolio. Using DRL we can fully automate investment strategies that require an intelligent machine to take into account the risk and return of a financial asset over time to make profitable investment decisions. Although there have been rapid improvements in the DRL algorithm, uncertainty in the parameters learned by machines from sample data (in a real-world problem) could cause major problems to decision-makers not only in finance but also in other practical areas where the cost of failure is significant including healthcare, robotics, self-driving cars, etc. Thus, the highly volatile nature of the financial market still requires robust approaches to mitigate the impact of estimation error on an investment strategy that may lead to a substantial loss of capital. This research is one of the firsts to develop a robust learning framework for DRL trading strategies using optimization algorithms. We deploy Robust Optimization and Actor-Critic algorithms, a popular algorithm for Policy Optimization of a Reinforcement Learning agent, to immunize the learning process of the artificial agent from the adversarial impact of sampling errors.

The main advantage of using DRL in portfolio management is that it bypasses the difficulty of pricing forecasts and focuses directly on optimizing the final portfolio performance, thereby closely aligning the machine learning problem with the objectives of the investor (Fischer, 2018).

In Robust Optimization, Markov decision processes (MDP) provides a mathematical framework used for modeling decision-making problems where the outcomes are partly random and partly controllable. They capture several attractive features that are important in decision making under uncertainty: they control risk in sequential decision making via a state transition probability matrix, while taking into account the possibility of information gathering and using this information to apply recourse during the multistage decision process (Puterman, 1994) (Bertsekas & Tsitsiklis, 1996), (Feinberg & Shwartz, 2002).

Moreover, by providing a robust mathematical model of portfolio management for the DRL algorithm we take into account the need of investors who constantly seek for more stable performances, making the software portfolio manager suitable for real-time investments strategies.

1.2 Literature Review

Mathematical models designed to solve the allocation problem of a financial portfolio gained a lot of attention after the introduction of the Modern Portfolio Theory (MPT) by (Markowitz, 1952). It considers rational investors and models the problem keeping the mean-variance analysis framework in perspective, where the variance of the portfolio is minimized with a fixed value for the expected return on the entire portfolio. The framework also assumes a market without any taxes or transaction costs, and where short selling is disallowed but assets are infinitely divisible and can be traded with any non-negative fractions. (Tobin & Hester, 1967) presents the inclusion of risk-free assets in the traditional Markowitz formulation by the development of the Separation theorem which states that in the presence of a risk-free asset, the optimal risky portfolio can be obtained without any knowledge of the investor's preferences, whereas, Sharpe's Capital Asset Pricing Model (CAPM) (Sharpe, 1964) takes into account the asset's sensitivity to non-diversifiable risk while it is being added to an already existing well-diversified portfolio. It considers the importance of the covariance structure of the returns, the variance of the portfolio and the market premium (i.e. the difference between the expected return on the asset from the market and the risk-free rate of return on the asset). The model assumes that the investors are rational and risk-averse, are broadly diversified across a range of investments, and that they cannot influence the prices

of the assets. Assumptions regarding trade or transaction costs, short-selling and trades with non-negative fractions do apply from the traditional Markowitz's framework.

More recently, Ross's Arbitrage Pricing Theory (APT) (Ross, 1976) models the expected return of a financial asset as a linear function of various macro-economic factors (where, sensitivity to changes in each factor is represented by the factor-specific beta coefficients of the regression model). Assumptions regarding trade or transaction costs, short-selling and trades with non-negative fractions do apply from the traditional Markowitz's framework. In (Uryasev, 2000), Conditional-Value-at-Risk (CVaR) is proposed as a risk measure of the portfolio, and therefore optimization would be on top of that metric. The CVaR is derived from the weighted average of "extreme" losses in the tail of the distribution of possible returns, in addition to the cutoff point of the Value at Risk (VaR). One of the advantages of CVaR is that it can be minimized efficiently using LP (Linear Programming) techniques. In studies of financial calculus, considering the equity markets in perspective, Fernholz's Stochastic Portfolio Theory (Fernholz, 2002) and (Karatzas & Fernholz, 2008) discusses a descriptive theory that provides a framework for analysing portfolio behaviour and equity market structure that has both theoretical and practical applications. It provides insights into questions of market behaviour and arbitrage principle. It is used to construct portfolios with controlled behaviour under certain conditions and provides methods for managing the performance of the portfolio. Stochastic optimization methods tend to appear as solution approaches to portfolio optimization problem modeled with reference to the Stochastic Portfolio Theory as mentioned above. Stochastic-search based optimization using local search procedures refers to introducing randomness in to the search process to accelerate progress towards leading to the optimal local solution. Challenges in such a methodology are with regard to handling constraints. (Crama & Schyns, 2003) provides a simulated annealing-based approach to complex portfolio selection problems by a systematic inclusion of constraints. (Ardia, Boudt, Carl, Mullen, Peterson, 2010) and (Krink & Paterlini, 2011) provide differential evolution based stochastic-search heuristic methods for multi-objective portfolio optimization problem with realistic constraints. (Hu, Liu, Zhang, Lijun Su, 2015) studied the application of evolutionary computation in the discovery of rules in algorithmic trading for shares; (Ertenlice, Kalayci, 2018) conducted a swarm intelligence research for portfolio optimization, discussing algorithms and applications. Numerous other portfolio optimization models from conventional numerical methods of analytic and non-analytical approximations to

stochastic programming, evolutionary computations and swarm intelligence using population based algorithmic developments, and, machine learning based computationally guided mechanisms have been proposed to address practical issues such as transaction costs, risk measurement, among others, to enhance the performance of portfolio management in a real-life setting. Further literature presented in this section, intends to broadly highlight machine learning approaches to the portfolio optimization problem.

In the past decade, machine learning algorithms have arisen as important tools to help find the patterns and predict future movements of assets using supervised learning techniques. (Khaidem, Saha, Roy Dey, 2016) predicts the direction of stock market prices using random forests (RF) and shows that the model is robust in predicting the future direction of stock movement. (Akhavan Niaki & Hoseinzade, 2013) forecast the daily direction of Standard & Poor's 500 (S&P 500) index using an artificial neural network (ANN) and shows ANN could significantly improve the trading profit as compared with the buy-and-hold strategy. (Tsantekidis, Passalis, Tefas, Kanniainen, Gabbouj, Iosifidis, 2017) show that the convolutional neural network (CNN) is better suited to predict the price movements of stocks than multilayer neural networks and support vector machines. The main idea behind this (supervised) approach is to train a predictive model, a neural network, or a random forest or support vector machines, on historical data to forecast the asset's price change. These forecasts are fed into a trading module to derive the actual trading action, e.g., to buy the financial asset in case the forecast surpasses a certain threshold. A major disadvantage of the price-prediction-based algorithms is that they highly depend on the degree of prediction accuracy. Unfortunately, future market prices are difficult to predict. Furthermore, converting price predictions into actions requires additional layers of logic. If this layer is hand-coded, then the whole approach is not fully automated, and thus is not very extensible or adaptable. For example, it is difficult for a prediction-based network to consider transaction costs as a risk factor (Jiang, Xu, Liang, 2017).

Previous successful attempts to provide fully automated schemes for the algorithmic trading problem, without predicting future prices, use a reinforcement learning (RL) approach. These include (Moody & Saffell, 2001) who proposed the idea of recurrent reinforcement Learning (RRL) ("Recurrent" means that previous output is fed into the model as a part of input for direct reinforcement) and demonstrates how direct reinforcement (which finds the optimal policy by directly maximizing the state value function) can be used to optimize risk-adjusted investment returns while accounting for the effects of transaction costs, (Dempster & Leemans, 2006) demonstrate that RRL trading systems can be embedded in a multi-layered risk management system and improve agent performance,

(Cumming, 2006) apply RL to build trading systems on intraday FX data and (Deng, Bao, Kong, Ren, Dai, 2017) show how deep learning techniques can be levered to extract higher-level features from the state and proposed an extension to backpropagation through time to efficiently train the agent. These RL algorithms output discrete trading signals on an asset. Being limited to single-asset trading, they do not apply to general portfolio management problems, where trading agents manage multiple assets.

Recently, Google DeepMind has devised a solid algorithm to tackle the continuous action space problem. Building off the prior work of (Silver, Lever, Heess, Degris , Wierstra, Riedmiller, 2014) on Deterministic Policy Gradients method that uses off policy data and target at modeling and optimizing the policy directly, they have produced a policy-gradient actor-critic algorithm called Deep Deterministic Policy Gradients (DDPG) (Lillicrap, Hunt, Pritzel, Heess, Erez, Tassa, Silver, Wierstra, 2015) also use an Actor-Critic method (read more in section 4.2). The continuous output in these actor-critic algorithms is achieved by a neural network approximating action policy function, and a second network is trained as the reward (value) function estimator.

Using the idea behind DDPG, (Jiang, Xu, Liang, 2017) proposes an actor-only RL framework designed for portfolio management to simultaneously manage a multi-asset portfolio by taking into consideration market risks and the transaction costs associated with buying and selling assets in a stock exchange.

Despite these achievements, they do not study the issue of sampling errors and over-fitting and how to mitigate the adverse impact of it. While the literature mainly focuses on actor-only and citric-only approaches we exploit the structure of actor-critic algorithm is more suited to for incorporating robustness to the agent's policy while maximizing the final return of a portfolio (we discuss more about RL methods in Section 3).

In robust control of a sequential decision-making problem, to address the issue of uncertainty in the transition matrix of MDPs, (Shapiro & Kleywegt, 2002) proposed a Bayesian approach that requires a perfect knowledge of the whole prior distribution on the transition matrix, making it difficult to apply in practice. (Nilim & El Ghaoui, 2005) considers a robust control problem for Markov decision processes, where uncertainty in the transition matrices is described in terms of possibly nonconvex sets. They show that where perfect duality holds for this problem it can be solved using the "robust dynamic programming" algorithm. They prove that the complexity of the robust recursion is almost the same as that of the classical recursion.

1.3 Research Question

Recognizing the lack of risk management practices in RL algorithms (where learning is just based on a sample training set) and recognizing their importance in the portfolio management problem where sampling error and uncertainty in asset prices may lead to wrong decisions by the RL trading agent and causing a significant capital loss, we propose this research question:

- "How can a RL trading agent maximize portfolio return while immunizing the learning process of portfolio management from the adversarial impact due to sampling errors?"

1.4 Contribution and Outline of the thesis

In this thesis, we propose a robust extension to the state-of-the-art DDPG (Deep Deterministic Policy Gradient) algorithm with direct application to portfolio management. Then, we build a robust automated trading agent in Python programming language for deploying the developed algorithm, Robust DDPG, to trade with real-time price data. After evaluating the agent on several back-test experiments, we show that the new software agent not only offers a safer path for the investor's portfolio value during the trading period and reduce the riskiness of the investment strategy, It also can provide a higher rate of return comparing to DDPG algorithm (Table 3). The robust software portfolio manager of this research could be utilized by big investment companies such as hedge funds and investment banks as a more secure and discreet framework - that is inherently more robust to unpredictable and new events in the market - to utilize Artificial Intelligence (AI) technology for the investment strategies. It can also be used by anyone interested in investing and gaining profits by using an algorithmic trading system.

The rest of this thesis is organized as follows. In section 2, we provide the mathematical formulation of portfolio management. In section 3, the general Markov Decision Processes (MDP) formulation and a description of RL methods is provided. After that, a state-of-the-art reinforcement learning algorithm is utilized to provide the optimal solution and a robust optimal solution for the problem in section 4. At the end of section 4, the pseudocode for the new enhanced algorithm, Robust DDPG (Robust Deep Deterministic Policy Gradient) is explained. In section 5, we demonstrate results that the models have achieved so far in back-test experiments and compare their investment performance with each other and the benchmark using statistical tests. Finally, in section 6, the final conclusion of this thesis and potential future studies on this topic are discussed.

2. Portfolio Management Problem

In this section, a mathematical formulation of portfolio management is provided. We provide the MDP model of the portfolio management problem in section 4.1. After that, the proposed robust RL method is demonstrated in detail.

2.1 Problem formulation

We would like to manage a portfolio by distributing our investment into a number of stocks based on the market situation. We define *N* to be the number of stocks we would like to invest in. At the initial time of trading ($t = 0$), the total investment volume is assumed to be 1 dollar. The close price is the last price anyone paid for a security during the business hours. We define the close/open relative price vector as

$$y_t = [1, \frac{v_{1,t,close}}{v_{1,t-1,close}}, \frac{v_{2,t,close}}{v_{2,t-1,close}}, \cdots, \frac{v_{N,t,close}}{v_{N,t-1,close}}] \quad (1)$$

, where $\frac{v_{i,t,close}}{v_{i,t-1,close}}$ is the relative price ratio of stock *i* at time *t*. Note that, the first element of y_t represents the relative price of cash, which is always 1. We define the portfolio weight vector as

$$w_t = [w_{0,t}, w_{1,t}, \cdots, w_{N,t}] \ , \quad (2)$$

where $w_{i,t}$ represents the fraction of investment in stock *i* at time *t* and so

$$\sum_{i=0}^{N} w_{i,t} = 1 \quad (3)$$

Note that $w_{0,t}$ represents the dollar amount at hand. Then the profit after time *T* is

$$p_T = \prod_{t=1}^{T} y_t \cdot w_{t-1} \ , \quad (4)$$

where $\cdot$ is the inner product of the vectors. In a typical portfolio management problem, the initial portfolio weight vector w_0 is chosen to be the first basis vector in the Euclidean space, so $w_0 = [1,0,\cdots,0]$. Note that the portfolio vector at the beginning of period t is w_{t-1}.

