TECHNICAL UNIVERSITY OF CRETE
SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING

Reinforcement Learning for Financial Portfolio Management

Diploma Thesis

Antonios Vogiatzis

COMMITTEE:
Michail G. Lagoudakis, Associate Professor (ECE)
Georgios Chalkiadakis, Associate Professor (ECE)
Michail Doumpos, Professor (PEM)

Chania, February 2019

ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ
ΣΧΟΛΗ ΗΛΕΚΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Ενισχυτική Μάθηση για διαχείριση Οικονομικού Χαρτοφυλακίου

Διπλωματικη Εργασια

Αντώνιος Βογιατζής

ΕΠΙΤΡΟΠΗ:
Μιχαηλ Γ. Λαγουδακης, Αναπληρωτής Καθηγητής (ΗΜΜΥ)
Γεωργιος Χαλκιαδακης, Αναπληρωτής Καθηγητής (ΗΜΜΥ)
Μιχαηλ Δουμπος, Καθηγητής (ΜΠΔ)

Χανιά, Φεβρουάριος 2019

Abstract

Portfolio Optimization saw a huge flood of interest in recent years, because of the rapidly growing ability of modern computers. The financial community continually seeks outstanding techniques from other fields to enhance financial market modelling. In this thesis, we propose a machine learning approach to the portfolio optimization problem, which calls for optimizing the allocation of capital across various financial assets, such as bonds, stocks, or funds, to maximize a preferred performance metric, such as expected returns or risk-adjusted return. We use the reinforcement learning framework, which offers innovative methods of learning good decision making policies that maximize an autonomous agent's performance in an unknown and uncertain environment. Using state-of-the-art technology based on policy gradient and deep neural networks, we developed and implemented a portfolio trading system with reinforcement learning. Subsequently, we assessed the success of our portfolio management approach and evaluated its performance using real data from the Standard & Poor's 500 repositories of the American stock market. The results we obtained achieve about 2% more wealth in the best scenario compared to the baseline models. We believe that the proposed approach outlines several metrics as evaluation indices and could expand in the future for adaptive solutions to specific cases of portfolio management, delivering better performance.

Περίληψη

Τα τελευταία χρόνια παρατηρείται τεράστια αύξηση ενδιαφέροντος για βελτιστοποίηση χαρτοφυλακίου (portfolio optimization), λόγω της ραγδαία αναπτυσσόμενης ικανότητας των σύγχρονων υπολογιστών. Η οικονομική κοινότητα επιδιώκει συνεχώς τις καινοτόμες τεχνικές από άλλους τομείς για τη βελτίωση της μοντελοποίησης της χρηματοπιστωτικής αγοράς. Στην παρούσα διπλωματική εργασία προτείνουμε μια προσέγγιση μηχανικής μάθησης στο πρόβλημα βελτιστοποίησης χαρτοφυλακίου, το οποίο καλεί για βελτιστοποίηση της κατανομής κεφαλαίων σε διάφορα χρηματοοικονομικά περιουσιακά στοιχεία, όπως ομόλογα, μετοχές ή κεφάλαια, με στόχο τη μεγιστοποίηση μιας προτιμώμενης μετρικής επιδόσεων, όπως αναμενόμενη απόδοση ή προσαρμοσμένη απόδοση βάσει ρίσκου. Χρησιμοποιούμε το πλαίσιο ενισχυτικής μάθησης (reinforcement learning),το οποίο προσφέρει καινοτόμες μεθόδους εκμάθησης ορθών πολιτικών λήψης αποφάσεων που μεγιστοποιούν τις επιδόσεις ενός αυτόνομου πράκτορα σε ένα άγνωστο και αβέβαιο περιβάλλον. Χρησιμοποιώντας τεχνολογία αιχμής που βασίζεται σε τεχνικές policy gradient και deep neural networks, αναπτύξαμε και υλοποιήσαμε ένα σύστημα διαχείρισης χαρτοφυλακίων με ενισχυτική μάθηση. Στη συνέχεια, αξιολογήσαμε την επιτυχία της προτεινόμενης προσέγγισης και αξιολογήσαμε την απόδοσή της χρησιμοποιώντας πραγματικά δεδομένα από τους καταλόγους Standard & Poor's 500 της αμερικανικής χρηματιστηριακής αγοράς. Τα αποτελέσματα που λάβαμε καταγράφουν περίπου 2% περισσότερο πλούτο σε σύγκριση με τα βασικά μοντέλα. Πιστεύουμε ότι η προτεινόμενη προσέγγιση σκιαγραφεί διάφορες μετρικές ως δείκτες αξιολόγησης και θα μπορούσε να επεκταθεί στο μέλλον για προσαρμοστικές λύσεις σε συγκεκριμένες περιπτώσεις διαχείρισης χαρτοφυλακίου, παρέχοντας καλύτερες επιδόσεις.

Acknowledgements

First, I would like to thank my advisor Michail G. Lagoudakis for his trust and guidance during the course of this thesis, and Michail Doumpos for the help with the economic domain.

My friends from Chania, Gianni, Giorgo, Niko and especially Maria during my thesis writing. Together we had some amazing moments during the last six years.

Last, but not least, I would like to thank my family for their love, support, and constant encouragement.

Contents

List of Figures

List of Algorithms

Chapter 1

Introduction

The Portfolio Management Problem is to optimize the allocation of capital across various financial assets such as bonds, stocks or funds to maximize a preferred performance metric, like expected returns or risk-adjusted returns [1]. It was a matter of interest for the financial society to seek the outstanding techniques from others areas to improve the modelling of the financial markets, as econometrics and machine learning were very close and related concepts (Corea, Francesco, [2017]). In this thesis we study and evaluate the use in the portfolio optimization domain of reinforcement learning techniques.

1.1 Thesis Contribution

This thesis depicts an approach developed for portfolio optimization with reinforcement learning. The purpose of this report is to improve the efficiency of asset allocation training agents. An internal representation (model) of the markets is developed for a finite universe of financial instruments, asset, such as stocks, which enables it to determine optimally how finite budget funds should be allocated to those assets. The agent is trained on real information about the market (historical inventory prices). The performance is then compared on a 10 stock data set of Standard & Poor's 500 companies with a standard portfolio management benchmark.

1.2 Thesis Overview

Finally, we give a brief overview of the content of our thesis:

- In Chapter 2 we present all the background information needed for this thesis. We give an overview of the terms and concepts of Stock Market and Portfolio Management, and the concepts we will utilize for Reinforcement learning .
- In Chapter 3 we state our Portfolio optimization problem and discuss the related work. We will also present the appropriate material for the policy gradient.
- In Chapter 4 we describe our optimization methods which are based on Deep Deterministic Policy Gradient (DDPG).
- In Chapter 5 performance is evaluated and compared with some baseline methods of the proposed approach.
- Finally, Chapter 6 acts as an epilogue for this theory and presents our findings and future improvements.

Chapter 2

Background

2.1 Financial market

2.1.1 Asset

An **asset** represents an economic resource for a company. Examples of assets are stocks, money on hands or deposit, bank loans, corporate bonds, mutual funds, etc. We concentrate mainly on stocks and money, but the principles cover all kinds of assets.

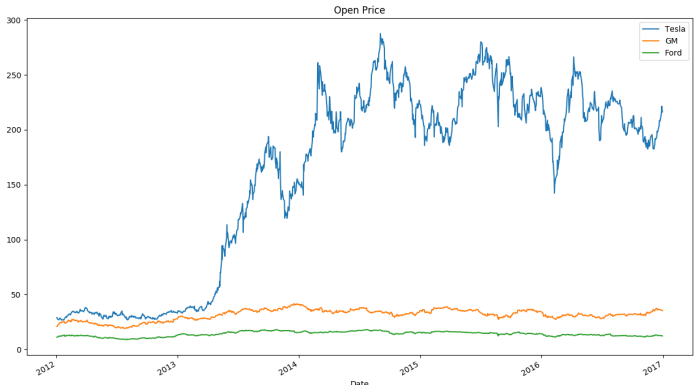
2.1.2 Portfolio

A **portfolio** is a asset grouping with certain attributes:

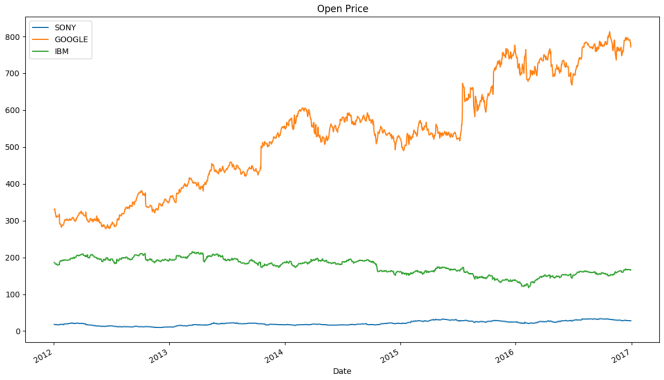
- **Component:** M assets that comprise
- **Portfolio vector, $\mathbf{w}_t$:** the $i-th$ index illustrates the proportion of the entire investment to the $i-th$ asset:

$$\mathbf{w}_t = \left[w_{1,t}, w_{2,t}, ..., w_{M,t}\right]^T \in \mathcal{R}^M \quad and \quad \sum_{i=1}^{M} \mathbf{w}_{i,t} = 1 \quad and \quad \mathbf{w}_{i,t} \geq 0 \tag{2.1}$$

  In order to reduce risk, portfolios are preferable to single assets [2].



(a) prices series 1



(b) prices series 2

Figure 2.1: Time Series

2.2 Financial Time-Series

The changing nature of the economy is causing prices to develop over time as a result of the changeable balance of supply and demand. That utilize technological approaches and tools for analysis and modelling to treat market dynamics as a time series.

2.2.1 Prices

Let $\mathbf{p}_t$ be the **price** of an asset at discrete time index t, then the sequence $\mathbf{p}_1$, $\mathbf{p}_2$, ..., $\mathbf{p}_{T-1}$, $\mathbf{p}_T$, is a time series. Examples of asset prices series are shown in Figure 2.1:

The **price vector** $\mathbf{p}_t$ is defined, as follows:

$$\mathbf{p}_t = \left[p_{1,t}, p_{2,t}, ..., p_{M,t}\right] \in \mathcal{R}^M \tag{2.2}$$

It can be more intuitive to work with such time series, as people are used to thinking regarding prices. Price time series, however, have some disadvantages. Prices are generally only positive, making it harder to use models and approaches requiring or generating negative figures. Moreover, price timesets are generally non-stationary, which means that their statistical characteristics are less stable over time.