In a real-world scenario, buying or selling assets in a market is not free. The cost is normally in the form of a commission fee. In order to take into account the transaction cost, a trading cost factor η is considered and the trading cost of each trade η_t is calculated in equation (6). Due to price movement in the market, at the end of the same period, the weights evolve into

$$w'_t = \frac{y_t \odot w_{t-1}}{y_t \cdot w_{t-1}} \tag{5}$$

$$\eta_t = \eta \sum_{i=0}^{N} \left| w'_{t,i} - w_{t,i} \right|, \tag{6}$$

where $\odot$ is the element-wise product of the vectors. Denoting p_{t-1} as the portfolio value at the beginning of period t and p'_t at the end, now the equation 4 becomes:

$$p_T = \prod_{t=1}^{T} \frac{p_t}{p_{t-1}} = \prod_{t=1}^{T} \frac{(1-\eta_t)p'_t}{p_{t-1}} = \prod_{t=1}^{T} (1-\eta_t) y_t \cdot w_{t-1} \tag{7}$$

The job of a portfolio manager is to maximize p_T by choosing portfolio weight vector *w* at each time *t* based on historical stock information. In section 4.1, the logarithm of the final wealth (equation 7) is used for the reward function in the MDP setting, instead of p_T itself. Note that we are utilizing the log function for the reward so because of the Jensen's inequality by using a concave utility function the trading agent would be risk-averse (Johnson, 2007). Figure 1 is an illustration of the effect of the trading cost.

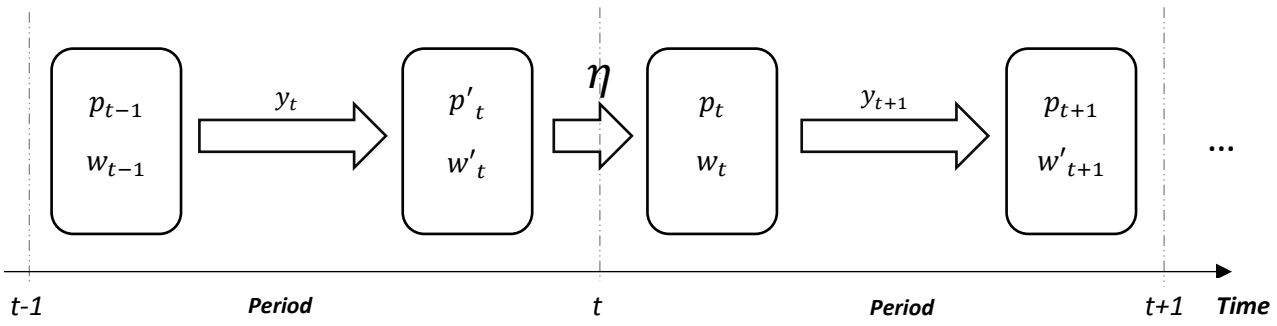

Figure 1. Transaction cost effect on portfolio value during time

2.2 The challenge of solving portfolio management problem

The Portfolio management problem gets particularly difficult to solve when there is a need to consider multiple assets or use a longer price history of assets. In such cases, techniques such as dynamic programming (DP) will suffer from curse of dimensionality and are computationally intractable. Thus, in this thesis, we used DRL to overcome these challenges and make a robust DRL framework for portfolio management.

3. Reinforcement Learning

Reinforcement Learning (RL) is a general class of algorithms in the field of machine learning that allows an agent to learn how to behave in a stochastic and possibly unknown environment, where the only feedback consists of a scalar reward signal. In order to maximize the long-run reward, the agent must learn which actions are the most profitable by trial-and-error. Therefore, RL algorithms can be seen as computational methods to solve Markov decision processes by directly interacting with the environment (financial market in this case), for which a model may or may not be available (not available in this case).

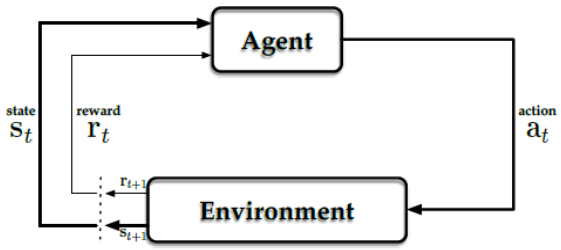

Figure 2. Reinforcement Learning

3.1 General MDP formulation

- **Markov Property:** for finite-state MDPs, the Markov property requires that "the future is independent of the past given the present". Therefore, the state s_t is Markov if and only if:

$$\Pr\{s_{t+1} \mid s_t\} = \Pr\{s_{t+1} \mid s_1, \dots s_t\} \quad (8)$$

An RL problem that inherits the Markov property is called a Markov Decision Process. MDP is a stochastic dynamical system that consists of the following elements:

- Agent: An RL agent is the entity which we are training to make correct decisions.
- Environment: The environment is the surrounding with which the agent interacts. The agent cannot manipulate the environment; it can only control its own actions
- State: $s_t \in S$, the state defines the current situation (observation) of the agent.
- Action: $a_t \in A$ The choice that the agent makes at the current time step. We know the set of actions (decisions) that the agent can perform in advance.
- Policy: π A policy is the thought process behind picking an action. It is a function that maps the states to the actions. If the policy is deterministic, then we have:

$$\pi : S \rightarrow A \tag{9}$$

$$a_t = \pi(s_t) \tag{10}$$

- State transition probabilities: the probability of moving from one state of a system into another state.

$$P_{ss'}^{a} = \Pr\{s_{t+1} = s' \mid s_t = s,\ a_t = a\} \tag{11}$$

In MDPs, the agent and the environment interact continually. At each step the agent selects an action based on its policy to maximize the expected sum of all future rewards and the environment responds to the action and presents the new state to the agent. In an RL problem, we seek to maximize the expected return where the return, R_t, is a summation of all the discounted rewards received by the agent from time step t onwards

$$R_t = r_{t+1} + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+1+k}\,, \tag{12}$$

Where the discount factor γ is a value (that can be chosen) between 0 and 1. For MDPs, we can define state-value function $V_\pi(s)$ that estimates the expected return starting from a specific state.

$$V_\pi(s) = E_\pi[R_t|S_t = s] = E_\pi[\sum_{k=0}^{\infty} \gamma^k r_{t+1+k}|S_t = s] \quad (13)$$

similarly, we can define action-value function $Q_\pi(\mathrm{s},\mathrm{a})$ that estimates the expected return starting from a specific state and taking a specific action in that state.

$$Q_\pi(\mathrm{s},\mathrm{a}) = E_\pi[R_t|S_t = s\,, A_t = a] = E_\pi[\sum_{k=0}^{\infty} \gamma^k r_{t+1+k}|S_t = s, A_t = a] \quad (14)$$

The Bellman equation decomposes the value function into two parts, the immediate reward plus the discounted future values and it gives a standard representation for value functions.

$$\begin{aligned} V_\pi(s) = E_\pi[R_t|S_t = s] &= E_\pi[\textstyle\sum_{k=0}^{\infty} \gamma^k r_{t+1+k}|S_t = s] = E_\pi[\, r_{t+1} + \gamma \textstyle\sum_{k=\mathbf{1}}^{\infty} \gamma^k r_{t+1+k}|S_t = s] \\ &= r_{t+1} + \gamma E_\pi[\textstyle\sum_{k=\mathbf{1}}^{\infty} \gamma^k r_{t+1+k}|S_t = s] = r_{t+1} + \gamma\, V_{\boldsymbol{\pi}}(s') \end{aligned} \quad (15)$$

- Bellman Equation for action-value function can similarly be written as:

$$Q_\pi(s,a) = r_{t+1} + \gamma\, Q_{\boldsymbol{\pi}}(s',a')\,, \quad (16)$$

Where s' denotes the next state. In an MDP environment, there are many different value functions according to different policies. The optimal Value function is one which yields maximum value compared to all other value function. The Bellman Optimality Equation expresses that the value of a state under an optimal policy must be equal to the expected return from the best action in that state.

$$\begin{aligned} V^*(s) &= max_a\, Q^{\pi^*}(s,a) = max_a E[r_{t+1} + \gamma\, V^*(s') \mid S_t = s, A_t = a] \\ Q^*(s,a) &= E[r_{t+1} + \gamma\, max_{a'}\, Q^*\,(s',a') \mid S_t = s, A_t = a] \end{aligned} \quad (17)$$

3.2 Methods

RL approaches branch into these major categories:

- **Critic-only approach:** The idea of this approach is to learn the Q value function. During the decision-making process, the agent observes the current state of the environment and selects the action with the best outcome according to the estimated value function.

$$Q^*(s_t, a_t) = \mathrm{E}[r_{t+1} + \gamma \, max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1}) \,|s_t = s_t, a_t = a_t] \tag{18}$$

so, the optimal policy π^* then simply becomes

$$\pi^*(s_t) = argmax \; Q^*(s_t, a_t) \tag{19}$$

- **Actor-only approach:** In this approach, the agent observes the state of the environment and acts directly, i.e., without computing and comparing the expected outcomes of different actions. The agent hence learns a direct mapping (a policy) from states to actions. In essence, actor-only method updates the probability distribution of actions so that actions with higher expected reward have a higher probability value of being chosen for an observed state. The key advantage of this approach is the continuous action space of the agent (e.g., to obtain fine-grained portfolio weights) and the typically faster convergence of the learning process. To achieve this, a policy is specified by a set of parameters θ, so

$$a_t = \pi_\theta(s_t) \tag{20}$$

Because, by definition, the reward function r evaluates the performance of π_θ in the training periods, policy parameters are continuously updated along the gradient direction of the reward function with a learning rate α. Here the objective function parameterized by θ is defined,

$$Z(\theta) = \max \; E[\sum_{t=0}^{T-1} r_{t+1} | \pi_\theta] = \sum_{t=0}^{T-1} P(s_t, a_t | \tau) r_{t+1} \tag{21}$$

$$\frac{\partial Z(\theta)}{\partial \theta} + \theta = \theta \;, \tag{22}$$

Where $P(s_t, a_t|\tau)$ is the probability of occurrence of s_t, a_t in the trajectory τ. A trajectory τ is a sequence of states and actions in the agent's environment: $\tau = (s_0, a_0, s_1, a_1, s_2, a_2, \dots)$.

- **Actor-Critic approach:** The actor-critic approach forms the third category and aims at combining the advantages of the actor-only and the critic-only approach. The key idea is to simultaneously use an actor,

who determines the agent's action given the current state of the environment, and a critic, who judges the selected action. Simply speaking, the actor learns to choose the action which is considered best by the critic and the critic learns to improve its judgment.

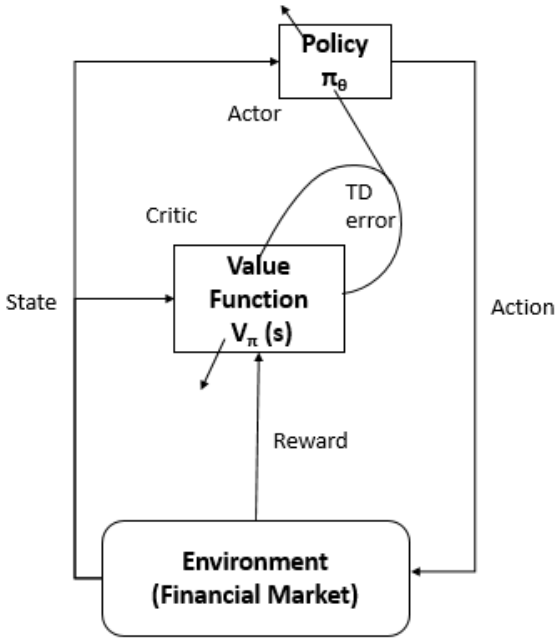

Figure 3. Actor-Critic Algorithm

- **Q-Learning (TD control):** is one of the classical solutions to the MDPs that use a tabular method to characterize the value functions and finds the optimal policy by iteratively updating Q-table. This method is an off-policy reinforcement learning algorithm (it figures out the optimal policy regardless of the agent's motivation) that seeks to find the best action to take given the current state. In this method, the states were discretized into a fixed number of bins with equal length to overcome the curse of dimensionality in dynamic programming methods. After initializing Q-table with arbitrary values, at each training step, the Q-table will be updated by the following rule and actions will be taken by an ϵ-greedy policy (where epsilon refers to the probability of choosing to explore, exploits most of the time with a small chance of exploring).

$$Q_t(s,a) = Q_{t-1}(s,a) + \alpha[r(s,a) + \gamma \max_{a'} Q(s',a') - Q_{t-1}(s,a)] \quad (23)$$

$$\pi^*(s) = arg\,max_a\, Q^\pi(s,a) = arg\,max_a\, E[r_{t+1} + \gamma\, V^\pi(s_{t+1}) \mid S_t = s, A_t = a], \quad (24)$$

where α is the learning rate and the term $[r(s,a) + \gamma \underset{a'}{max} Q(s',a') - Q_{t-1}(s,a)]$ is known as TD error.

This research explores the potential advantages of an actor-critic method to learn financial markets. Building off a novel actor-critic method called Deep Deterministic Policy Gradient (DDPG) robust setting for the algorithm was devised which is discussed in section 4 of this thesis.

3.3 Deep Reinforcement Learning

Deep reinforcement learning combines neural networks with a reinforcement learning architecture that enables software-defined agents to learn the best actions possible in virtual environment in order to attain their goals. That is, it unites function approximation and target optimization, mapping state-action pairs to expected rewards.