An alternative way is to adopt time series that do not correspond to actual value but change the asset's currency value. These time series assume negative values, and their statistical characteristics are usually more stable than the price series characteristics.

2.2.2 Returns

Because of the uncertainty of the financial markets [3], future asset prices are often treated as random variables. Correspondingly, useful information can be obtained by examining the statistical properties of them. Measuring all variables in a commensurate metric is the goal of this paragraph. The most frequently used forms of **returns** are presented below.

- **Simple Return :** $\mathbf{r}_{relative}$ of an asset defined as

$$\mathbf{r}_{relative}(t) = \frac{p(t) - p(t-1)}{p(t-1)} = \frac{p(t)}{p(t-1)} - 1 \tag{2.3}$$

- **Log return:** $\mathbf{r}_{log-return}$ of an asset defined as

$$\mathbf{r}(t) = \log\left(\frac{p(t)}{p(t-1)}\right) \tag{2.4}$$

where p(t) is at t the value of the asset .

For example, if $p(t) = 101$ and $p(t-1) = 100$ then $\mathbf{r}_{relative}(t) = \frac{101-100}{100} = 1\%$.

There are various reasons for using log-returns in the industry and certain are linked to long-term assumptions about the behavior of asset returns and are outside of our scope. However, two very interesting properties are worth highlighting. Log returns are additive and make it easier for our time series to be processed, whereas relative returns

do not. In the following equation we can see the additiveness of the log results. Which is simply the log-return from $\mathbf{t}_1$ to $\mathbf{t}_2$.

$$\mathbf{r}(t_1) + \mathbf{r}(t_2) = \log\left(\frac{\mathbf{p}(t_1)}{\mathbf{p}(t_0)}\right) + \log\left(\frac{\mathbf{p}(t_2)}{\mathbf{p}(t_1)}\right) = \log\left(\frac{\mathbf{p}(t_2)}{\mathbf{p}(t_0)}\right) \tag{2.5}$$

Secondly, log returns are roughly the same as relative values of $\frac{\mathbf{p}(t)}{\mathbf{p}(t-1)}$ adequately close to 1. By expanding Taylor's 1st-order log $\frac{\mathbf{p}(t)}{\mathbf{p}(t-1)}$ around 1, we get:

$$\log\frac{\mathbf{p}(t)}{\mathbf{p}(t-1)} \simeq \log(1) + \frac{\mathbf{p}(t)}{\mathbf{p}(t-1)} - 1 = \mathbf{r}_{relative}(t) \tag{2.6}$$

- **Portfolio Return:** Let the **returns vector R be** $= \left[R_t^{(1)}, R_t^{(2)}, ..., R_t^{(M)}\right]^T \in \Re^M$, Where R^M is the simple return of the t time index of i^{th} asset. Then the **simple return portfolio** is set as the weighted sum of the component returns

$$\begin{aligned}\mathbf{R}_t^{(p)} = \frac{\mathbf{p}_t^{(p)}\mathbf{p}_{t-1}^{(p)}}{\mathbf{p}_{t-1}^{(p)}} &= w_1 \times \mathbf{R}_t^{(1)} + w_2 \times \mathbf{R}_t^{(2)} + ... + w_M \times \mathbf{R}_t^{(M)} \\ &= \sum_{i=1}^{M} w_i \times R_t^{(i)} = w^T R_t\end{aligned} \tag{2.7}$$

  Then the **portfolio returns time series** $R^{(p)} \in R^{T-1}$, is the dot product of $\mathbf{R}$ and $\mathbf{w}$

$$\mathbf{R}^{(p)} = \begin{bmatrix}\mathbf{R}_1^{(p)} \\ \mathbf{R}_2^{(p)} \\ \vdots \\ \mathbf{R}_{T-1}^{(p)}\end{bmatrix} = \mathbf{R} \times w = \begin{bmatrix}\mathbf{R}_1^{(1)} & \mathbf{R}_1^{(2)} & ... & \mathbf{R}_1^{(M)} \\ \mathbf{R}_2^{(1)} & \mathbf{R}_2^{(2)} & ... & \mathbf{R}_2^{(M)} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{R}_{T-1}^{(1)} & \mathbf{R}_{T-1}^{(2)} & ... & \mathbf{R}_{T-1}^{(M)}\end{bmatrix} \times \begin{bmatrix}\mathbf{w}_1 \\ \mathbf{w}_2 \\ \vdots \\ \mathbf{w}_M\end{bmatrix} \tag{2.8}$$

- The **portfolio log return** $\mathbf{r_t^{(p)}}$ is defined to be:

$$\mathbf{r}_t^{(p)} = \ln(\mathbf{p}_t^{(p)}) - \ln(\mathbf{p}_{t-1}^{(p)}) = \ln\left(\frac{\mathbf{p}_t^{(p)}}{\mathbf{p}_{t-1}^{(p)}}\right) = \ln(1 + \mathbf{w}^T \times \mathbf{R}_t) \tag{2.9}$$

2.3 Reinforcement Learning

Reinforcement (RL) is a machine learning area which focuses on how software agents take action in an environment to maximize some sense of cumulative rewards. It was used mainly in games (e.g. Atari), with performance that matched or even exceeded people. Currently, the combination of neural networks has enabled the algorithm to resolve more complicated tasks.In addition to the problems of the interest in reinforcement learning, optimal control theory has been examined mainly in order to establish and characterize the optimal solutions and algorithms for exact calculation and less in terms of learning or approximation, particularly in the lack of a mathematical environment model. Economics and game theory can use reinforcement learning to explain how balance can arise under limited reasonableness. Machine learning usually consists of a Markov Decision Process (MDP) environment, as many reinforcement learning algorithms use dynamic programming techniques for this context.

Reinforce learning defers from supervised learning problems, as the agent is not explicitly provided with good and bad behaviour. This makes it much more appropriate for mutual problems because it can often be difficult to adequately describe optimal behaviour, which can allow the agent to understand the environment. Reinforcement learning is a general problem which can then be solved by a number of methods and any such problem is known as a reinforcement learning problem.

The figure 2.2 shows a typical setting in which RL works. A ***agent*** controller will receive system's ***state*** and a ***reward*** which associated with the last **state transition**. Then it calculates an action that is returned to the system. As a response, the system changes to a ***new state*** and repeats the cycle.

2.3.1 Important elements of Reinforcement Learning

Reinforcement Learning agents could contain one or more of the following elements:

- **Policy:** π, indicate the behaviour of an agent. It is an "state to action" mapping function, such that:

$$\pi : S \rightarrow A \tag{2.10}$$

  where S and A are state space and action space respectively. A policy function can be Deterministic, $A_{t+1} = \pi(s_t)$ or Stochastic $\pi(\alpha|s) = P[\alpha_t = \alpha|s_t = s]$.

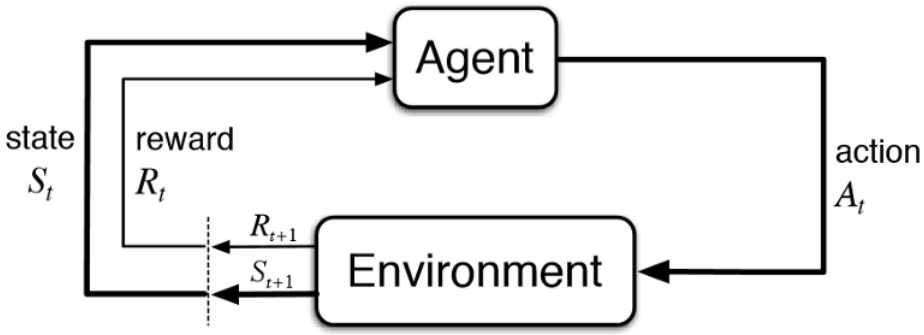


Figure 2.2: Reinforcement Learning Setting (Sutton and Barto, 1998)

- **Return:** If γ is the discount for future rewards, the return G_t (also referred expected total discounted reward) at index t is given by:

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{m=0}^{\infty} \gamma^m r_{t+m+1}, \quad \gamma \in [0, 1] \tag{2.11}$$

2.3.2 Markov Decision Process

A Markov Decision Process (MDP) is a mathematical modelling framework in discreet time for decision making, espesially useful when the results of a process are partly a result of action by the agents and partly randomly. In areas like economics, control, production, and refurbishment education, they have found extensive usage.

Markov property

The **Markov property** is met by A state S_t [4] if and only if:

$$P[S_{t+1}|S_t, S_{t-1}, \ldots, S_1] = P[S_{t+1}|S_t] \tag{2.12}$$

This indicate that the prior condition, S_t, is enough to predict the future, which can thus be neglected for the long-term history.

An MDP can be described as a 5-tuple (S,A,P,R,γ), where:

- $S = \{\mathbf{s_1}, \mathbf{s_2}, \ldots, \mathbf{s_n}\}$ is a **finite** collection of state of the procedure.
- $A = \{\alpha_\mathbf{1}, \alpha_\mathbf{2}, \ldots, \alpha_\mathbf{m}\}$ is a **finite** set of action of the procedure. The set of actions are the feasible options which the agent can proceeds at a given time.
- $P =$ is a transition model of MPD, where $P = \{\mathbf{s}, \alpha, \mathbf{s}^{'}\}$ is the chance of moving in to state $s^{'}$ when taking action a from state s, such that they satisfy the Markov property, as in the definition (Markov property) (2.12). The **Probability matrix P** of the transition of state may be represented:

$$P^{\alpha}_{ss'} = P[S_{t+1} = s^{'} | S_t = s_t, A_t = \alpha] \tag{2.13}$$

- $R =$ is the reward function (real number) of the process. It is Markovian as well and can be the instant or the expected instant reward (for stochastic rewards) at each time step. The **reward function R**,is defined as :

$$R : S \times A \to B, R^{\alpha}_{s} = E[R_{t+1} S_t = s_t, A_t = \alpha] \tag{2.14}$$

  where S, A, B are the state space, the action space and the reward set, respectively.

- $\gamma \in (0,1]$ is a discount factor. When γ=1 a reward maintain its full value independently of the time state. As γ decreases, the effect of reward in the future is declined exponentially by γ^t.

The optimization goal in an MDP is the maximization (or minimization depending on the problem) of the expected total discounted reward.

Optimality

In addition to the expressiveness of MDPs, they can be resolved optimally and become very alluring.

Value Function

Among all the policies **optimal state-value function**, V^{π} is the maximum state-value function:

$$V^{\pi}(s) = \max_{\pi} V^{\pi}(s), \quad \forall s \in S \tag{2.15}$$

The **optimal action value function**, Q^{π} is the maximum action value function over all policies:

$$Q^{\pi}(s,\alpha) = \max_{\pi} Q^{\pi}(s,\alpha), \quad \forall s \in S, \forall \alpha \in A \tag{2.16}$$

Policy

Defining a partial policy order [4]

$$\pi < \pi^{'} \Rightarrow V_{\pi}(s) \leq V_{\pi}^{'}(s), \quad \forall s \in S \tag{2.17}$$

Theorem: An optimal policy π_* exists that is better or equal among all the policies, such that $\pi_* \geq \pi, \forall \pi$

Bellman Equation

The agent attempts from any state in which it is located to get the highest expected reward. To achieve this, we must try to achieve the optimum value function, in other words the maximum sum of cumulative rewards [5].

- The Bellman equation wiil be used to break down, **the value function** in two component, an immediate reward, R_{t+1}, and a discounted successor value γR_{t+1}.

$$\begin{aligned} \mathbf{V(s) = E[G_t | S_t = s]} &= \mathbf{E[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots | S_t = s]} \\ &= \mathbf{E[R_{t+1} + \gamma (R_{t+2} + \gamma R_{t+3} + \ldots | S_t = s]} \\ &= \mathbf{E[R_{t+1} + \gamma G_{t+1} | S_t = s]} \\ &= \mathbf{E[R_{t+1} + \gamma V_{\pi}(S_{t+1}) | S_t = s]} \end{aligned} \tag{2.18}$$

- The Q_{π} **action value** can be decomposed to similar manner such as:

$$Q_{\pi}(s) = E_{\pi}[R_{t+1} + \gamma Q_{\pi}(S_{t+1}, A_{t+1}) | S_t = s, A_t = \alpha] \tag{2.19}$$

The Markov Decision Processes defined by Bellman with the **Bellman Expectation Equations** as shown in Equations 2.16 and 2.17.

2.4 Reinforcement Learning Agents

- **Value-Based Agent**, the agent will evaluate all the states in the state space, and the policy will be kind of implicit, i.e. the value function tells the agent how well each action in a given state is and the agent selects the best.
- **Policy-Based Agent**, instead of representing the value function inside the agent, we explicitly represent the policy. The agent searches for the optimum action value function, so it can act optimally.
- **Actor-Critic Agent**, this agent is a value-based and policy-based agent. It's an agent that stores both of the policy, and how much reward it is getting from each state.
- **Model-Based Agent**, the agent tries to build a model of how the environment works, and then plan to get the best possible behavior.
- **Model-Free Agent**, here the agent does not try to understand the environment, i.e. it does not try to build the dynamics. Instead we go directly to the policy and/or value function. We just see experience and try to figure out a policy of how to behave optimally to get the best possible rewards.

RL agents usually suffer from local maximal because of exploitation. At the early stages of their training, they keep visiting the same states that locally maximize their reward, without exploring the rest of the state space. This is the **exploration-exploitation** trade-off that many policies try to balance [6].

2.4.1 Policy Gradient

The objective of reinforcement learning is to determine the best behavioral strategy for the Agent in order to obtain optimal awards. The **policy gradient** methods target at modeling and optimizing the policy directly. The policy is usually modeled as a parameterized function with respect to $\theta, \pi_\theta(\alpha|s)$. The value of the reward (objective) function depends on this policy and then various algorithms can be applied to optimize θ for the best reward.

The reward function is defined as:

$$J(\theta) = \sum_{s \in S} d^{\pi}(s) V^{\pi}(s) = \sum_{s \in S} d^{\pi}(s) \sum_{\alpha \in A} \pi_{\theta}(\alpha|s) Q^{\pi}(s, \alpha) \tag{2.20}$$

with d^{π} the Markov Chain 's stationary distribution for π_{θ} (on-policy state distribution under π).
It is natural that policy- based methods will be more useful in continuous space, since the values are estimated by an innumerable number of actions and (or) states. For this reason value-based approaches are way too expensive computationally in the continuous space.

Policy Gradient Theorem

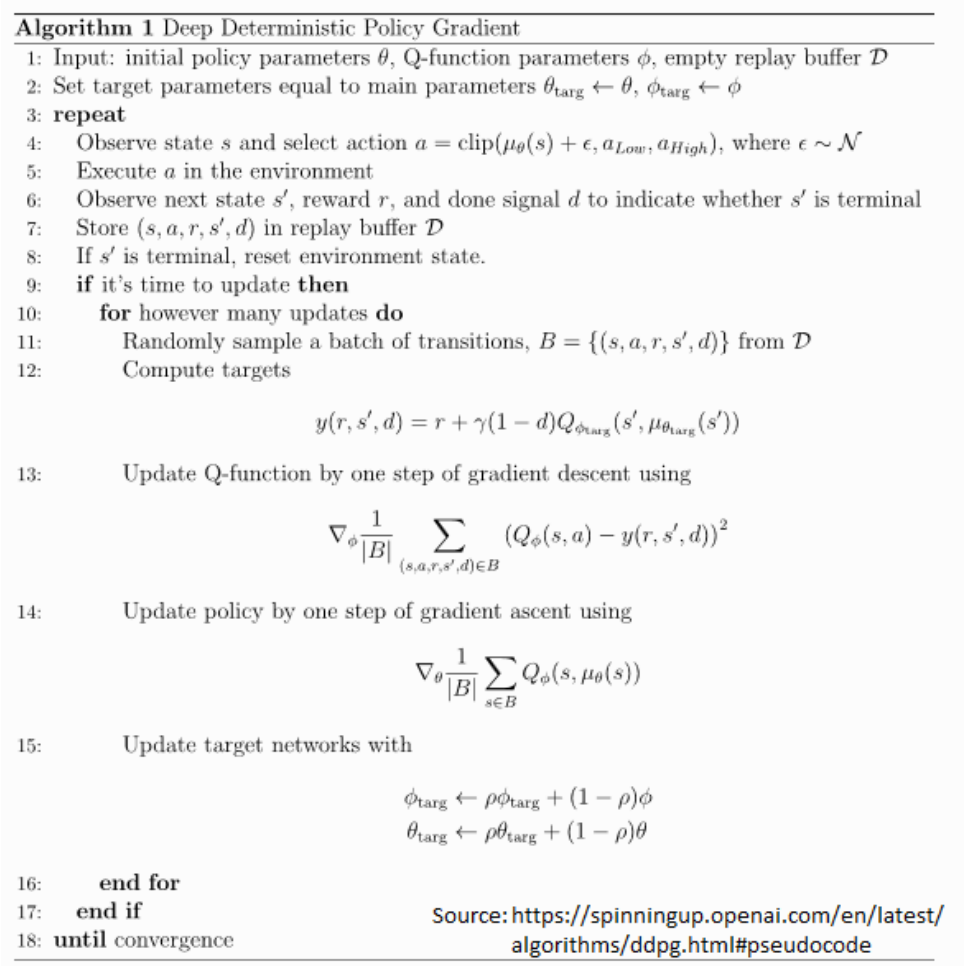
Computing the gradient $\nabla_{\theta} J(\theta)$ is tricky because it depends on both the action selection (directly determined by π_{θ}) and the stationary distribution of states following the target selection behavior (indirectly determined by π_{θ}). Given that the environment is generally unknown, it is difficult to estimate the effect on the state distribution by a policy update.

The policy gradient theorem is a nice way to reform the derivative of the objective function, so that the derivative of d_{π} does not occur and gradient calculations $\nabla_{\theta} J(\theta)$ are simplified a lot (Equation 2.20), and the proof is seen in (Sutton,Barto, 2017; Sec. 13.2) [7]

Deep Deterministic Policy Gradient

Deep Deterministic Policy Gradient (DDPG) [8], is a model-free, off-policy, actor-critic algorithm, combining Deep Policy Gradient (DPG) with Deep Q-Network (DQN). DQN (Deep Q-Network) stabilizes Q-function learning by replaying experience and the frozen target network. The original DQN functions in a discrete space and is expanded into continuous space by actor-critic framework while learning a deterministic policy.

One detail in the paper that is particularly useful in robotics is how to normalize the different physical units of low dimensional features. For example, a model is designed to learn a policy with the robot's positions and velocities as input; these physical statistics are different by nature and even statistics of the same type may vary a lot across multiple robots. Batch normalization is applied to fix it by normalizing every dimension across samples in one mini-batch. Figure 2.3 summarizes the DDPG algorithm.