Neural networks are a collection of software 'neurons' arranged in layers, connected together in a way that allows communication between neurons. Each neuron has its own set of parameters ϑ. At each iteration, the neuron calculates a weighted average of the values of the input vector x and adds bias b,

$$z = \vartheta^T \cdot x + b \tag{25}$$

$$\hat{O} = f(z) \tag{26}$$

Then the result of this calculation is passed through a non-linear activation function f (e.g. RELU) to calculate $\hat{O}$ which will be the input to the successor neurons. Finally, at each iteration, after passing the input data through all of the units of the neural network, network parameters will be adjusted by the gradient descent algorithm along the gradient direction of the loss function.

In this thesis, a neural network was deployed to build a deep actor-critic structure for value function (Q (s, a)) approximation and policy parameterization (μ_θ) specifically, for the portfolio management task. The base structure for actor and critic networks is inspired by actor-critic agents of the original paper that introduced DDPG (Lillicrap, Hunt, Pritzel, Heess, Erez, Tassa, Silver, Wierstra, 2015). The actor and critic each have a reference neural network with similar structure to regularize the agent's learning process and increase its stability. Adam (Kingma & Ba, 2014) which is a gradient descent optimization algorithm is used to learn and update the neural network parameters (weights). The neural networks use the rectified non-linear functions (RELU) (Glorot &

Bordes & Bengio, 2011) for all hidden layers. Two different species of the devised actor-critic framework are used to investigate the performance of two different types of advanced deep networks to solve the problem. One uses Convolutional layers (CNN) (see Figure 4 for more details) and the other uses a Long Short-Term Memory (LSTM) layer with 32 hidden units at their core. These are followed by two Fully Connected layers with 64 units of neurons for each. In the actor network, a Softmax layer outputs the next period's action (assets weights) which will be fed to the critic network in the last fully-connected layer and the output of the critic' network will be a scalar value representing $Q_\pi(\mathrm{s}_t, \mathrm{a}_t)$.

This research deploys the following deep neural networks in a reinforcement learning setting and investigates their performance in learning the portfolio management task.

- **Fully Connected Neural Networks:** consists of a series of fully connected layers that connect every neuron in one layer to every neuron in the next.
- **Convolutional Neural Networks (CNN or ConvNets):** Basically, they are regularized (less prone to overfitting) versions of fully connected networks and take advantage of the hierarchical pattern in data and assemble more complex patterns using smaller and simpler patterns.
- **Long Short-Term Memory (LSTM) Networks:** Unlike standard feedforward neural networks, LSTM has feedback connections. It can not only process single data points (such as images), but also entire sequences of data (such as speech or video). A LSTM unit is composed of a cell, an input gate, an output gate and a forget gate. The cell remembers values over arbitrary time intervals and the three gates regulate the flow of information into and out of the cell.

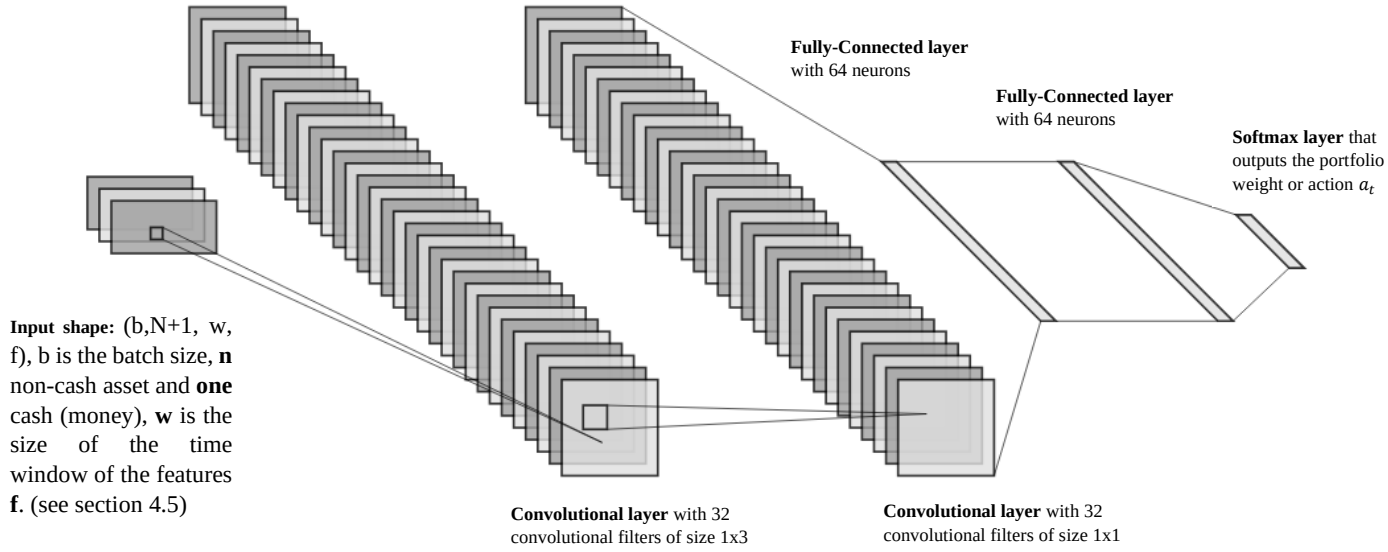

Figure 4. Abstract view of the policy (actor) network in the CNN-based actor-critic agent

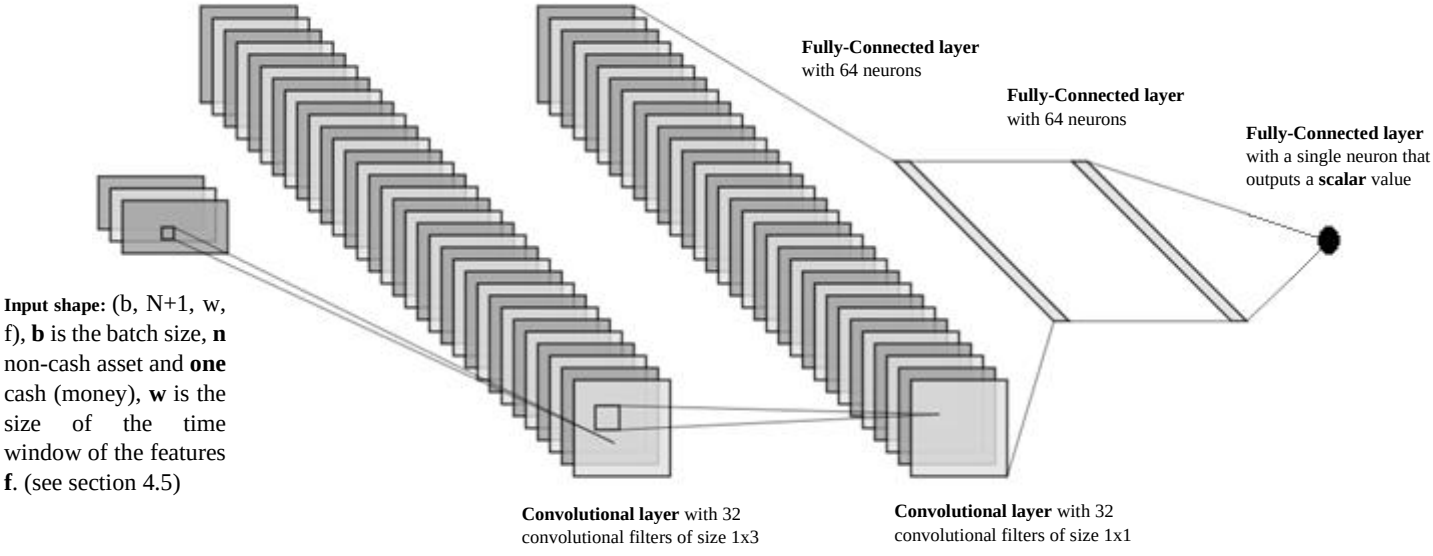

Figure 5. Abstract view of the critic network in the CNN-based actor-critic agent

LSTM networks are well-suited for time-series data because of their robustness to lags of unknown duration between important events in the input data, but convolutional layers although mainly used for high-dimensional image type data, have also shown good performance in time-series problems and even outperformed LSTM based

networks (Jiang, Xu, Liang, 2017). We believe they can find suitable deep features of the n-dimensional input time-series prices using convolution and pooling operations hence, enhancing the trading strategy.

Using deep learning as a function approximator for the agent's policy function enables the agent to use not only open, high, low, close prices of the stocks as input features to the deep networks but also other features such as trading volumes of each asset, textual news about the stocks and other alternative data. But for experimental purposes, just one feature is considered as the input to the policy network which is explained in next section of this thesis. Some advantages of using the proposed actor-critic network structure are its scalability with portfolio size and various inputs such as numerical (e.g., stock price, technical indicators, …) or categorical (e.g., tweeter, news, …) data because of using deep learning structure as well as flexibility to asset collection desirable in any portfolio management problem. Note that, text data must first be encoded as numbers using text encoding techniques (e.g., one-hot encoding, word embedding, …) to be used as input.

4. Methodology

4.1 Formulation of Portfolio Management based on Bellman equation (Q-value)

- States and Action

State s_t is the observation up to time *t*. As time goes by, the impact of historical data decreases. Thus, only the historical close prices in a within fixed window length W is considered. We define

$$s_t = \left[\vec{v_{1,t}}, \vec{v_{2,t}}, \cdots, \vec{v_{N,t}}\right], \tag{27}$$

where

$$\vec{v_{i,t}} = \begin{bmatrix} \frac{v_{i,t-W+1}}{v_{i,t-W}} \\ \frac{v_{i,t-W}}{v_{i,t-W+1}} \\ \vdots \\ \frac{v_{i,t}}{v_{i,t-1}} \end{bmatrix}, \tag{28}$$

where W is historical fixed window length And N is the number of stocks. Simply this means that the state s_t captures all the relevant information from the history.

The **action** a_t is just a portfolio weight vector w_t. Note that this action, unlike typical actions considered in MDP which take only finite values, can take any value within an interval, making this problem closer to an actual scenario.

- State Tensor

The state tensor (a mathematical object analogous to but more general than a vector, represented by an array of components that are functions of the coordinates of a space) of historic price data is fed into the neural network to generate the output of a portfolio weight vector. This section describes the structure of the input tensor and its normalization scheme. All missing data are filled with the previously available close price.

The input to the neural networks at the end of period t is a tensor, Ψ_t, of rank 3 with shape (N+1, w, f) where N is the number of preselected non-cash assets, w is the size of the input periods before t, and f is the feature number. Since the final portfolio value is determined by the relative price ratio of each period t (equation 7), instead of using the absolute open, high, low, and close price values of each period as input features to the neural networks, the following normalization technique is utilized:

$$\Psi_t = s_t = \mathbb{C} \times [(\vec{v}_{0,t} - 1), (\vec{v}_{1,t} - 1), \dots, (\vec{v}_{N,t} - 1)] \quad (29)$$

, where $\mathbb{C}$ is a scale factor (in the experiments, $\mathbb{C}$ is set to 100) and $\vec{v}_{i,t}$ is the historical fixed window length of relative (close) price ratio (equation 28).

- State Transitions

The underlying state evolution is determined by the market as the result of the stochastic nature of the financial asset's price process. The agent sees the observation state, which is the price of the stocks. Note, state transition probabilities themselves are uncertain, and in Section 4.2 we seek a robust decision (with respect to the estimated value function) for it to enable the agent to learn to eliminate the adverse risk of uncertainty accordingly.

- Reward Function

The logarithm of equation 7 is chosen as the reward function and is denoted by r. Due to the properties of logarithms, not only do we use a risk-averse objective for the agent (as discussed in section 2.1), but also have a specific reward at each step for the agent's actions during the training phase instead of having reward 0 at each timestamp and p_T at the end, thus enabling the policy to be trained more efficiently. As the agent does not have control over the choices of the initial investment, p_0, and the length of the whole portfolio management process, T, this job is equivalent to maximizing the average logarithmic cumulated return r:

$$r_{t+1}{}_{s}^{a} = [r_{t+1}|s_t = s, a_t = a] \tag{30}$$

$$r = \frac{1}{T} log \frac{p_T}{p_0} = \frac{1}{T} log \prod_{t=1}^{T+1} (1 - \eta_t)\, y_t \cdot w_{t-1} = \frac{1}{T} \sum_{t=1}^{T} log(\eta_t y_t \cdot w_{t-1}) \tag{31}$$

Where η_t is calculated using equation 6. In the experiments, η_t is set to 0.0025.

- Other Components of MDP

Before delving into the RL solution, other important components of the MDP model of portfolio management are explained below.

- **Policy** μ: maps states to actions taken by the agent. The policy can hence be understood as the agent's rules for how to choose actions. So, for a deterministic policy we have

$$a_t = \mu(s_t) \tag{32}$$

- **Value Function** $Q_\mu(\mathrm{s}_t, \mathrm{a}_t)$: maps states and actions to the total (discounted) reward the agent can expect from a given state until the end of the episode in the training phase under policy μ. So,

$$Q_\mu(\mathrm{s}, a) = E_\mu[r_t|s_t = s\,, a_t = a] = E_\mu[\sum_{k=0}^{T-t-1} \gamma^k\, r_{t+1+k}|s_t = s, a_t = a] \tag{33}$$

- **The Bellman Equation:**

$$Q(s_t, a_t) = r_i + \gamma E[Q(s_{i+1}, \mu(s_{i+1}|\theta^{\mu})|\theta^{Q})], \tag{34}$$

Where the *i* index refers to the *i*'th sample in the batch training data.