Algorithm 1 Deep Deterministic Policy Gradient

Input: initial policy parameters θ, Q-function parameters ϕ, empty replay buffer $\mathcal{D}$
Set target parameters equal to main parameters $\theta_{\text{targ}} \leftarrow \theta$, $\phi_{\text{targ}} \leftarrow \phi$
repeat
  Observe state s and select action $a = \text{clip}(\mu_\theta(s) + \epsilon, a_{Low}, a_{High})$, where $\epsilon \sim \mathcal{N}$
  Execute a in the environment
  Observe next state s', reward r, and done signal d to indicate whether s' is terminal
  Store (s, a, r, s', d) in replay buffer $\mathcal{D}$
  If s' is terminal, reset environment state.
  if it's time to update **then**
    for however many updates **do**
      Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$
      Compute targets

$$y(r, s', d) = r + \gamma(1 - d)Q_{\phi_{\text{targ}}}(s', \mu_{\theta_{\text{targ}}}(s'))$$

      Update Q-function by one step of gradient descent using

$$\nabla_\phi \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} \left(Q_\phi(s, a) - y(r, s', d)\right)^2$$

      Update policy by one step of gradient ascent using

$$\nabla_\theta \frac{1}{|B|} \sum_{s \in B} Q_\phi(s, \mu_\theta(s))$$

      Update target networks with

$$\phi_{\text{targ}} \leftarrow \rho\phi_{\text{targ}} + (1 - \rho)\phi$$
$$\theta_{\text{targ}} \leftarrow \rho\theta_{\text{targ}} + (1 - \rho)\theta$$

    end for
  end if
until convergence

Source: https://spinningup.openai.com/en/latest/algorithms/ddpg.html#pseudocode

Figure 2.3: The DDPG Algorithm

2.5 Artificial Neural Networks

Neural networks (NNs) are a loosely biological-brain-modeled computational approach. Brains can be viewed as an interconnected neuron web transmitting complex patterns of electrical signal: input signals are given to dendrites and the output signal is fired via an axon based on those inputs. This effect is imitated by NNs using artificial neurons that can be described mathematically as a graph. NNs receive input to the input neuron from the outside world and proliferate forward- lookingly: the input is transmitted to other neurons in the network and each neuron alters the signal according to its internal

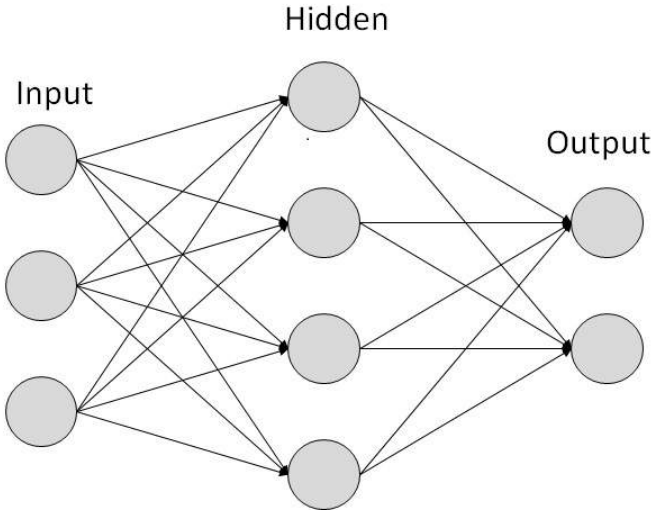


Figure 2.4: Artificial Neural Network

rules until the signal goes through the whole structure and reaches the output from its output neurons to the outside. Such systems "learn", generally without having to be programmed with task specific regulations, to perform tasks through examples.

2.5.1 The Architecture of an Artificial Neural Network

An Artificial Neural Network (ANN), as shown in Figure 2.4 is a set connected neurons in layers:

- **Input layer:** Introduces the initial data for further treatment into the system through subsequent artificial neuron layers.
- **hidden layer:** A layer of weighted inputs from artificial neurons to produce the output via activation function between input layers and output layers.
- **Output layer:** The last neuron layer generating the outputs for the network.

2.5.2 Multi-layer ANN

Due to their hidden layers (for example, Convolutionary Network, Recurrent Neural Network, etc...), multilayer ANNs can solve more complicated classification and regression

tasks.

There are many ways, neural multilayer networks can be set up. They typically have at least one input layer, which sends weighted inputs to a number of hidden layers. These advanced settings are also related to non-linear builds that use sigmoids and other functions to direct artificial neurons firing or activation. Although some of these systems are physically constructed with physical materials, most of them are designed with neural activity software functions.

2.5.3 Activation Functions

The node activation function redefines this node's output given the input or set of inputs on the artificial neural networks.

Sigmoid

A sigmoid feature is a math funtion with a typical "S" or sigmoid curve. The sigmoid function often indicates a special case for the logistic function which generates a set of probability outputs between 0 and 1. In binary classification, sigmoid activation is a common feature.

$$Sigmoid(x) = \frac{1}{1 + e^{(-x)}} = \frac{e^x}{1 + e^x} \tag{2.21}$$

Tan-h

The hyperbolic tangent, or tan-h function, is an alternative to logistic sigmoid. Just like the Sigmoid logistic function, the tan-h function is also a Sigmoid function, but it produces values of [-1,+1] in its range. This means that strongly negative tan-h inputs map negative outputs. Likewise, strongly positive.

$$tanh(x) = \frac{2}{1 + e^{(-2x)}} - 1 = 2 \times Sigmoid(2x) - 1 \tag{2.22}$$

Softmax

The Softmax activation function is used for multi-class classification, as opposed to the Sigmoid activation functions. Softmax function calculates the probabilities distribution

of the event over "n" different events. This function calculates the probabilities of each target class over all possible target classes as a general rule. The calculated probabilities are then used to regulate the target class of the data.

$$Softmax(x) = \frac{e^i}{\sum_i e^{(i)}} \tag{2.23}$$

ReLU

Instead of the sigmoid function, the latest artificial neural networks utilize rectified linear units (ReLUs). When the input is below 0, the linear unit with the rectified output is 0 or the raw input otherwise. This is, if the input is greater than 0, the result equals the input.

$$ReLU(x) = \max(0, x) \tag{2.24}$$

Leaky ReLU

The leaky ReLU function works in a manner that is similar to ReLU, except that in order to avoid the "dying ReLU" problem, the latter is replaced by a small alpha amount rather than the negative data from the inputs.

$$LeakyReLU(x) = \begin{cases} x, & \text{if } x > 0 \\ \alpha x, & otherwise \end{cases} \tag{2.25}$$

In general, non-linearity is not introduced in the network without using an activation function. An activation function allow us to model a response variable, which varies nonlinearly with its explanatory variables (targon variable, class label, or score). Non-linear means that the output can not be reproduced by the linear input combination.
An other way of thinking about this: without a nonlinear activation function on the network, artificial neural networks, regardless of how many layers they have, will behavior exactly as a single layer Neural Network.

Some activation functions could affect the problem of the vanishing gradient (the problem with the disappearing gradient occurs when the gradient becomes so small in prior layers of a deep-neural network that it does little to improve the weights of the earlier layers).

2.5.4 Backpropagation

Backpropagation is a method used to determine the gradient needed in artificial neural networks to calculate network weight. Back-propagation is shorthand for "the backward propagation of error", as an output error is calculated and distributed backward across network layers. It's being used for deep neural network training.

Backpropagation means that the rule of delta can be generalized into multi-layer feedforward networks to calculate gradients for the individual layers iteratively using the chain rule. It is strongly linked with the Gauss-Newton approach and is part of ongoing research on the neural background.

Backpropagation is a more general case of the technique called automatic differentiation. For the learning process, Backpropagation is often used to adjust the mass of neurons with the downward gradient algorithm, in order to calculate the loss function progression.

2.5.5 Gradient Descent Optimization algorithms

In optimization, gradient method is an algorithm to solve problems of the form:

$$\min_{x \in \mathbb{R}^n} f(x) \tag{2.26}$$

Search directions at the current point defined by the function gradient.

Gradient downward is among the most prominent neural network optimization algorithms. Each up-to-date Deep Learning library also includes implementations of multiple gradient-descent optimization algorithms (e.g. lasagne's, caffe's, and keras' documentation).

Gradient descent is a way to minimize an objective function $J(\theta)$ parameterized by a model's parameters $\theta \in \mathbb{R}^d$ by updating the parameters in the opposite direction of the

gradient of the objective function $\nabla_\theta J(\theta)$ w.r.t. to the parameters. The η learning rate determines the steps to reach a (local) minimum. In other words, we walk downwards to reach a valley in the pendulum of the surface created by the objective function.

The gradient descent comes in three variants depending on the number of data used for determining the gradient. Depending on the amount of data, we compromise the correctness of the updated parameter with the time it takes to update the parameter.

Batch gradient descent

Vanilla descent, known as the batch descent, calculates the cost function gradient with respect to the full θ data set parameters:

$$\theta = \theta - \eta \nabla_\theta J(\theta) \tag{2.27}$$

The batch gradient descent is slow and inseparable to memory-free datasets, because the gradient levels for the whole dataset need to be calculated to make just one update. The batch descent does not allow us to upgrade our model online, i.e. with new instances in real time.

Stochastic gradient descent

In contrast, Stochastic gradient descent (SGD) updates each training example $x^{(i)}$ with a parameter, and label $y^{(i)}$:

$$\theta = \theta - \eta \nabla_\theta J(\theta; x^{(i)}; y^{(i)}) \tag{2.28}$$

The batch gradient descent offers superfluous calculations of huge amounts of data because gradients have been recalculated for similar examples before each parameter was updated. By performing one update a time SGD eliminates this redundancy. It is therefore generally much faster and can also be used to learn online.

Mini-batch gradient descent

Finally, the mini-batch descent takes the best of the two worlds and makes an update for every small batch of n training examples:

$$\theta = \theta - \eta \nabla_\theta J(\theta; x^{(i:i+n)}; y^{(i:i+n)}) \tag{2.29}$$

This means that a) reduces the update variance of the parameter to allow greater stability and b) the commonly used advanced deep learning libraries for highly optimized matrix optimizing enable the gradient to be calculated very powerful. Common mini-batch sizes range from 50 to 256, but can differ for various applications. A Mini-batch gradient descent is the preferred algorithm for the neural network training.

Below are some algorithms that the deep-learning community often uses to optimize the task.

Adagrad

Adagrad [9] is a gradient optimization algorithm that simply adjusts the rate according to parameters, with respect to the change, makes smaller updates for parameters related to repeatedly appearing characteristics (i.e. low learning rate) and larger updates for parameters related to rare features (i.e. high learning rates). This is why it is suitable for inadequate data processing. This strategy often improves the performance of convergence compared to the standard stochastic gradient descent when data are sparse and descriptive. It has a base learning rate η, but this is multiplied using the $G_{j,j}$ diagonal elements of the outer matrix of the product.

$$G = \sum_{\tau=1}^{t} g_\tau g_\tau^{\mathsf{T}} \tag{2.30}$$

where $g_\tau = \nabla Q_i(W)$ the gradient, at iteration τ. The diagonal is given by

$$G_{j,j} = \sum_{\tau=1}^{t} g_{\tau,j}^2 \tag{2.31}$$

This vector is updated after every iteration.

One of Adagrad's main advantages is the elimination of the need to manually adjust the learning rate. The default value for most implementations is 0.01 and it is left at that.