- **The State Transition Probability $P_{ss'}$** :

$$P_{ss'} = \Pr\{s_{t+1} = s' \mid s_t = s\} \tag{35}$$

The artificial portfolio manager interacts with the simulated market environment for a fixed number of steps. At each step of the episode the agent picks an action a_t based on policy $\mu(s_t)$ to maximize the value function $Q_{\mu}(s_t, a_t)$ and make a transition to the next state. Note that in this implementation we assume the actions taken by the agent does not affect the state transition. Then the agent's policy will be updated based on the observed reward in each step. We can also define all state transitions in terms of a State Transition Matrix P, where each row tells us the transition probabilities from one state to all possible successor states. Sum of each row is equal to 1.

$$P = \begin{bmatrix} P_{11} & \cdots & P_{1d} \\ \vdots & \ddots & \vdots \\ P_{d1} & \cdots & P_{dd} \end{bmatrix} \tag{36}$$

, where d is the number of different states. Note, since we do not know the state transition probabilities, we use the Empirical Transition Matrix $\hat{P}$ that is estimated just from the training data by using an unsupervised binning method. We will discuss how we obtain the empirical transition matrix in detail in section 4.2.1.

4.2 Robust DDPG

In this section, first we explain the Deep Deterministic Policy Gradient (DDPG) algorithm in depth. Then, the robust MDP model of portfolio management and Robust Deep Deterministic Policy Gradient (Robust DDPG) are demonstrated in detail.

Deep Deterministic Policy Gradient (DDPG) is a policy gradient algorithm that uses a stochastic behavior policy ($\mu_{\theta^Q}(s_t, a_t) = P(a_t = a| s_t = s)$) for good exploration but learns a deterministic policy ($a_{t+1} = \mu_{\theta^{\mu}}(s_t)$).

Policy gradient algorithms utilize a form of policy iteration: they evaluate the policy and then follow the policy gradient to maximize performance. Since DDPG is off-policy (concurrently learns θ^Q and θ^μ) and uses a deterministic policy, this allows for the use of the Deterministic Policy Gradient theorem which will be derived shortly.

Before delving into the DDPG algorithm learning process, it is worthwhile to explain a novel technique used by the DDPG algorithm in the training phase so called Experience Replay. If the network learned only from consecutive samples of experience as they occurred sequentially in the environment, the samples would be highly correlated and would therefore lead to inefficient learning. In this technique, we store the experiences of the agent during training, and then randomly sample experiences to use for learning in order to break up the temporal correlations within different training episodes. Exploration is done via adding noise to the action itself, in the DDPG paper (Lillicrap, Hunt, Pritzel, Heess, Erez, Tassa, Silver, Wierstra, 2015), the authors use Ornstein-Uhlenbeck Process to add noise to the action output (Uhlenbeck & Ornstein, 1930). The Ornstein-Uhlenbeck Process generates noise that is correlated with the previous noise, as to prevent the noise from canceling out the overall dynamics.

DDPG's structure primarily uses two neural networks, one for the actor and one for the critic. These networks compute action predictions for the current state and generate a temporal-difference (TD) error signal at each time step (Temporal difference (TD) learning refers to a class of model-free reinforcement learning methods which learn by bootstrapping[1] from the current estimate of the value function. These methods sample from the environment, like Monte Carlo methods, and perform updates based on current estimates, like dynamic programming methods (Bellman recursion). The input of the actor network is the current state, and the output is a single real value representing an action chosen from a continuous action space. The critic's output is simply the estimated Q-value of the current state and of the action given by the actor $Q(\mathrm{s}_t, \mathrm{a}_t)$. The deterministic policy gradient theorem provides the update rule for the weights of the actor network (see Theorem 1). The Critic loss is a simple TD-error where we use target networks Q' to compute Q-value for the next state. We need to minimize this loss. It was recently discovered that using a set of target networks (The target networks are delayed networks compared to main networks. The weights of targets are updated periodically based on the main

[1] In general, bootstrapping in RL means estimating something based on another estimation.

networks) to generate the targets for TD error computation regularizes your learning algorithm and increases stability. Accordingly, here are the equations for the TD target y_i and the loss function for the critic network

$$y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'}) \tag{37}$$

$$L = \frac{1}{b}\sum_{i}^{b}(y_i - Q(s_i, a_i|\theta^Q)^2) \tag{38}$$

Here, a batch of size *b* has been sampled from the experience replay buffer, with the *i* index referring to the *i*'th sample. The target for the TD error computation, y_i , is computed from the sum of the immediate reward and the outputs of the target actor and critic networks, having weights $\theta^{\mu'}$ and $\theta^{Q'}$ respectively. Then, the critic loss can be computed w.r.t. the output $Q(s_i, a_i|\theta^Q)$ of the critic network for the i'th sample.

(Silver, Lever, Heess, Degris, Wierstra, Riedmiller, 2014) proved that the stochastic policy gradient $\nabla_{\theta^\mu}\mu$, which is the gradient of the policy's performance, is equivalent to the deterministic policy gradient, which is given by:

$$\nabla_{\theta^\mu}\mu \approx \mathbb{E}_{\mu'}[\nabla_a Q(s, a|\theta^Q)|_{s=s_t, a=\mu(s_t)}\nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s=s_t}] \tag{39}$$

Note that policy term in the expectation is not a distribution over actions.

Theorem 1. *(Deterministic Policy Gradient Theorem). Suppose the Markov Decision Process satisfies the appropriate conditions (see more* (Silver, Lever, Heess, Degris, Wierstra, Riedmiller, 2014)). *These imply that* $\nabla_{\theta^\mu}\mu(s|\theta^\mu)$ *and* $\nabla_a Q(s, a|\theta^Q)$ *exist and that the deterministic policy gradient exists. Then,*

$$\begin{aligned}\nabla_{\theta^\mu}\mu &\approx \int_S h^{\mu'}(s_t)\nabla_a Q(s, a|\theta^Q)|_{s=s_t, a=\mu(s_t)}\nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s=s_t}ds \\ &= \mathbb{E}_{\mu'}[\nabla_a Q(s, a|\theta^Q)|_{s=s_t, a=\mu(s_t)}\nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s=s_t}]\end{aligned} \tag{40}$$

Proof. *For a greedy stochastic policy* $\mu(a|s, \theta)$ *over a continuous action space, a global maximization step is required at every time step. Rather, we employ a deterministic policy* $\mu(s|\theta)$ *and update the policy parameters by moving them in the direction of the gradient of the action-value function. We take an expectation to average*

over the suggested directions of improvement from each state w.r.t. the state distribution under the target policy μ' given by $h^{\mu'}(s)$.

$$\theta_{k+1}^{\mu} = \theta_k^{\mu} + \alpha \mathbb{E}_{\mu'^k}[\nabla_\theta Q(s, \mu(s|\theta_k^{\mu})|\theta_k^{Q})] \tag{41}$$

By applying the chain rule,

$$\theta_{k+1}^{\mu} = \theta_k^{\mu} + \alpha \mathbb{E}_{\mu'^k}[\nabla_a Q(s, a|\theta_k^{Q})|_{a=\mu(s|\theta_k^{\mu})} \nabla_\theta \mu(s|\theta_k^{\mu})] \tag{42}$$

Now, the expectation in the right-hand side is exactly what we were looking for.

Algorithm 1 DDPG Algorithm

Randomly initialize critic network $Q(s_i, a_i|\theta^Q)$ and actor $\mu\,(s\,|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process ℵ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu\,(s_t\,|\theta^\mu) + \aleph_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random batch of b transitions (s_t, a_t, r_t, s_{t+1}) from R
    Set $y_i(s) = r_i(s) + \gamma Q'\big(s_{i+1}, \mu'\big(s_{i+1}\big|\theta^{\mu'}\big)\big|\theta^{Q'}\big)$
    Update critic by minimizing the loss: $L = \frac{1}{b}\sum_i\ (y_i - Q(s_i, a_i|\theta^Q)^2)$
    Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{b}\sum_i \nabla_a Q(s, a\,|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu}\mu\,(s\,|\theta^\mu)|_{s_i}$$

    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

Now we consider a robust control problem for the MDP model of the portfolio management problem (section 2.1) then we solve for the optimal solution using a robust dynamic programming algorithm and present our modified Q update for DDPG algorithm.

The estimation errors in MDPs when finding the optimal decisions utilizing real-world data may have a huge impact on the solution, which is often quite sensitive to changes in the state transition probabilities (Nilim & El Ghaoui, 2005). Therefore, this thesis addresses the issue of uncertainty at a higher level and seek robust decisions assuming uncertain state transition probabilities.

Since the initial parameters for citric before any updates are random biased values, TD target is a biased estimate during training and could be misleading, especially when training the agent on a sample data for a limited time horizon. For instance, sampling errors and the uncertainty in estimated parameters could lead to sub optimal investment policies that increase the risk of losing capital. In such cases, we would like a robust estimation from the critic of the deployed actor-critic algorithm. To mitigate uncertainty in transition matrix rows, we assume uncertainty set $\wp$ for all the rows in the transition matrix and apply the robust recursion in the critic's TD target. So,

$$y_i(s) = r_i(s) + \gamma \, min_{P \in \wp} E_P\left[Q'\left(s_{i+1}, \mu'\left(s_{i+1}|\theta^{\mu'}\right)|\theta^{Q'}\right)\right] , \tag{43}$$

where

$$P = \hat{\mathrm{P}}\left(s_{i+1}|s_i\right), \ \wp := \{P \,|d(P, \hat{\mathrm{P}}\left(s_{i+1}|s_i\right)) \leq \delta\} \tag{44}$$

where the uncertainty set $\wp$ is assumed to have the rectangular uncertainty property. For the proof of optimality of the recursion above, we encourage the reader to read (Nilim & El Ghaoui, 2005). Each step of the robust algorithm involves the solution of an optimization problem,

$$min_{P \in \wp} E_P\left[Q'\left(s_{i+1}, \mu'\left(s_{i+1}|\theta^{\mu'}\right)|\theta^{Q'}\right)\right] \tag{45}$$

, where the empirical transition matrix $\hat{\mathrm{P}}$, and a likelihood constraint of the form

$$\sum_j \hat{\mathrm{P}}(j) \log p(j) \geq \beta \ , p \in \Delta^n \ , \tag{46}$$

, where

$$\beta < \beta_{max} \text{ and } \beta_{max} = \sum_j \hat{\mathrm{P}}(j) \log \hat{\mathrm{P}}(j) \tag{47}$$

were utilized to describe the uncertainty on each transition matrix (Lehmann & Casella, 1998) (Poor, 1998) (Nilim & El Ghaoui, 2005) and to find the solution to the equation 45 referred to as the "inner problem". The variable p corresponds to a particular row of a specific transition matrix and Δ^n is defined as the probability simplex in $\mathbb{R}^n$. β_{max} denotes the maximal value of the likelihood function appearing in the uncertainty set

$$\wp(\beta) := \{p \in \Delta^n : \sum_j \hat{\mathrm{P}}(j) \log p(j) \geq \beta\}. \tag{48}$$

(note that we have dropped the row subscript k in the lower bound β_k, the empirical frequencies vector $\widehat{\mathrm{P}_k}$ and the uncertainty set $\wp_k$)

We are now all set up to solve for

$$min_{P \in \wp} E_P\left[Q'\left(s_{i+1}, \mu'\left(s_{i+1}\middle|\theta^{\mu'}\right)\middle|\theta^{Q'}\right)\right],\ \sum_j \hat{\mathrm{P}}(j) \log p(j) \geq \beta\ , p \in \Delta^n \tag{49}$$

In experiments, β is heuristically set to $\beta = \beta_{max} - 0.1$ and fed into the optimization module in each iteration of the training phase. Finally, we use CVXPY optimization module in python to solve the optimization problem in equation 49.

4.2.1 Uncertainty sets and off-line calibration

Since we do not know the state transition probabilities in stock price data to apply the robust dynamic programming technique discussed above, we use the Empirical Transition Matrix $\hat{\mathrm{P}}$ that is estimated just from the training data. Before training begins, an unsupervised binning method (equal-width discretization) is used upon the state tensor (that we already discussed in section 4.1) with a bin size of x_i for asset i to find the frequencies of traveling from a specific state to all other possible states. So

$$x_i = \frac{max\,(v_i) - min\,(v_i)}{M} \quad , \tag{50}$$

where M is the number of bins that each state variable will be discretized into, with interval boundaries as $\{ min\,(v_i) + x_i\,, min\,(v_i) + 2x_i\,, \ldots, min\,(v_i) + Mx_i \}$.

- **Ellipsoidal models**

We also consider another uncertainty model called Ellipsoidal models that arise when second-order approximations are made to the log-likelihood function arising in the likelihood model. To ensure the rectangular uncertainty property, similar to (Nilim & El Ghaoui, 2005) we first form the projections on the space. These assume a similar shape, that of an ellipsoid intersected with the transition probability simplex, specifically,

$$\wp\,(\beta) := \{p \in \Delta^n : \sum_j \frac{(p\,(j) - \hat{\mathrm{P}}\,(j))^2}{\hat{\mathrm{P}}\,(j)} \geq k^2\}\,, \tag{51}$$

where $k^2 := 2(\,\beta_{max} - \beta\,)$. We refer to the above model as the constrained ellipsoidal model. In the constrained likelihood case, the inner problem assumes the form

$$min_{P \in \wp} E_P\left[Q'\left(s_{i+1}, \mu'\left(s_{i+1}\middle|\theta^{\mu'}\right)\middle|\theta^{Q'}\right)\right],\ \sum_j \frac{(p\,(j) - \hat{\mathrm{P}}\,(j))^2}{\hat{\mathrm{P}}\,(j)} \geq k^2 \quad , p \in \Delta^n \tag{52}$$

4.2.2 Modified Q Update

We introduce the degree of the investor's robustness as $\lambda \in [0,1]$ which basically is the degree that the RL agent should be updated by the gradient of the robust value iteration (equation 20). Finally by replacing the TD target with

$$y_i(s) = r_i(s) + \gamma\,(\left(\lambda(Q'\left(s_{i+1}, \mu'\left(s_{i+1}\middle|\theta^{\mu'}\right)\middle|\theta^{Q'}\right)\right) + ((1-\lambda) min_{P \in \wp} E_P[Q'(s_{i+1}, \mu'\left(s_{i+1}\middle|\theta^{\mu'}\right))|\theta^{Q'})] \tag{53}$$

we try to immunize the learning of the portfolio management from the downside risk of the investment strategy.

4.2.3 Algorithm

Algorithm 2 Robust DDPG Algorithm

Randomly initialize critic network $Q(s_i, a_i|\theta^Q)$ and actor $\mu\ (s\ |\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
Estimate empirical transition matrix $\hat{\mathrm{P}}$, $P = \hat{\mathrm{P}}_i(s_{i+1}|s_i)$
for episode = 1, M **do**
  Initialize a random process $\aleph$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu\ (s_t\ |\theta^\mu)$ + $\aleph_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random batch of b transitions (s_t, a_t, r_t, s_{t+1}) from R
    Robust optimization and solving for $min_{P\in\wp} E_P[Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'}))|\theta^{Q'})$
    Set $y_i(s) = r_i(s) + \gamma\ (\left(\lambda(Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})\right) + ((1-\lambda) min_{P\in\wp} E_P[Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'}))|\theta^{Q'})]$
    Update critic by minimizing the loss: $L = \frac{1}{b}\sum_i\ (y_i - Q(s_i, a_i|\theta^Q)^2)$
    Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{b}\sum_i \nabla_a Q(s, a\ |\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu}\mu\ (s\ |\theta^\mu)|_{s_i}$$

    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

4.3 Implementations Details

We used TensorFlow and CVXPY libraries in python to develop and train the models described in this thesis. After training of each model, the training parameters of each model will be saved into TensorFlow checkpoints files and stored in a local directory for future use. When evaluating the models, the models will be loaded from the checkpoint files and tested on the unobserved data where we calculated the performance metrics for each model by analyzing its portfolio value and saved them into a result table. After evaluating all the models, the descriptive statistic of the performance metrics values saved in the result table is calculated and further analysis to compare the performance of the models is conducted upon that table that will be discussed in the next section.

4.4 Limitations

Before discussing the experiments and drawing any conclusion we would like to mention some challenges of this work

- We assume that the liquidity of all the assets that are being traded in the system is high enough that, each trade can be carried out immediately at the last price when an order is placed.
- We assume that the capital invested by the RL agent is insignificant, so it has no influence on the market.
- By increasing the number of assets in the portfolio, the size of the state's variable becomes larger and many of the transition probabilities in each row of the transition matrix may go to zero, thus estimating the empirical state transition matrix becomes less sensible.

In a real-world trading environment, if the trading volume in a market is high enough, these two assumptions are near to reality. In the experiments, we choose up to 3 assets in each portfolio, however, in the real-life scenario the size of the portfolio highly depends on the investor's type. Individuals typically have less than 10 assets in their portfolios and institutional investors could choose a portfolio of around a hundred assets. Also, it is worth mentioning that in order to efficiently and practically use this system and to reduce the training computations time of the algorithm in algorithmic trading setting one should have access to strong processing power e.g., GPU[2] computing power.

5. Experimental Results

For these experiments, we use price data of the listed stocks in NASDAQ100 from the US equity market which is being explored by algorithmic trading and High-Frequency-Trading (HFT) strategies intensively. In Section 5.1 we discuss how we train the portfolio management agents, DDPG and Robust DDPG, and validate the training phase just before using them for portfolio selection in the testing date range with real market data (back-testing) as described in section 5.2. In the next section, we provide the conclusion of the experiments by comparing the performances of the agents with each other and the Uniform Buy and Hold (UBAH) benchmark, one of the most fundamental investment strategies commonly used to compare against when designing new methods, in eliminating the downside risk of their investment and maximizing their final portfolio using commonly used performance measures.

[2] GPU computing is the use of a GPU (graphics processing unit) as a co-processor to accelerate CPUs for general-purpose scientific and engineering computing.

- **Maximum Drawdown (MDD)** (Atiya & Magdon-Ismail, 2004): MDD is the biggest loss from a peak to a trough, and is written mathematically as

$$MDD = \max_{\substack{\xi > t \\ t}} \frac{p_t - p_\xi}{p_t} \tag{54}$$

,99th percentile Drawdown (99DD), 95th percentile Drawdown (95DD) and 90th percentile Drawdown (90DD) were calculated to highlight the downwards deviation of the portfolio selection strategy. Total Time Under Water (TTDUW) is calculated which shows how many days in total the portfolio value is in the drawdown state and Maximum Time Under Water (MDUW) measures investment strategy's riskiness by calculating the maximum distance in time, from a previous peak to a new peak. In other words, it calculates how long it takes an investor to recover its money at the start of the maximum drawdown period and it is derived from the calculation of the MDD. These metrics evaluate the riskiness of the investment models in this thesis. In terms of the profitability, we considered the following metrics:

- **Sharpe Ratio (SR)** (Sharpe, 1964, 1994): is a risk adjusted mean return, defined as the average of the risk-free return by its deviation, and mathematically

$$S = \frac{E_t[PR_t - PR_F]}{\sqrt{\underset{t}{\text{var}}(PR_t - PR_F)}} \tag{55}$$

where $PR_t = \frac{p_t}{p_{t-1}}$ are periodic returns, and PR_F is the rate of return of a risk-free asset. In these experiments the risk-free asset is money. Because the quoted currency is also money, the risk-free return is zero, $PR_F = 0$, here. And finally, final Portfolio Value (fPV) (equation 7) that measures profitability of the portfolio management agent during the back-testing. All price data is downloaded from Kaggle's official website[3]. The codes are in Python Programming Language and deep learning models are made using the TensorFlow library and its higher-level APIs in python.

[3] https://www.kaggle.com/

5.1 Training and Validation

First, the portfolio management agent was trained for a fixed number of episodes and after validating, the optimal number of training episodes was chosen. Agents trade and learn for a fixed number of time-ordered steps. Before training begins, a random start date is chosen for all episodes and a batch of time-ordered steps are extracted from training data accordingly. Because the data sets are time-series, batches starting with different start dates are considered valid and distinctive, even if they have overlapping intervals.

One of the advantages of this setting is that it can significantly increase the amount of training data, enabling the agent to identify and avoid spurious correlations in training data points thus decreasing the chance of over-fitting. By selecting a constant *random seed* when choosing the start dates for agents' training episodes fairness in evaluating and validating the trained algorithms is insured.

Another advantage of the proposed setting for training the portfolio management agent is that it enables validation of the training process by using the accumulated rewards and average maximum value of the q-value function of each episode's $Q(s_t, a_t)$. TensorBoard (Tensorflow's visualization toolkit) is deployed to analyze these two metrics during the validation phase to find the optimal number of training episodes for the DRL agent. By this validation approach we have more control over the hyper-parameter sensitivity of the DRL models and can reduce the generalization error of the models.

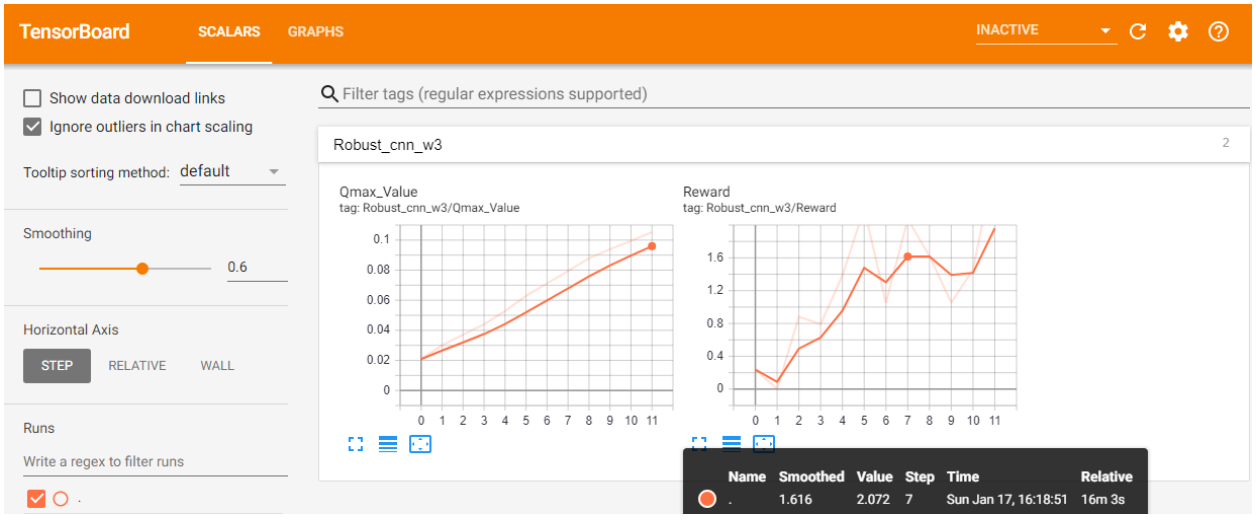

Figure 5 TensorBoard for live validation: an CNN based agent during training

5.2 Experiment 1 Back-Test Results (Multiple-Stock Portfolio)

To compare the performance of the Robust DDPG vs DDPG algorithm and UBAH (Uniform Buy And Hold) strategy, 5 years of daily price data ($\approx 2013\ to\ 2018$) of 30 stocks listed in NASDAQ100 were selected. A portfolio is randomly generated consisting of 3 assets and cash money, and training begins. We trained all the algorithms on 30 different portfolios. The date range for training for all agents is from 2012-08-15 to 2015-08-15 and the testing date range is from 2015-08-16 to 2017-08-11. The degree of the investor's robustness λ is set to 0.55 for robust algorithms. The following is the probability density plots of the performance of the Robust DDPG model and DDPG algorithms. More detailed view of the results is depicted in the appendix section (Tables 3 to 6)

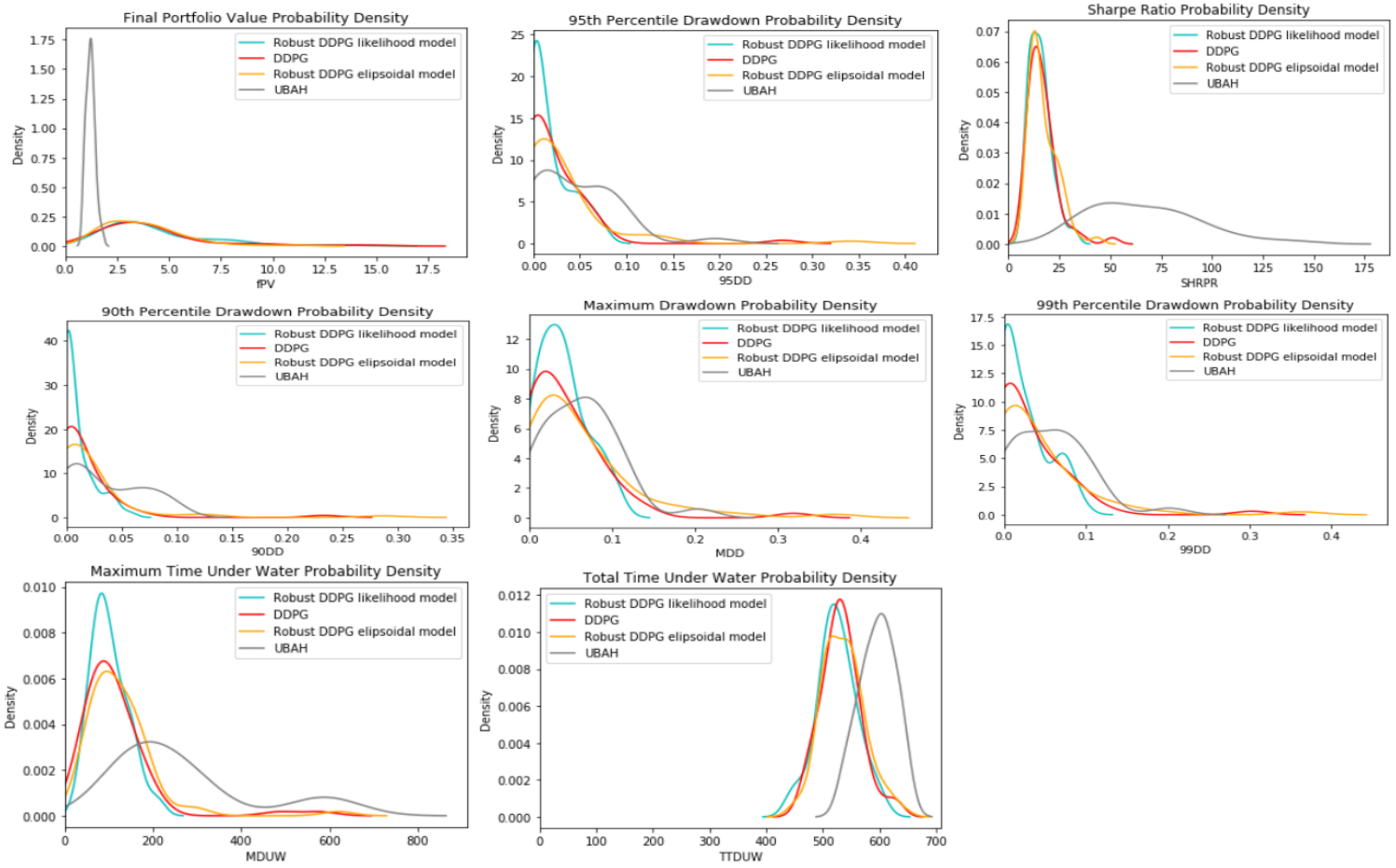

Figure 6 Performance measures probability density of the trading models - Experiment 1