The main weakness of Adagrad is the aggregation of the square denominator gradients: since each additional period is positive, the accumulated amount continues to grow while

training is underway. This reduces the learning rate and finally gets very small, so that an additional knowledge can no longer be gained from the algorithm.
Adagrad customizes the general η learning rate for each step θ_i in its update rules, based on past gradients calculated for θ_i:

$$\theta_{t+1,i} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \cdot g_t \tag{2.32}$$

Adadelta

Adadelta [10] is an Adagrad extension to lower its aggressive monotonous rate of learning. Adadelta limits the winder of the accumulated past gradients to certain fixed size w rather than all past square gradients. The sum of gradients is defined repetitively as the decreasing mean for all gradients, which does not save the previous gradients inefficiently. The running average $E[g2]t$ then only rely upon the previous average and current gradient (as the fraction γ):

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2 \tag{2.33}$$

The Adagrad update parameter, which we previously derived therefore takes shape:

$$\Delta\theta_t = -\frac{\eta}{\sqrt{G_t + \epsilon}} \times g_t \tag{2.34}$$

RMSprop

In Lecture 6e of his Coursera Class [11], Geoff Hinton proposed RMSprop an unpublished, adaptive learning rate method.

RMSprop and Adadelta were both independently refined while the need arises to deal with the radically reduced learning rates of Adagrad. RMSprop is actually equal to the first vector of Adadelta update:

$$E[g^2]_t = 0.9 E[g^2]_{t-1} + 0.1 g_t^2 \tag{2.35}$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} \cdot g_t \tag{2.36}$$

RMSprop also divides the learning rate by an average of squares that declines exponentially. Hinton recommends γ to be defined as 0.9, and a excellent learning rate default value η is 0.001.

Adam

Adaptive Moment Estimation (Adam) [12] is another way for each parameter to calculate adaptive learning rates. Adam maintains a decaying average v_t of the past squared paths, as well as an exponentially decaying average m_t of previous paths. The decaying averages of past m_t and past squared gradients v_t have been calculated in the following manner:

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1-\beta_1) g_t \\ v_t &= \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \end{aligned} \tag{2.37}$$

m_t and v_t are the first (average) and second (uncentered) moment of gradient, therefore the name of the method. As m_t and v_t are initialized as vectors of 0's, Adam's creators note that they are inclined to zero, particularly in the primaty steps and in particular when the decadence rates are meager (i.e. β_1 and β_2 are close to 1).
By calculating first and second instant biases, they counteract these biases

$$\begin{aligned} \hat{m}_t &= \frac{m_t}{1-(\beta_1)^t} \\ \hat{v}_t &= \frac{v_t}{1-(\beta_2)^t} \end{aligned} \tag{2.38}$$

These parameters are then used as we saw in Adadelta and RMSprop, which gives the Adam update rule:

$$\theta_{(t+1)} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \cdot \hat{m}_t \tag{2.39}$$

Default values of 0.9 are proposed by the authors for β_1, 0.999 for β_2, and 10^{-8} for ϵ. Empirically, they show that Adam works well and compares favorably well to other adaptive learning methods.

Chapter 3

Problem Statement

3.1 Portofolio Optimization

A number of the benefits of this artificial intelligence field lead to the development of Reinforcement learning (RL) on financial markets. RL, in particular, enables the "predictions" and "construction of portfolios" to be combined in a single integrated phase to closely align machine learning problem with investor goals. In the meantime, consideration can be given to major constraints, such as transaction costs, the market cash and the risk aversion for investors. The RL community of research has made significant progress in the field of finance [13].

Especially in portfolio management the whole investment is spread across multiple assets and there are many combinations of portfolio vectors definitions (Equations 2.1 and 2.8). The objective of **portfolio optimization** is to efficiently resolve the allocation problem, where an objective function is formulated and optimized in relation to a portfolio vector that refers to investor preferences. We can see examples of portfolio combination in Figure 3.1.

Different metrics are introduced in subsection 3.1.1 , which combine in order to formulate objective and utility function that are then used to assemble the **Markowitz Model**.

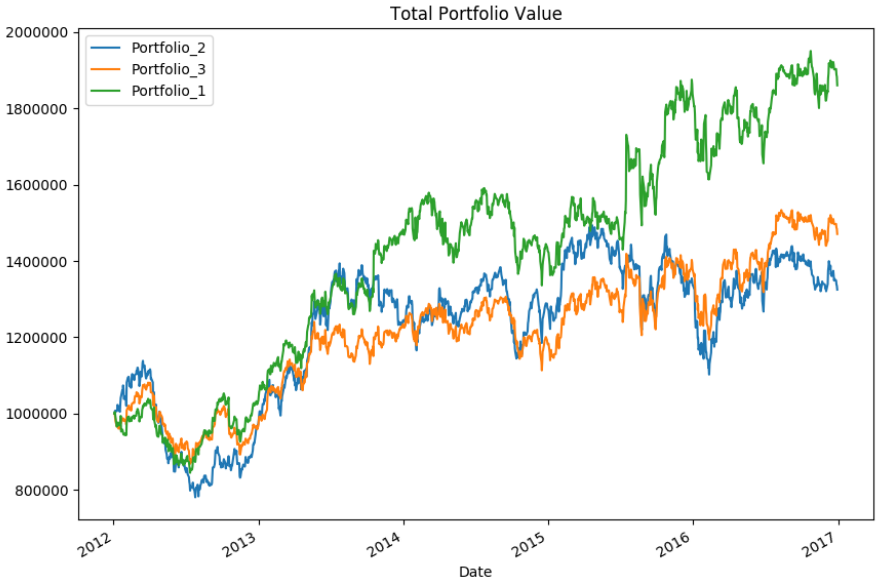


Figure 3.1: Different portfolio weight combination

3.1.1 Modern Portfolio Theory

The theory of Modern Portfolio Theory (MPT) is how risk investors can construct portfolios, in order to optimize or maximize expected returns on the basis of market risk levels, with the concern that risk constitutes an increasing reward element. The theory reveals that an "efficient boundary" of optimal portfolios can be built which offer the maximum expected return in a certain risk level. This theory was invent by **Harry Markowitz** [14].

The modern theory of portfolio argues that risk and income properties of an investment should not be examined separate, but that the impact of the investment on risk and returns across the portfolio should be assessed. MPT has shown that an investor can develop a multiple asset portfolio that maximizes returns at a fix risk level. Likewise, with the desired level of expected profit, a investor can build a portfolio with the lowest potential risk. The return on individual investments is less crucial than the investment's attitude to the unified portfolio based in statistical measures as the variance

and correlation.

Portfolio Risk and Expected Return

MPT assumes that investors are not risky and therefore prefer a less risky portfolio to an more risky portfolio at a certain level of return. This means that only when he or she expects more reward will an investor take more risk.

At $t+1$ time index the expected return of a property is:

$$E[R_{t+1}|R_t, R_{t-1}, \ldots, R_0] \longrightarrow E[R_{t+1}] = \mu_{t+1} \tag{3.1}$$

The anticipated portfolio returns is calculated as a weighted quantity of the respective assets returns defined as:

$$\begin{aligned} E[R_{t+1}^{(p)}] = \mu_{t+1}^{(p)} &= E[w_1 R_{t+1}^{(1)} + w_2 R_{t+1}^{(2)} + \cdots + w_M R_{t+1}^{(M)}] \\ &= w_1 \mu_{t+1}^{(1)} + w_2 \mu_{t+1}^{(2)} + \cdots + w_M \mu_{t+1}^{(M)} \\ &= w^T [\mu_{t+1}^{(1)} \quad \mu_{t+1}^{(2)} \quad \cdots \quad \mu_{t+1}^{(M)}] \\ &= w^T \mu_{t+1} \end{aligned} \tag{3.2}$$

The portfolio risk is a complex function of the differences between each asset and its correlations. To calculate the risk of a four-asset portfolio, an investor needs four asset variances and six correlating values each because six possible combinations are available with four assets. The total portfolio risks or standard deviations are lower than the weighted amount calculated on the basis of the matching assets. The risk of an asset is the variance $\sigma_{t+1}^2 = Var[R_{t+1}]$ called **volatility** in finance.

The **portfolio variance** involves co-variance terms between the assets such that

$$\begin{aligned} Var[R_{t+1}^{(p)}] = (\sigma_{t+1}^{(p)})^2 &= Var[w_1 R_{t+1}^{(1)} + w_2 R_{t+1}^{(2)} + \cdots + w_M R_{t+1}^{(M)}] \\ &= w_1^2 Var[R_{t+1}^{(1)}] + w_2^2 Var[R_{t+1}^{(2)}] + \cdots + w_M^2 Var[R_{t+1}^{(M)}] \\ &\quad + 2w_1 w_2 Cov[R_{t+1}^{(1)}, R_{t+1}^{(2)}] + 2w_1 w_3 Cov[R_{t+1}^{(1)}, R_{t+1}^{(3)}] \\ &\quad + \ldots \\ &\quad + 2w_M w_{M-2} Cov[R_{t+1}^{(M)}, R_{t+1}^{(M-2)}] \\ &\quad + 2w_M w_{M-1} Cov[R_{t+1}^{(M)}, R_{t+1}^{(M-1)}] \\ &= \sum_{i=1}^{M} \sum_{j=1}^{M} w_i \sum\nolimits_{t+1} w_j \\ &= w^T \sum\nolimits_{t+1} w \end{aligned} \tag{3.3}$$

where $\Sigma_{t+1} \in R^{M \times M}$ is the covariance matrix of returns at the time t for the returns of the M assets.