Based on the results, both DDPG and robust DDPG outperformed the benchmark in all tests and on average final portfolio value of the robust DDPG models is 5 % bigger than DDPG models final portfolio value and its

approximately 4 times bigger than UBAH final portfolio value. While they both experienced a relatively low drawdown (decline in a fund) value, the preliminary results indicates that the Robust DDPG (likelihood model) can offer a safer path for the investor's portfolio value during the trading period and on reduces the riskiness of the investment strategy by mitigating the mean values of the risk proxies compared to the rest of the models. Therefore, ensuring a faster recovery from the drawdown period compared to DDPG algorithm. Although Robust DDPG with ellipsoidal model constraint outperforms the benchmark significantly, the probability density plots indicate that robust DDPG model with likelihood constraint and the non-robust DDPG model show better performance than ellipsoidal ones in terms mitigating the riskiness of the investment and profitability. The following figures compares the performance of the Robust DDPG models LSTM-based and CNN-based vs DDPG models LSTM-based and CNN-based algorithms (Tables 7 to 12).

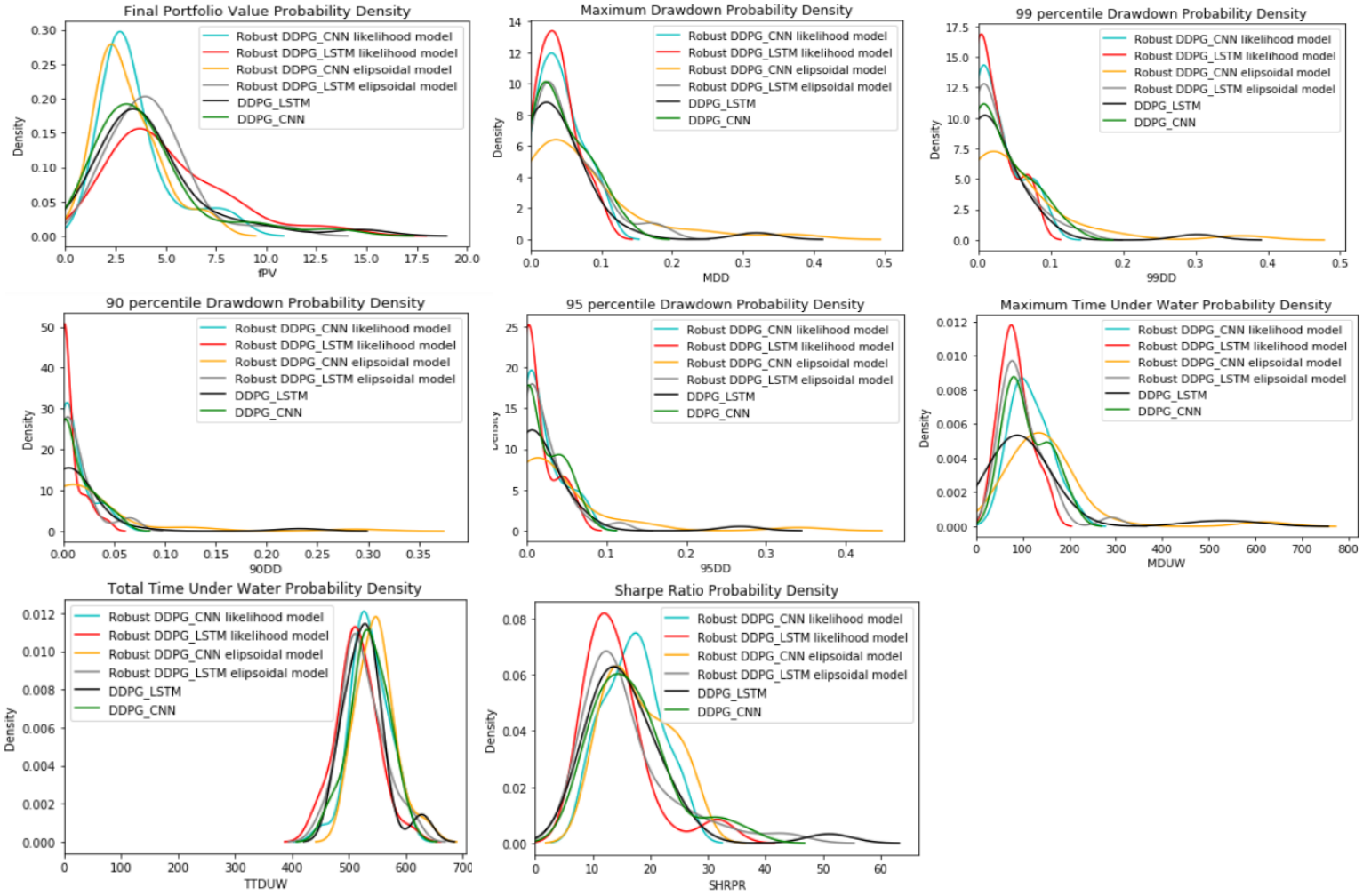

Figure 7 Performance measures probability density of the trading models (LSTM & CNN) - Experiment 1

Robust DDPG_LSTM likelihood model is the winner algorithm in this experiment and has obtained the highest portfolio value on average (5-fold return) and it has the biggest chance of reducing the riskiness of the investment on average in terms of all of the risk proxies.

5.3 Experiment 2 Back-Test Results (Single Stock Portfolio)

In order to compare the performance of the RL algorithm with dynamic programming solutions and to not run into the "curse of dimensionality" in DP algorithms, trading models were simplified to trade a single stock against the quote asset where the historical price window size is just a day and the DP model actions are eleven discrete values as follow {0, 0.1, 0.2, …, 1}. For this experiment, 5 years of daily price data (≈2013 to 2018) of 32 stocks listed in NASDAQ100 were selected. A portfolio is randomly generated consisting of 1 asset and cash money and training begins. The following is the statistical summary of the performance of the Robust DDPG likelihood model, DDPG and Q-learning tabular (TD control) algorithms in managing a single stock portfolio. More detailed view of the results is depicted in the appendix section. (Tables 13 to 18)

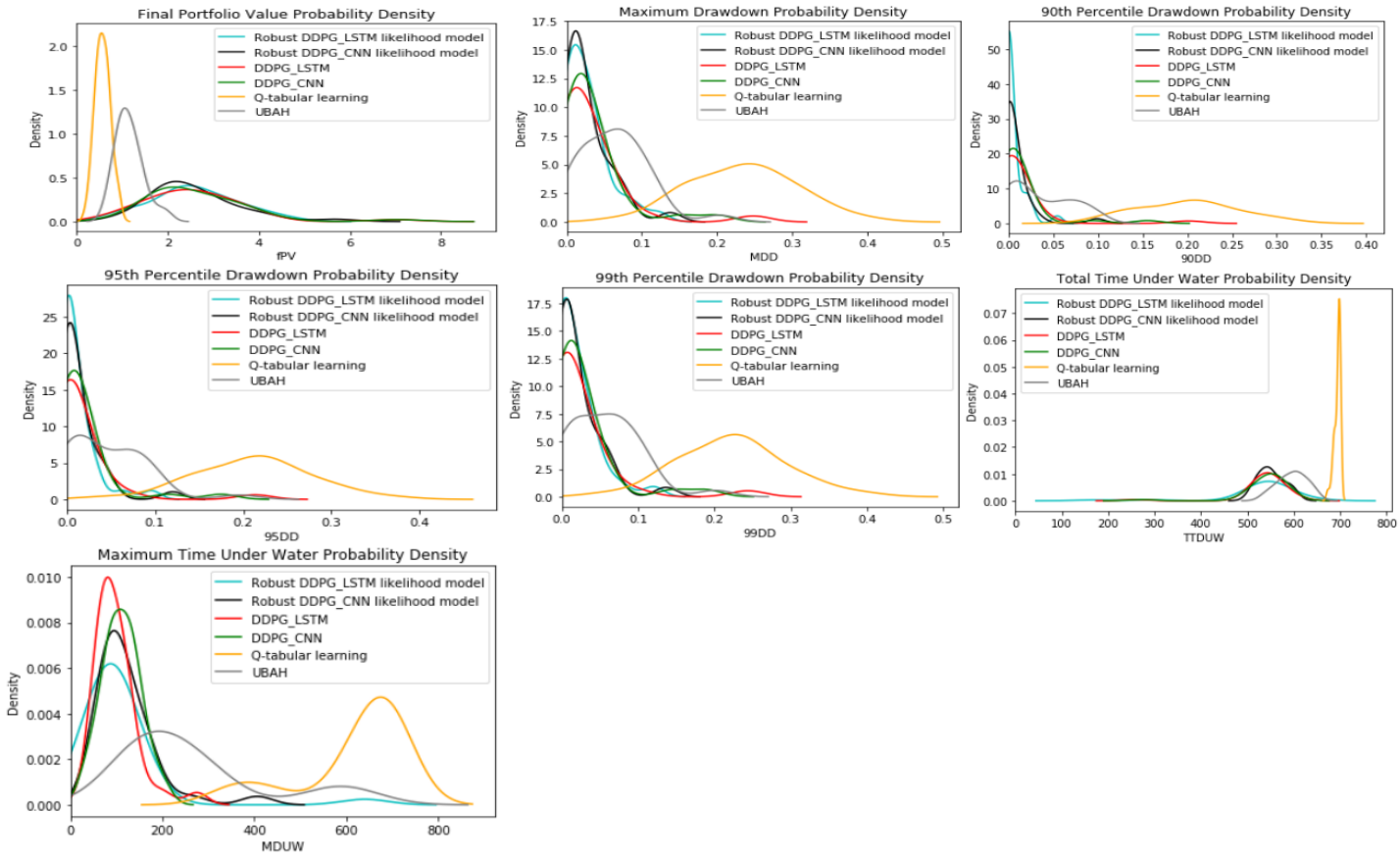

Figure 8 Probability density of performance measures - Experiment 2

The result of this experiment shows that the Robust DDPG and DDPG algorithm outperformed the benchmark and Q learning algorithm and showed almost identical final portfolio value on average. In Addition, Robust DDPG_LSTM likelihood model is outperforming all other algorithms in almost all risk proxies and Q-tabular learning algorithm have shown a poor performance comparing to the benchmark and the DRL algorithms which could emphasize the merits of using deep policy networks instead of the tabular methods.

5.4 Hyperparameter Tuning

In this section, we deploy Genetic algorithms to estimate the optimal degree of investor robustness (λ) and the size of the uncertainty set (β) in robust DDPG algorithms. Genetic algorithms represent one branch of the field of study called evolutionary computation, in that they imitate the biological processes of reproduction and natural selection to solve for the 'fittest' solutions (Goldberg, 1989). These algorithms are considered to be far more powerful and efficient than random search and exhaustive search algorithms, yet require no extra information about the given problem. Final portfolio value of the robust DDPG trading agent is considered for designing the 'fitness function' in the genetic algorithms. Each generation is populated with 40 'chromosomes'. We run the optimization algorithm for 20 'new generations' and 'selects' the best 10 chromosomes of each generation. Furthermore, The 'mutation probability' and 'gene mutation probability' of the algorithm are set to 0.2 and 0.5 respectively. Also, the range of the values that each gene could take is chosen beforehand and is fed to the optimization algorithm. Because running the optimization algorithm for all models is time-intensive, here we chose a random portfolio consist of 3 US stocks (`['CMCSA', 'COST', 'CSX']`) and compare the performance of the Robust CNN algorithm after and before choosing the best hyperparameters.