Co-variances are calculated with the unbiased estimator (Wilmott, 2007) [15]

$$Cov[R_t^{(i)}, R_t^{(j)}] = \frac{1}{T-1} \sum_{\tau=1}^{T} (R_\tau^{(i)} - \bar{R}^{(i)})(R_\tau^{(j)} - \bar{R}^{(j)}) \tag{3.4}$$

Variances are also calculated with the correction formula of the Bessel [16]

$$Var[R_t] = \frac{1}{T-1} \sum_{\tau=1}^{T} (R_\tau - \bar{R})^2 \tag{3.5}$$

Single Asset vs Portfolio

Since portfolio are equivalent to complex assets, the reason that is preferable to simple asset is that portfolio minimize risk by allowing diversification [17]

- **Full investment in one asset**

  Invest all the budget on a single asset, we choose asset i=1 then

  - *Portfolio Vector*: $w = [1 \quad 0 \quad \dots \quad 0]^T \in R^M$
  - *Expected Portfolio Return*:
    $E[R_{t+1}^{(p)}] = \mu_{t+1}^{(p)} = w^T \mu_{t+1} = [1 \quad 0 \quad \dots \quad 0][\mu \quad \mu \quad \dots \quad \mu]^T = \mu$
  - *Portfolio Variance*:

$$\begin{aligned} Var[R_{t+1}^{(p)}] = (\sigma_{t+1}^{(p)})^2 &= w^T \sum\nolimits_{t+1} w \\ &= [1 \quad 0 \quad \dots \quad 0] \begin{bmatrix} \sigma^2 & 0 & \dots & 0 \\ 0 & \sigma^2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \sigma^2 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \sigma^2 \end{aligned} \tag{3.6}$$

- **Equal-weight portfolio**

  Share the budget uniformly to the M assets:

  - *Portfolio Vector*: $w = [\frac{1}{M} \quad \frac{1}{M} \quad \dots \quad \frac{1}{M}]^T \in R^M$
  - *Expected Portfolio Return*:
    $E[R_{t+1}^{(p)}] = \mu_{t+1}^{(p)} = w^T \mu_{t+1} = [\frac{1}{M} \quad \frac{1}{M} \quad \dots \quad \frac{1}{M}][\mu \quad \mu \quad \dots \quad \mu]^T = \frac{1}{M} M \mu = \mu$

– *Portfolio Variance*:

$$
\begin{aligned}
Var[R_{t+1}^{(p)}] = (\sigma_{t+1}^{(p)})^2 &= w^T \sum\nolimits_{t+1} w \\
&= [\frac{1}{M} \quad \frac{1}{M} \quad \dots \quad \frac{1}{M}] \begin{bmatrix} \sigma^2 & 0 & \dots & 0 \\ 0 & \sigma^2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \sigma^2 \end{bmatrix} \begin{bmatrix} \frac{1}{M} \\ \frac{1}{M} \\ \vdots \\ \frac{1}{M} \end{bmatrix} \\
&= \frac{1}{M^2} M\sigma^2 = \frac{\sigma^2}{M}
\end{aligned}
\tag{3.7}
$$

The above two options have the same expected return $\mu_{t+1}^{(p)}$, but the variance of the second option is shrunk by a factor of M. By this we can tell why "Uniform Portfolio" is selected by the investors. This is an example where a portfolio with the same expected return with a single asset is less risky and less volatile than a asset.

3.1.2 Metrics Ratios

- **Sharpe Ratio**

  The Nobel Prize winner William F. Sharpe [18] created the Sharpe ratio to help investors compare the return of an investment with its risk:

  $$SR = \frac{E[R_\alpha - R_b]}{\sqrt{var[R_\alpha - R_b]}} \tag{3.8}$$

  where R_α is the asset return, R_b is the risk free rate. $E[R_\alpha - R_b]$ is the expected value of the excess of the asset return over the benchmark return, and σ_α is the standard deviation of the asset excess return. One idea in this calculation is that a portfolio invests in "zero risk", for example buying the U.S. Treasury bills (for which the expected return is the risk-free rate) have the Sharpe ratio of exactly zero. In general, the higher the Sharpe ratio, the better the risk-adjusted return.

  The Sharpe Ratio can be considered as the **Signal-to-Noise Ratio** (SNR) analog for finance [19].

- **Sortino Ratio**

  By dividing excess return by downside deviation, the Sortino proportion enhances

the Sharpe rate by isolating downside volatility from total volatility. The Sortino ratio differs from the total volatile by the asset standard difference in negative returns on assets, known as the downside deviation, in that it differentiates harmful volatility from the overall volatility. The Sortino ratio takes the return of the asset and reduces the risk-free rate and divides it by the deviation of the asset. The ratio was named after Frank A. Sortino [20].

Like the Sharpe ratio, a higher Sortino ratio is better. If two similar investments were considered, the most high Sortino investor would be the rational investor, because this means that the investment generates more profit per unit of bad risk. The Sortino ratio formula is the following:

$$Sortino_{Ratio} = \frac{E[R_\alpha - R_b] - R_f}{\sigma_d} \tag{3.9}$$

where R_f is the Risk Free-Rate of Return and σ_d is the Standard Deviation of Negative Asset Returns.

- **Omega ratio**

  The Omega ratio is a relative measure of the likelihood of achieving a given return, such as a minimum acceptable return (MAR) or a target return. It was devised by Keating and Shadwick in 2002 and is defined as the probability weighted ratio of gains versus losses for some threshold return target [21]. The higher the omega value, the greater the probability that a given return will be met or exceeded. Omega represents a ratio of the cumulative probability of an investment's outcome above an investor defined return level (a threshold level), to the cumulative probability of an investment's outcome below an investor's threshold level. The omega concept divides expected returns into two parts, gains and losses, or returns above the expected rate (the upside) and those below it (the downside). Therefore, in simple terms, consider omega as the ratio of upside returns (good) relative to downside returns (bad).

  The ratio is calculated as:

$$\Omega(r) = \frac{\int_r^{\inf}(1 - F(x))dx}{\int_{-\inf}^r F(x)dx} \tag{3.10}$$

where $F(x)$ is the cumulative distribution function of the returns and r is the target return threshold which defines a value compared to a loss.

The commonly used Sharpe ratio can be contrasted to Omega by taking into account the return-to-volatility ratio. The Sharpe ratio only takes into account the first two times of the return, while the Omega ratio takes into account all the times by construction.

- **Maximum Drawdown (MDD)**
  A maximum drawdown (MDD) is the maximum loss from a peak to a trough of a portfolio, before a new peak is attained. Maximum drawdown (MDD) [22] is a measure of the relative risk of one inventory assessment strategy versus another, given the focus on capital preservation, a concern of the vast majority of investments. For instance, the average output of two evaluation strategies, tracking error and volatility, can vary greatly in their drawdown compared with the benchmark. However, it is important to note, without considering the frequency of large losses, that it only measures the size of the biggest loss. Because it only measures the biggest downturn, MDD does not indicate how long an investor has taken to recover from the loss or even recovered from the investment at all. It can be used as an independent measure or as an input into other measurements such as "Calmar Ratio" and the maximum drawdown rate. Maximum drawdown in percentages is expressed and computed as:

$$MDD = \frac{ThroughValue - PeakValue}{PeakValue} \tag{3.11}$$

3.1.3 Markowitz Model

A portfolio allocation problem that is finding a portfolio vector w, as stated in **Assumptions Of MPT** on the portfolio vector definition 2.1 in a universe of M assets, is formulated mathematically by (Markowitz, Kroll et al. [14]). The Markowitz model therefore provides the optimal w_* portfolio, which minimizes volatility for a given level of return, in such a way:

$$\sum_{i=1}^{M} w_{*,i} = 1, \quad w_* \in R^M \tag{3.12}$$

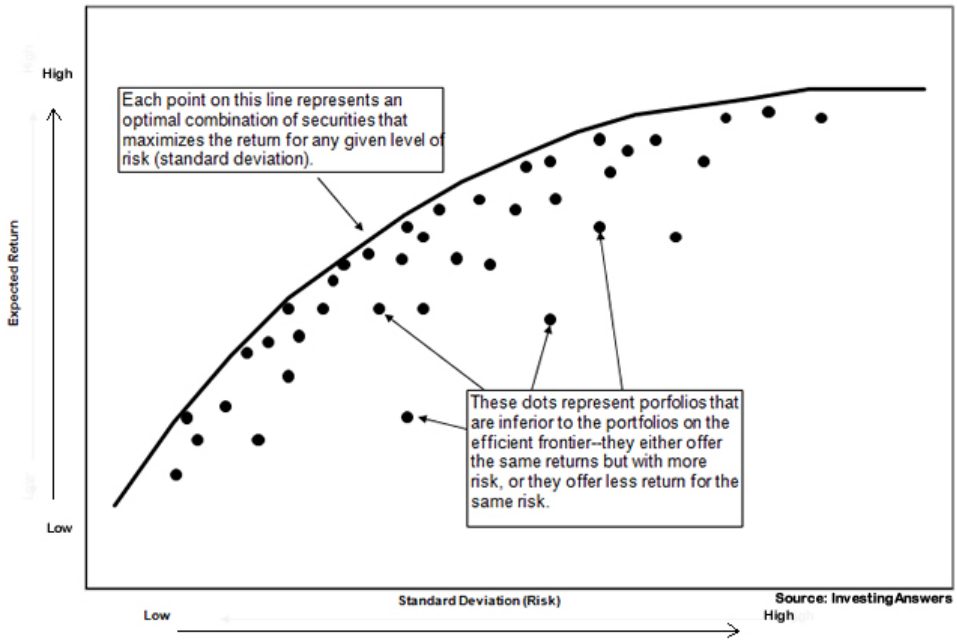


Figure 3.2: Efficient Frontier

The criteria that investors satisfy (Assumptions Of MPT):

- Investors are rational and avoid risks whenever possible
- Investors aim for the maximum returns for their investment
- All investors share the aim of maximizing their expected returns

The fundamental principle of this theory is the possibility for investors to compose an "efficient set of portfolios—Efficient Frontier" provides maximum expected returns for a specific level of risk. An investor's tolerance for risk determines the type of "efficient portfolio" he opts for. An investor with the lowest tolerance opts for a portfolio that offers him the maximum expected return given the lowest possible risk and vice versa. The Figure 3.2 gives an overview of the concept of Efficient Frontier.

As covered in Section 2, different combinations of assets produce different expected returns. One of the most important realization after Prof. Markowitz proved an efficient

set of portfolios was the power of diversification. By simply constructing portfolios with different combinations of assets, investors could achieve a maximum expected return given their risk preferences due to the fact that the returns of a portfolio are greatly affected by nature of the relationship between assets and their weights in the portfolio.

3.1.4 Transaction Costs

Transaction costs are the key feature when selecting a real portfolio as they lower net returns and reduce future investments available capital. Costs may be considered separately in a restricted way, imposing a predefined amount not to be exceeded or deducted from the expected portfolio return and/or capital reduction for the real investment. When defining a complete portfolio-optimization model, some of the limitations on transaction cost definition can be relaxed without affecting the accuracy of the model, as the optimization "drive" transaction costs up to the minimum limit value [23].

To simplify the transaction cost analysis we will be charging 0.25% for every activity (Quantopian, 2017). Thus, when a new portfolio vector (portfolio re-balancing) is determined, it is necessary to reduce from the budget the corresponding transaction costs. To demonstrate this, assume Tom buy two shares of ABC stock from his dealer, Nikos. He pays $200 for the shares at $100 per share. Nikos originally take the shares for a total of $200, incurring a $0.50 transactions charged to Tom. If Tom chooses to sell both stocks, the stock price increases at $110 then the transaction cost is $0.55.

Multi-Stage Decision Problem

The participation of transactions cost is a multistage decision problem for portfolio management that in simple words makes two series of states with the same beginning and end states have diverse values.
We know that w_* portfolio vector affect w on optimum allocation. In a sequential optimization of the portfolio the past w_t decisions will have a candid impact on optimal future decisions w_{t+1} apart from maximizing immediate rewards, the negative consequence for future decisions should be eliminated as well. The use, even when this involves acting suboptimally in the near future from a traditional optimization viewpoint, of reinforcement learning agents that aim at maximizing long-term rewards could be beneficial.

3.2 Optimization Setup for Reinforcement Learning

3.2.1 Action Space

To solve the asset allocation problem, the trading agent must at every stage t be able to regulate the Portfolio Vector w_t, accordingly the action α_t at the time t is the portfolio vector w_{t+1} at the time $t+1$:

$$\alpha_t \equiv w_{t+1} \overset{(2.1)}{=} [w_{1,t+1}, \quad w_{2,t+1}, \quad \ldots, \quad w_{M,t+1}] \tag{3.13}$$

The Action Space $\mathbb{A}$ is thus a subset of the continuous R^M real M-dimensional space.

$$\alpha_t \in \mathbb{A} \subseteq R^M, \quad \forall t \geq 0 \quad and \quad \sum_{i=1}^{M} \alpha_{i,t} = 1 \tag{3.14}$$

The action room is continuous (infinite) and therefore consider the stock market as an infinite decision-making process for Markov (IMDP).

3.2.2 Observation Space

We can only monitor asset prices at any time step t, so that price vector p_t is the o_t observation as described:

$$o_t \equiv p_t \overset{(2.2)}{=} [p_{1,t}, \quad p_{2,t}, \quad \ldots, \quad p_{M,t}] \tag{3.15}$$

The M-Dimensional positive real-space subsets $\mathbb{R}^M$ (prices are non-negative real-world values) are therefore the observer space $\mathbb{O}$:

$$o_t \in \mathbb{O} \subseteq \mathbb{R}^M, \quad \forall t \geq 0 \tag{3.16}$$

3.2.3 State Space

We process the o_t observations, and obtain s_t to help the state agent training. In addition, as a previous time step vector portfolio w_{t-1} affect transaction costs we are also adding

w_t to the agent state or equitably α_{t-1} to the agent's state obtain the 2-tuple:

$$s_t = \langle w_t, \rho_{t-T:t} \rangle = \left\langle \begin{bmatrix} w_{1,t} \\ w_{1,t} \\ \vdots \\ w_{1,t} \end{bmatrix}, \begin{bmatrix} \rho_{1,t-T} & \rho_{1,t-T+1} & \cdots & \rho_{1,t} \\ \rho_{2,t-T} & \rho_{2,t-T+1} & \cdots & \rho_{2,t} \\ \vdots & \vdots & \ddots & \vdots \\ \rho_{M,t-T} & \rho_{M,t-T+1} & \cdots & \rho_{M,t} \end{bmatrix} \right\rangle \tag{3.17}$$

with $\rho_{i,(t-\tau)\to t}$ is the cumulative asset returns i for the time interval $[t-\tau, t]$.

Therefore, the $\mathbb{S}$ state space is a sub-set of the $\mathbb{R}^K$ permanent K dimension of real space, where K is hidden in the state manager of the CNN:

$$s_t^{\alpha} \in \mathbb{S} \subseteq \mathbb{R}^K, \quad \forall t \geq 0 \tag{3.18}$$

General, Infinite Partially Observable Markov Decision Process (IPOMDP) should model the financial market (portfolio optimization), since:

- The space of action is continuous (infinite), $\mathbb{A} \subseteq \mathbb{R}^M$
- The o_t observation is not adequate (partially observable) environmental statistics
- State space (infinite) is continuous, $\mathbb{S} \subseteq \mathbb{R}^K$

3.2.4 Reward Signal

In order to compose a Reinforcement learning setup, the resolution of the reward signal is usually the daring step. The reward is a scalar value that fully sets out the agent's goals, and the maximization of the intended cumulative award is the perfect solution to this task. The field of Inverse Reinforcement Learning is the optimum reward generating function [24] and [25].

The aim of the reinforcement learning techniques is to maximize the *expected cumulative reward signal* by default, so the problem of optimization, which the agent solves (parameterized by θ) is:

$$\max_{\theta} \quad \sum_{t=1}^{T} \mathbb{E}[\gamma^t r_t] \tag{3.19}$$

3.3 Related Work

In the sections below we describe several developments in the optimization of machine learning portfolio strategies to gain insight into the current work.

3.3.1 Support Vector Machines

A study conducted by Shunrong Shen, Haomiao Jiang and Tongda Zhang explored different algorithms of stock market forecasting, especially those that used support vector machine (SVM) [26]. The project provides a new algorithm that predicts the future stock trend by means of SVM, based on the temporal correlation between global stocks and different financial products. While we will focus more on techniques of reinforcement learning, their work demonstrates the variety of machine learning methods currently being tested on the stock market. Their findings focused more on predictability than profitability, while reinforcement learning methods were designed to optimize an objective value function based on some performance metrics.

3.3.2 Recurrent Reinforcement Learning

- In this approach the decision-making of investment developed by J. Moody and M. Saffell [27] is considered as a stochastic problem and strategies are directly identified. They have an adaptive algorithm for discovering investment policies, called **Recurrent Reinforcement Learning (RRL)**. Dynamic programming and enhancement algorithms like TD-learning and Q-learning are different from direct enforcement approaches, which try to estimate a value function for the control problem. This facilitates the representation of the problem through the RRL Direct Reinforcement Framework and prevents Bellman's dimensionality and offers convincing efficiency benefits. They demonstrate how direct reinforcement can be used to optimize risk-adjusted returns on investment, taking account costs. They use real financial information intra-daily and find that their RRL-based approach produces better trade strategies than Q-learning systems.

- Steve Y. Yang and Saud Almahdi [28] are also taking another approach to solving optimal asset allocation problems and a number of trading decision schemes based on methods of enhanced learning. They establish an optimum allocation of variable weights in line with a consistent downside risk measure E(MDD). The Calmar Ratio, specifies their method using the RRL method for both buying and selling signals and asset allocation weights, with a consistent risk-adjusted performance goal. The expected maximum risk downward-focused objective function is shown through the most frequently traded exchange funds portfolio as a higher return than previously proposed RRL functions (i.e. Sharpe or Sterling Ratio), and variable weight portfolios in various scenarios of transactions cost equal portfolios .
  NOTE: The Calmar ratio represents a comparison between the average annual compound rate of return and the maximum risk attraction for commodity trading consultants and hedge funds. The smallest the Calmar ratio, the worse the investment was carried over the specified period on a risk-based basis, the higher the Calmar ratio, the better it was.

- Deep learning (DL) combined with reinforcement learning in the work of Deng et al. [29], introduced a recurrent deep neural network (NN) for real-time financial signal representation and trading. DL automatically detects the dynamic market conditions for informative learning, and the RL module then interacts with deep representations and decides to accumulate ultimate income in an unknown environment. The system of learning is performed in a complex NN with a highly recurring structure. They thus propose a time-based task-back-cutting to tackle the problem of deep training slowdown. The strength of the neural system is confirmed on both the stock and commodity markets under wide-ranging test conditions.

The RRL approach is clearly different from dynamic programs and strengthening algorithms, such as TD-learning and Q-learning, which try to approximate calculate a value function for the control problem. With the RRL framework, simple, elegant problem representation is created, the dimensionality of Bellman is avoided and efficiency offers compelling advantages: Compared to Q-learning when exposed to rowdy data sets, RRL has a more stable performance. Q-learning algorithm is more sensitive to selecting the

value (maybe) because of the recursive dynamic optimization property whereas RRL algorithms can choose the objective function and save time.

3.3.3 Q-learning

- Reinforcement learning for trading systems and portfolios developed by Moody [30] proposed training systems through reinforcement learning by optimizing financial objective functions. The functions used for value are profit or wealth, the Sharpe ratio and the proposed online learning differential Sharpe ratio. They tested Q-learning and Real-Time Reinforcement Learning (Max Immediate Reward). Their reinforcement trader achieves a simulated out-of-sample profit of over 4000% for a 25 year period 1970 through 1994, compared to the return for a buy and hold strategy of about 1300%. This superior result is achieved with substantially lower risk. One more pros of this work is that they used softmax output layer (Multiple Assets).
  Note: The Sharp derivative ratio is economically similar to the marginal utility in that it is eager to bear the risk for one unit of sharp increase which has been described as being:
  $$\frac{dS}{d\tau} = \lim_{\tau \to 0} \frac{S[t+\tau] - S[t]}{\tau}$$
  Rather than finding a symbolic solution to the derivative, the gradient would be measured across multiple time steps.

- Optimal Asset Allocation Using Adaptive Dynamic Programming in the work of Ralph Neuneier [31] proposed Q-learning via Neural Network for Discrete Action Space (binary selection). He tested it in German Stock Index (DAX). The neural networks are used as an approach to value that leads to a strategy of allocation of assets that exceeds heuristic standards.

3.3.4 Policy Gradient Algorithms

- Automated Trading Systems (ATS) effect on financial markets is increasing annually and algorithms now account for the multiplicity of orders placed on the stock exchanges, developed by Necchi [32] proposed a policy gradient agent with

Parametric-Based Exploration into the scope of algorithmic Trading ("single asset"). He establish a MDP with Historic Prices as state space, Portfolio Vector as action space with daily log returns as reward function. He run successful back-tests on synthetic data and prove the significance of transaction costs.

- Policy Gradient combined with Neural Network is the proposed agent is the work of Jiang et al. [33]. They enable CNN and LSTM policy Gradient Agents into the scope of multiple assets (softmax output layer). Neural network has the jobs to investigate an asset's history and assess the probable growth in the near future. For each asset the evaluation rate is reduced by its deliberate weight change and the consequent soft-max layer is the new weight of the portfolio for the next trading period. The RL framework's reward function is the exact average periodic logarithmic returns. They use a portfolio Vector memory.