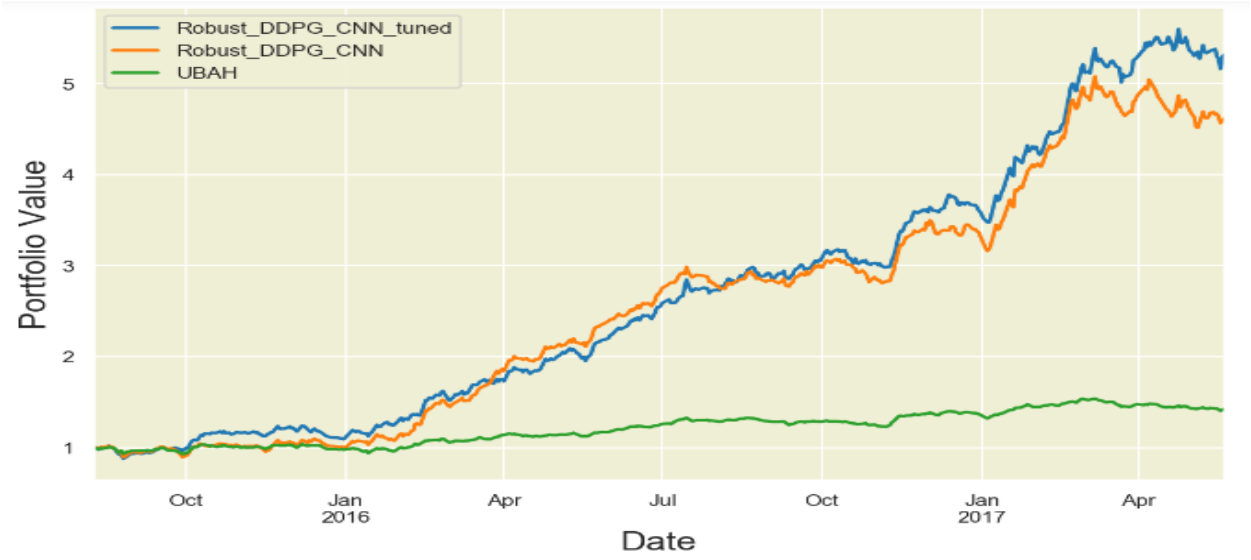

Figure 9 Portfolio value during time for the Robust DDPG_CNN model before and after hyperparameter tuning

MDD	99DD	95DD	90DD	TTDUW	MDUW	SHRPR	fPV
-0.066257	-0.048866	-0.030985	-0.028331	478	45	10.218073	5.316474

Table 1 Robust DDPG CNN after hyper-parameter optimization step where λ = 0 and β = -1

MDD	99DD	95DD	90DD	TTDUW	MDUW	SHRPR	fPV
-0.09255	-0.076007	-0.056875	-0.029471	497	74	10.721561	4.615055

Table 2 Robust DDPG CNN before hyper-parameter optimization step where λ = 0.55 and β = -2.4

By tuning the robust model, we have seen a quicker and steadier growth for portfolio value as well as a significant improvement in almost all performance measures, specifically a 15 percent growth for the final portfolio value and 40 percent decline for maximum time under water.

6. Conclusion and Future Works

Based on the back-test results both DDPG and robust DDPG outperformed the benchmark in all tests and experienced a relatively low drawdown value. In the previous section, we have demonstrated that the resulting algorithm of this research, Robust DDPG, can offer a safer path for the investor's portfolio value during the

trading period and reduce the riskiness of the investment strategy in terms of mean value of all chosen risk proxies. Therefore, ensuring a faster recovery from the drawdown period compared to the DDPG algorithm. Moreover, the Robust DDPG algorithm not only has shown successful performance in immunizing the investment strategy from the adversarial risks but also it is able to offer the same or bigger final portfolio value comparing to the DDPG algorithm. Thus, making this robust framework desirable for risk-averse investors. Comparing experiment 1 & 2, Robust DDPG models, has a bigger advantage in performance measures values (especially final portfolio value) than DDPG in the multiple stock scenario where uncertainty and the estimation errors due to the sampling technique of the DDPG is higher. Suggesting that we should see more intense advantage in robust models' performance compared to the non-robust ones because by increasing the portfolio size, estimation errors increase and mislead the learning process more than before. Furthermore, ellipsoidal models couldn't provide a consistent improvement comparing to DDPG and Robust DDPG algorithm as they have lower values for final portfolio value and almost all risk proxies in our experiments. Robust DDPG_LSTM likelihood model has the most desirable performance in terms of both profitability and risk proxies and it's the winner algorithm of this thesis and generally LSTM based agents had a better performance in both robust DDPG and DDPG algorithms compared to CNN agents in handling risk and final return of the investment making it more suitable for the portfolio management problem. Finally, using Genetic algorithm for tuning the optimal degree of investor robustness (λ) and the size of the uncertainty set (β) in robust DDPG algorithms had a significant impact on the robust model's performance specifically the final portfolio value of the model, making it a promising technique to be used in a practical application for this algorithm.

In future works, the investigation of the performance of the DDPG algorithm with different types of risk measures and a comparison in eliminating the risk of the trading actions of the software portfolio manager is also an interesting subject to the authors of this paper. Moreover, a comparison between the portfolio selection performances of the resulting algorithm with other portfolio management strategies, especially reinforcement learning ones could shed new insights about using RL for portfolio management.

Appendix

- Experiment 1 Back-Test Results (Multiple-Stock Portfolio) Detailed View

	90DD	95DD	99DD	MDD	MDUW	SHRPR	TTDUW	fPV
count	60.000000	60.000000	60.000000	60.000000	60.00000	60.000000	60.000000	60.000000
mean	0.009485	0.017578	0.025406	0.040368	101.60000	15.560903	524.966667	4.288473
std	0.014089	0.022145	0.028316	0.028964	40.96224	5.426941	35.143756	2.491453
min	0.000000	0.000000	0.000000	0.000000	33.00000	7.891323	441.000000	1.383286
25%	0.000000	0.000000	0.000000	0.018675	72.75000	10.686903	503.000000	2.700929
50%	0.002811	0.006910	0.016800	0.037290	95.00000	14.920895	523.500000	3.439520
75%	0.016033	0.028847	0.037874	0.056634	130.75000	18.304625	545.250000	4.974748
max	0.057351	0.075041	0.094972	0.106725	214.00000	32.736957	607.000000	13.609543

Table 3 Robust DDPG (both CNN and LSTM) Performance Statistics; Maximum Drawdown (MDD), 99th percentile Drawdown (99DD), 95th percentile Drawdown (95DD), 90th percentile Drawdown (90DD), Maximum Time Under Water (MDUW), Total Time Under Water (TTDUW)

	90DD	95DD	99DD	MDD	MDUW	SHRPR	TTDUW	fPV
count	60.000000	60.000000	60.000000	60.000000	60.000000	60.000000	60.000000	60.000000
mean	0.014892	0.021731	0.031934	0.043950	115.783333	16.523645	531.633333	4.105058
std	0.032997	0.039311	0.048660	0.050916	88.490178	7.386841	34.940126	2.708728
min	0.000000	0.000000	0.000000	0.000000	41.000000	7.711617	459.000000	0.642275
25%	0.000000	0.000000	0.000000	0.010183	72.000000	11.897508	511.500000	2.349915
50%	0.002385	0.005254	0.008324	0.029534	94.000000	14.911045	532.000000	3.447035
75%	0.018163	0.038463	0.047573	0.063697	134.250000	19.232925	551.250000	4.504412
max	0.232788	0.267620	0.302613	0.318934	578.000000	51.072631	630.000000	14.698328

Table 4 DDPG (both CNN and LSTM) Performance Statistics

	fPV	MDD	99DD	95DD	90DD	TTDUW	MDUW	SHRPR
count	60.000000	60.000000	60.000000	60.000000	60.000000	60.000000	60.000000	60.000000
mean	3.682823	0.060212	0.040641	0.029251	0.020137	535.716667	126.433333	16.978792
std	1.899358	0.067044	0.060065	0.052397	0.043154	36.990650	83.255247	6.851092
min	0.474100	0.000000	0.000000	0.000000	0.000000	451.000000	48.000000	9.600883
25%	2.245959	0.021331	0.000165	0.000000	0.000000	509.000000	75.750000	11.608709
50%	3.467322	0.034288	0.020310	0.010313	0.007280	534.500000	109.500000	14.549540
75%	4.561646	0.081711	0.057372	0.032370	0.020497	555.750000	157.000000	21.348051
max	10.930463	0.368663	0.362856	0.340964	0.286690	639.000000	619.000000	43.344327

Table 5 Robust DDPG (both CNN and LSTM) (ELIPSOIDAL) Performance Statistic

	90DD	95DD	99DD	MDD	MDUW	SHRPR	TTDUW	fPV
count	30.000000	30.000000	30.000000	30.000000	30.000000	30.000000	30.000000	30.000000
mean	0.035674	0.047124	0.057398	0.063775	268.066667	66.459441	596.500000	1.223200
std	0.033687	0.044381	0.045560	0.044422	156.604913	26.256074	30.726547	0.209372
min	0.000000	0.000000	0.000000	0.000000	121.000000	30.712799	535.000000	0.915815
25%	0.006567	0.008756	0.020107	0.033380	163.000000	46.110173	574.000000	1.068532
50%	0.022732	0.034826	0.056264	0.064248	214.500000	64.279871	600.000000	1.215338
75%	0.064532	0.072740	0.080339	0.084501	292.000000	81.882694	619.500000	1.336737
max	0.094912	0.195391	0.201312	0.203372	626.000000	137.872565	646.000000	1.770992

Table 6 Benchmark Performance Statistics

	90DD	95DD	99DD	MDD	MDUW	SHRPR	TTDUW	fPV
count	30.000000	30.000000	30.000000	30.000000	30.000000	30.000000	30.00000	30.000000
mean	0.006532	0.014681	0.022509	0.037782	84.833333	14.189090	514.00000	5.095583
std	0.011421	0.020241	0.026231	0.027552	32.775183	5.819720	35.32802	2.866507
min	0.000000	0.000000	0.000000	0.000000	33.000000	7.891323	441.00000	1.383286
25%	0.000000	0.000000	0.000000	0.019416	64.250000	10.359144	495.50000	3.167502
50%	0.000000	0.003257	0.012147	0.036340	79.500000	13.268438	512.00000	4.083637
75%	0.007407	0.020853	0.035481	0.050649	106.250000	15.997088	536.50000	6.722542
max	0.043371	0.062341	0.074141	0.101566	155.000000	32.736957	607.00000	13.609543

Table 7 Robust DDPG_LSTM Performance Statistics

	90DD	95DD	99DD	MDD	MDUW	SHRPR	TTDUW	fPV
count	30.000000	30.000000	30.000000	30.000000	30.000000	30.000000	30.000000	30.000000
mean	0.012438	0.020475	0.028303	0.042954	118.366667	16.932715	535.933333	3.481363
std	0.015979	0.023888	0.030428	0.030559	41.926441	4.707048	31.872519	1.750452
min	0.000000	0.000000	0.000000	0.000000	47.000000	9.986401	452.000000	1.608133
25%	0.000000	0.000586	0.003137	0.018185	84.000000	13.553689	517.500000	2.452924
50%	0.005828	0.009616	0.016892	0.039618	109.000000	17.033345	532.000000	2.966288
75%	0.017688	0.032108	0.039360	0.058985	144.000000	19.638533	560.250000	3.916719
max	0.057351	0.075041	0.094972	0.106725	214.000000	25.334492	598.000000	8.193556

Table 8 Robust DDPG_CNN Performance Statistics

	90DD	95DD	99DD	MDD	MDUW	SHRPR	TTDUW	fPV
count	30.000000	30.000000	30.000000	30.000000	30.000000	30.000000	30.000000	30.000000
mean	0.016332	0.022198	0.029149	0.042157	120.800000	16.217193	527.900000	4.217863
std	0.043612	0.050659	0.058099	0.061723	117.626938	8.018151	36.421242	2.821156
min	0.000000	0.000000	0.000000	0.000000	41.000000	7.711617	476.000000	0.642275
25%	0.000000	0.000000	0.000000	0.001528	57.500000	12.161084	505.500000	2.590129
50%	0.002385	0.003071	0.005254	0.025303	88.500000	14.135425	524.000000	3.479875
75%	0.012111	0.022205	0.039627	0.055486	118.500000	18.590443	544.500000	4.564227
max	0.232788	0.267620	0.302613	0.318934	578.000000	51.072631	630.000000	14.698328

Table 9 DDPG_LSTM Performance Statistics

	90DD	95DD	99DD	MDD	MDUW	SHRPR	TTDUW	fPV
count	30.000000	30.000000	30.000000	30.000000	30.000000	30.000000	30.000000	30.000000
mean	0.013453	0.021263	0.034720	0.045743	110.766667	16.830097	535.366667	3.992254
std	0.017572	0.024025	0.037758	0.038184	45.198172	6.821097	33.591341	2.634800
min	0.000000	0.000000	0.000000	0.000000	47.000000	7.807995	459.000000	1.487912
25%	0.000000	0.000000	0.000224	0.017371	75.500000	11.434344	518.000000	2.274893
50%	0.004541	0.006843	0.025189	0.036915	97.500000	15.450793	534.000000	3.366814
75%	0.024908	0.041204	0.060310	0.072249	146.750000	19.727131	559.750000	4.473102
max	0.058286	0.076050	0.128452	0.137721	201.000000	36.459644	603.000000	13.329305

Table 10 DDPG_CNN Performance Statistics

	fPV	MDD	99DD	95DD	90DD	TTDUW	MDUW	SHRPR
count	30.000000	30.000000	30.000000	30.000000	30.000000	30.000000	30.000000	30.000000
mean	4.252538	0.049911	0.029909	0.020827	0.013272	523.733333	100.600000	16.004402
std	2.062374	0.045839	0.037193	0.027076	0.019002	37.321468	49.838775	7.943014
min	1.321857	0.000000	0.000000	0.000000	0.000000	451.000000	48.000000	9.600883
25%	3.063356	0.020411	0.000055	0.000055	0.000055	503.000000	73.250000	11.381849
50%	4.029470	0.031036	0.016403	0.011378	0.004494	517.000000	82.000000	13.631536
75%	5.010557	0.076053	0.043381	0.029822	0.019043	545.750000	115.250000	15.560570
max	10.930463	0.182589	0.142249	0.116367	0.068561	612.000000	291.000000	43.344327

Table 11 Robust DDPG_LSTM (ELIPSOIDAL) Performance Statistics

	fPV	MDD	99DD	95DD	90DD	TTDUW	MDUW	SHRPR
count	30.000000	30.000000	30.000000	30.000000	30.000000	30.000000	30.000000	30.000000
mean	3.113107	0.070514	0.051373	0.037676	0.027002	547.700000	152.266667	17.953182
std	1.553897	0.082608	0.075620	0.068598	0.057707	33.072699	101.179231	5.517032
min	0.474100	0.000000	0.000000	0.000000	0.000000	492.000000	57.000000	10.239527
25%	2.109137	0.022076	0.004230	0.000868	0.000000	524.250000	98.500000	14.150615
50%	2.480344	0.047080	0.024459	0.010313	0.009338	547.000000	141.500000	16.800645
75%	4.003971	0.088063	0.069976	0.036318	0.022416	562.250000	168.250000	22.243291
max	7.100667	0.368663	0.362856	0.340964	0.286690	639.000000	619.000000	28.118175

Table 12 Robust DDPG_CNN (ELIPSOIDAL) Performance Statistics

- Experiment 2 Back-Test Results (Single Stock Portfolio) Detailed View

	fPV	MDD	99DD	95DD	90DD	TTDUW	MDUW	SHRPR
count	32.000000	32.000000	32.000000	32.000000	3.200000e+01	32.000000	32.000000	3.200000e+01
mean	2.597072	0.030694	0.023129	0.015988	1.166601e-02	537.250000	99.281250	1.446642e+05
std	1.156337	0.047423	0.046437	0.039702	3.592661e-02	56.408075	46.552942	5.691768e+05
min	0.894098	0.000000	0.000000	0.000000	0.000000e+00	259.000000	37.000000	9.767491e+00
25%	1.838079	0.002182	0.000000	0.000000	0.000000e+00	526.000000	68.750000	1.696767e+01
50%	2.446950	0.014960	0.004794	0.000257	2.579072e-07	544.500000	86.500000	2.014120e+01
75%	3.198763	0.044151	0.028004	0.017934	9.352820e-03	558.750000	120.500000	2.696418e+01
max	6.975094	0.248103	0.243664	0.212991	2.009444e-01	613.000000	275.000000	2.331217e+06

Table 13 DDPG_LSTM Performance Statistics (single stock portfolio)

	fPV	MDD	99DD	95DD	90DD	TTDUW	MDUW	SHRPR
count	32.000000	32.000000	32.000000	32.000000	32.000000	32.000000	32.000000	3.200000e+01
mean	2.644146	0.033580	0.027090	0.018853	0.013858	540.218750	110.656250	9.218236e+04
std	1.129202	0.042739	0.040974	0.035813	0.031202	55.179902	40.961516	5.213371e+05
min	1.000007	0.000000	0.000000	0.000000	0.000000	273.000000	28.000000	1.037206e+01
25%	1.915056	0.009533	0.001174	0.000000	0.000000	529.500000	86.000000	1.559583e+01
50%	2.333681	0.023744	0.014348	0.008413	0.005781	551.500000	111.500000	2.058597e+01
75%	3.181885	0.040044	0.029356	0.020104	0.010177	563.500000	135.750000	2.842947e+01
max	7.026407	0.199907	0.189465	0.174910	0.155317	593.000000	205.000000	2.949150e+06

Table 14 DDPG_CNN Performance Statistics (single stock portfolio)

	fPV	MDD	99DD	95DD	90DD	TTDUW	MDUW	SHRPR
count	32.000000	32.000000	32.000000	32.000000	3.200000e+01	32.00000	32.000000	3.200000e+01
mean	2.583664	0.026184	0.019930	0.011285	5.246050e-03	530.50000	107.343750	1.165608e+05
std	0.931719	0.029290	0.027294	0.020876	1.160636e-02	85.13405	103.308685	4.684108e+05
min	0.938528	0.000000	0.000000	0.000000	0.000000e+00	172.00000	17.000000	1.229766e+01
25%	2.135573	0.002290	0.000050	0.000000	0.000000e+00	526.75000	66.750000	1.553956e+01
50%	2.460395	0.019653	0.010410	0.000752	4.839704e-09	548.00000	90.500000	1.956892e+01
75%	3.159440	0.037391	0.031547	0.014175	2.753166e-03	556.75000	110.250000	2.481605e+01
max	4.618129	0.121368	0.120205	0.095755	5.456001e-02	647.00000	640.000000	2.241098e+06

Table 15 robust DDPG_LSTM Performance Statistics (single stock portfolio)

	fPV	MDD	99DD	95DD	90DD	TTDUW	MDUW	SHRPR
count	32.000000	32.000000	3.200000e+01	32.000000	32.000000	32.00000	32.000000	32.000000
mean	2.561746	0.027138	2.103840e-02	0.013591	0.007459	551.03125	121.562500	29072.851846
std	0.953165	0.030527	2.945208e-02	0.024036	0.018594	28.02359	68.104728	164328.976906
min	0.999997	0.000000	0.000000e+00	0.000000	0.000000	502.00000	58.000000	10.879167
25%	1.973354	0.008481	3.440747e-11	0.000000	0.000000	530.00000	82.000000	17.582279
50%	2.326137	0.015998	8.600387e-03	0.004054	0.000012	547.00000	97.500000	21.689107
75%	2.987332	0.038109	3.136067e-02	0.014842	0.004653	570.25000	138.750000	25.703548
max	5.662727	0.137735	1.366166e-01	0.120047	0.098797	605.00000	406.000000	929608.387464

Table 16 robust DDPG_CNN Performance Statistics (single stock portfolio)

	MDD	99D	95DD	90DD	SHRPR	fPV	TTDUW	MDUW
count	32.000000	32.000000	32.000000	32.000000	32.000000	32.000000	32.000000	32.000000
mean	0.233810	0.222658	0.210382	0.199630	40.415305	0.573727	693.906250	616.500000
std	0.070242	0.068711	0.065659	0.054670	19.100861	0.160823	6.649566	116.529382
min	0.075786	0.064046	0.046158	0.097881	12.279697	0.241612	673.000000	330.000000
25%	0.170476	0.165991	0.156769	0.150370	27.447046	0.467366	689.500000	613.500000
50%	0.230821	0.221240	0.211727	0.203419	35.244042	0.564158	697.000000	667.000000
75%	0.276547	0.258969	0.246455	0.233710	55.446641	0.669250	699.000000	699.000000
max	0.390907	0.388981	0.361608	0.314609	81.095232	0.921909	699.000000	699.000000

Table 17 Q-Tabular Learning Performance Statistics

	fPV	MDD	99DD	95DD	90DD	TTDUW	MDUW	SHRPR
count	32.000000	32.000000	32.000000	32.000000	32.000000	32.000000	32.000000	32.000000
mean	1.156208	0.082900	0.073472	0.064488	0.056644	602.562500	332.781250	73.210873
std	0.303547	0.063308	0.063887	0.060353	0.054313	30.997333	179.240105	32.896631
min	0.695306	0.007485	0.000000	0.000000	0.000000	541.000000	101.000000	26.928511
25%	0.971608	0.028997	0.023195	0.015606	0.012557	578.500000	162.250000	49.727676
50%	1.090505	0.076824	0.056491	0.049175	0.042188	610.000000	321.500000	73.511333
75%	1.284907	0.116346	0.107513	0.097001	0.088683	626.250000	461.500000	90.935436
max	1.994090	0.259756	0.252973	0.238990	0.210795	649.000000	639.000000	176.364558

Table 18 Benchmark Performance Statistics (single stock portfolio)

References

Alexander Shapiro, Anton Kleywegt. (2002). Minimax analysis of stochastic problems. *Optim. Methods Software*.

Amir F Atiya, Malik Magdon-Ismail. (2004). Maximum drawdown. *Risk Magazine*.

Ankit Dangi. (2013). Financial Portfolio Optimization: Computationally guided agents to investigate, analyse and invest!?

Arnab Nilim, Laurent El Ghaoui. (2005). Robust Control of Markov Decision Processes with Uncertain Transition Matrices. *INFORMS*.

Avraam Tsantekidis,Nikolaos Passalis, Anastasios Tefas, Juho Kanniainen,Moncef Gabbouj, Alexandros Iosifidis. (2017). Forecasting stock prices from the limit order book using convolutional neural networks. *IEEE 19th Conference on Business Informatics*.

Richard S. Sutton, Andrew G. Barto. (1998). *Reinforcement Learning: An Introduction.*

David Ardia, Kris Boudt, Peter Carl, Katharine M. Mullen, Brian Peterson. (2010). Differential evolution (deoptim) for non-convex portfolio optimization.

David E. Goldberg. (1989). Genetic Algorithms in Search, Optimization, and Machine Learning.

David Silver, Guy Lever,Nicolas Heess, Thomas Degris, Daan Wierstra, Martin Riedmiller. (2014). Deterministic Policy Gradient Algorithms. *ICML*.

Diederik P. Kingma, Jimmy Ba. (2014). Adam: A Method for Stochastic Optimization.

Dimitri P. Bertsekas, John N. Tsitsiklis. (1996). Neuro-Dynamic Programming.

E. L. Lehmann, George Casella. (1998). Theory of Point Estimation. *Springer*.

Patrick Emami. (2016). *Deep Deterministic Policy Gradients in TensorFlow.*

Eugene A. Feinberg, Adam Shwartz. (2002). Handbook of Markov Decision Processes, Methods and Applications.

G. E. Uhlenbeck , L. S. Ornstein. (1930). On the Theory of the Brownian Motion.

H. Vincent Poor. (1998). An Introduction to Signal Detection and Estimation. *Springer*.

Harry Markowitz . (1952). Portfolio Selection. *The Journal of Finance*.

http://alexlenail.me/NN-SVG/LeNet.html. (n.d.). *Publication-ready NN-architecture schematics.*

Ioannis Karatzas, Robert E. Fernholz. (2008). Stochastic portfolio theory: An overview. *Handbook of Numerical Analysis*.

James Cumming. (2006). An investigation into the use of reinforcement learning techniques within the algorithmic trading domain. *Master's Thesis, Imperial College*.

James Tobin, Donald D. Hester . (1967). Risk Aversion and Portfolio Choice. *John Wiley and Sons, Inc.*

John Moody, Matthew Saffell. (2001). Learning to trade via direct reinforcement. *IEEE Transactions on Neural Networks*.

Luckyson Khaidem, Snehanshu Saha, Sudeepa Roy Dey. (2016). Predicting the direction of stock market prices using random forest.

Markowitz, H. M. (1968). Portfolio selection: efficient diversification of investments.

Martin L. Puterman. (1994). Markov Decision Processes: Discrete Stochastic Dynamic Programming. *Wiley-Interscience*.

Michael A. H. Dempster, V. Leemans. (2006). An automated FX trading system using adaptive reinforcement learning. *Expert Systems with Applications*.

Okkes Ertenlice, Can B. Kalayci. (2018). A survey of swarm intelligence for portfolio optimization: Algorithms and applications.

Robert E. Fernholz. (2002). Stochastic Portfolio Theory.

Robert Kissell. (2020). *Algorithmic Trading Methods.*

Andrzej Ruszczy´nski. (2010). Risk-averse dynamic programming for Markov decision processes. *Mathematical programming*.

Saud Almahdi, Steve Y. Yang. (2017). An adaptive portfolio trading system: A risk-return portfolio optimization using recurrent reinforcement learning with expected maximum drawdown. *Expert Systems with Applications*.

Seyed Taghi Akhavan Niaki, Saeid Hoseinzade. (2013). Forecasting S&P 500 index using artificial neural networks and design of experiments. *Journal of Industrial Engineering International*.

William F. Sharpe. (1994). The sharpe ratio. *The journal of portfolio management*.

Stan Uryasev. (2000). Conditional Value-at-Risk: optimization algorithms and applications.

Stephen A. Ross. (1976). The arbitrage theory of capital asset pricing. *Journal of Economic Theory*.

T.C. Johnson. (2007). *Utility Functions.* C2922 Economics.

Thiemo Krink, Sandra Paterlini. (2011). Multiobjective optimization using differential evolution for real-world portfolio optimization.

Thomas G Fischer. (2018). Reinforcement learning in financial markets - a survey. *Working Paper, FAU Discussion Papers in Economics*.

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

William F. Sharpe . (1964). Capital asset prices: A theory of market equilibrium under conditions of risk. *The journal of finance*.

www.reportlinker.com. (2021). *Algorithmic Trading Market Research Report by Trading Type, by Component, by Deployment, by Organisation Size, by End User - Global Forecast to 2025 - Cumulative Impact of COVID-19.* https://www.reportlinker.com/.

Xavier Glorot, Antoine Bordes, Yoshua Bengio. (2011). Deep Sparse Rectifier Neural Networks. *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*.

Yong Hu, Kang Liu, Xiangzhou Zhang, Lijun Su. (2015). Application of Evolutionary Computation for Rule Discovery in Stock Algorithmic Trading: A Literature Review.

Yue Deng, Feng Bao, Youyong Kong, Zhiquan Ren, Qionghai Dai. (2017). Deep Direct Reinforcement Learning for Financial Signal Representation and Trading. *IEEE Transactions on Neural Networks and Learning Systems*.

Yves Crama, Michael Schyns. (2003). Simulated annealing for complex portfolio selection problems.

Zhengyao Jiang, Dixing Xu, Jinjun Liang. (2017). A Deep Reinforcement Learning Framework for the Financial Portfolio Management Problem.