Chapter 4

Our Approach

The lack of a stable theoretical framework for financial trading is a challenge for developing a training system. Although not to the same degree, neural networks still being a developing field, suffers from similar problems. LeCun et al. [34] stated on the subject of backpropagation neural network:

Backpropagation is popular because it is conceptually simple, computer-friendly and often worked. It may, though, look more like art than science if it works well and sometimes works. Many apparently erratic decisions, including numbers and types of neurons, layering, learning rates and training and tests are necessary to design and train a network using backpropagation.

Although the benefit of the use of neural networks in prediction is that, after their learning is completed, the agents can learn from the examples, even if there is a significant noise in the training set, hidden and strongly not-linear dependencies. Those decisions can be crucial, however, because the decisions are broadly problematic and data reliant, there are no stupid recipe to make.

A trading algorithm strategy based in Neural Network is proposed, which presents an agent with the current market situation, enables him to analyze the situation from his own experience and then to decide on what should be the best investment action for them. We used computational tools that are mainly problem and data-hooked in a field

with few theoretical foundations to guide us. That means that during the design process we had to gain theoretical insights with empirical test. Since market data for such tests can not be simulated or generated, we must take data from historical source, which will then evaluate the performance of the system which can point out data cleaning issues.

4.1 Overview

To achieve this, we intend to apply the earlier defined reinforcement learning framework, because we understand that this issue can be resolved by RL methods. We see the area of state with the necessary market and current investment information, space of action as an opportunity for potential investment in a given period and the role of rewarding the profits of the agent. We will use techniques based upon Deep Policy Network learning methods and will evaluate their success in solving our problem.

The problem is the one of automated portfolio management: given a set of stocks, how to best allocate money through time to maximize returns at the end of a fixed number of timesteps. This enables us to create an automatic agent that best allocates the weight of its investment between various inventories.

We experiment to the stock market, using the framework on daily data with a daily rebalance.The thesis is decomposed in 4 parts:

- Data Preprocessing
- Environment Set-up
- Deep policy network design
- Training and Testing of the agent

4.2 Data Treatments

The trading tests are carried out on S&P 500 data, where there are about 500 tradable stocks. However, for the reasons given by Evans and Archer [35] portfolio sizes beyond 10 stocks do not make much sense. This section also describes a pre-process normalization of the data arrangement the neural networks take to be their input.

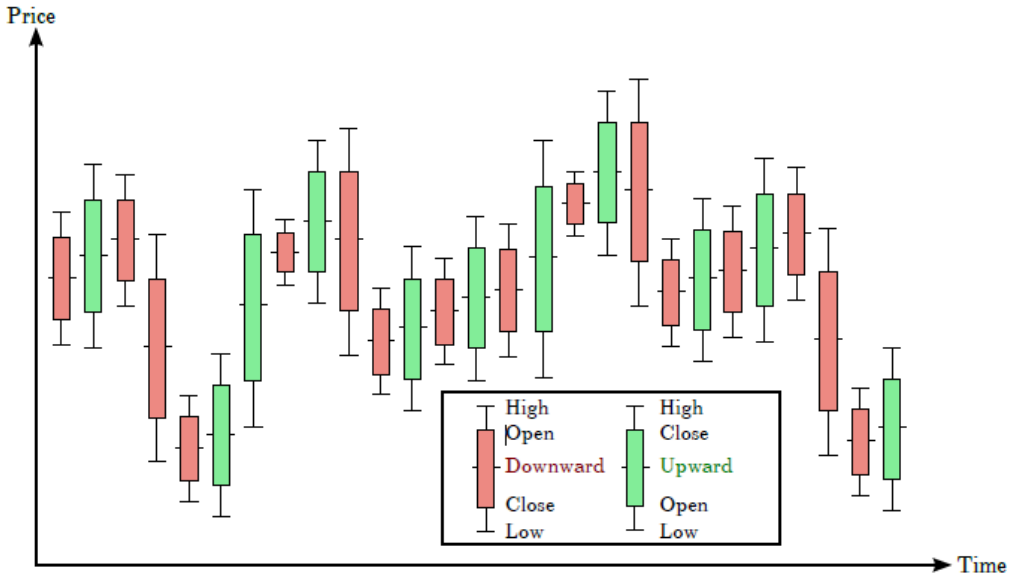


Figure 4.1: Candlestick Chart for (OHLC = Open High Low Close) prices

4.2.1 Candlesticks

A vector $d \in \mathbb{R}^4$ can be described as a candlestick that serve as the movement over time of a financial asset. We are going to use the following representation for arranging the 4 prices in a candle; $d = (close, low, high, open)^T$. Thus we can see our data set as a series of vectors in $\mathbb{R}^4$ in Figure 4.1. The candle has a small real body (distance between open and close) which indicates whether a stock's closing price was higher or lower than its opening price. If the body is red the stock closed lower, otherwise if the body is green it closed higher.

4.2.2 Data Preprocessing

Ten large companies are chosen in random for the portfolio in the experiments of this thesis. The portfolio size, $stocks + 1$, is 11 along with the cash. This statistic is selected from experience regarding portfolio diversification. For high volume markets, m may be as great as the entire number of assets available, such as the foreign exchange market. We follow the below two hypotheses:

- **Hypothesis of zero slippage:** All market assets are liquid enough to make every

trading at the last price immediately possible when an order has been placed.

- **Impact of zero market :** The investments by the Software Trading Agent are so small that they have no effect on the market.

Thus, we select the highest volume assets for improved liquidity and a better environment.

The entry is a raw time series of prices for each stock (High, Low, Open, Close). The output is a matrix of 4 rows and n = (number of available data points) columns. The columns contain the following:

- $\frac{Close(t-1)}{Open(t-1)}$
- $\frac{High(t-1)}{Open(t-1)}$
- $\frac{Low(t-1)}{Open(t-1)}$
- $\frac{Open(t)}{Open(t-1)}$

Price Vector

The historical price data for the output of a portfolio vector are fed to the neural network. At the end of the t period, the input to the neural network is X_t, rank 3, in (f, n, m) form, when the amount of inventory that we would like to study is m, the input periods before t are n and the number of features is $f = 4$. Since prices are far less relevant than the previous ones in the past, $n = 35$ is used to experiments. The m asset criterion was provided in Section 4.2. Since the portfolio performance is based solely on price changes, all prices in the input tensor represented below.

$$X_t = \begin{bmatrix} (\frac{Close(t-n-1)}{Open(t-n-1)}, & \cdots & , \frac{Close(t-1)}{Open(t-1)}), \\ (\frac{High(t-n-1)}{Open(t-n-1)}, & \cdots & , \frac{High(t-1)}{Open(t-1)}), \\ (\frac{Low(t-n-1)}{Open(t-n-1)}, & \cdots & , \frac{Low(t-1)}{Open(t-1)}), \\ (\frac{Open(t-n)}{Open(t-n-1)}, & \cdots & , \frac{Open(t)}{Open(t-1)}) \end{bmatrix} \tag{4.1}$$

We don't need to normalize the data since it's already of ratio of 2 prices closed to one.

4.3 Proposed Reinforcement Learning Model

In order to learn an optimal policy in relation to the training data set we propose a reinforcement learning algorithm which utilizes deep deterministic policy gradient algorithms. We highlight the main ingredients of our approach that we are going to use the reinforcement learning problem framework to identify our problem. The explicit reward function is given below.

4.3.1 State and Action Representation

The agent is the software portfolio director, responsible to conduct trading activities on an algorithmic portfolio management level in the financial market environment. This environment includes all market assets available and all market participants' expectations.

The agent can not obtain complete information on a large and complex environment such as the stock market, so the asset prices that are given to the agent are reflected all orders throughout the market's history. The complete Price Vector X_t consists of processed (OHLC) prices for every stock within the training data period.

We mentioned in Section 4.2 that the agent's trade action will not affect future market price states. However, the action taken at the start of the t period will have an effect on the reward of $t+1$ and will therefore have an influence on the decision. The agent aim of redistributing the assets is resolute by the difference in portfolio weight $w_t^{'}$ and w_t, which also takes part in the last period in the action. Since the last period of the w_{t-1} was established, the action of the agent can at that time be entirely enunciated via the w_t vector of the portfolio(see Equation 3.13).

A previous measure therefore has an impact on the decision on the current dependence, this effect is imprinted by including w_{t-1} in the environment, so the t status is shown to be X_t and w_{t-1} (see Equation 3.17). Because of the *Zero market impact* in Section 4.2

the amount of the portfolio in comparison to the market's overall volume of trading is negligible, therefore p_t is not included in the internal state.

So the State (at time t) is the input matrix X_t and the previous portfolio Weights w_{t-1} (at time $t-1$) and formulate as 2-tuple:

$$State = S_t = (X_t, w_{t-1}) \tag{4.2}$$

The Action (at time t) is the vector of investment weights which change at every step and described as:

$$Action = \alpha_t = w_t = (w_1, \ldots, w_n) \tag{4.3}$$

The deterministic policy may be seen as a special case of stochastic policy where there is only one extreme non-zero value over one measure in the probability distribution. We expect the stochastic policy to require more samples as it integrates the data over the whole state and action space. The deterministic policy gradient theorem can be plugged into common policy gradient frameworks.

4.3.2 Exploration and the Reward Function

At the end of the $t_f + 1$ period, it is the agent's responsibility to maximize the final portfolio value p_f. Because the agent has no control of p_0, and the entire portfolio management process, t_f, the job is equal to maximizing the average cumulative return R:

Adjusted Reward = Reward - Baseline Reward - $\alpha \times$ Maximum Portfolio Weight

$$r_t = \sum w_i^{'} \times y_i - \frac{1}{m} \sum y_i - \alpha \max(w_1, \ldots, w_n) \tag{4.4}$$

Where α is a fix term for balancing the portfolio, preventing the choice of the greatest stock in every step.

The reward function is defined such as it is the agent's return minus a baseline's return (baseline is an equi-weighted agent - invest in all the possible stocks in the same way) minus a term proportional to the maximum weight (this term is set-up to make the agent avoid to invest fully in one stock).

With this reward function, the current approach express exactly both episodic and cumulated rewards achieving to master domain knowledge and can be fully exploited by

the agent. The same segment of market history can be used to evaluate the different sequences in actions by using this isolation of actions and the external environment. Equally crucial for the final reward are also all occasional rewards.

For improving exploration we implement a random action function which taking random action with probability epsilon (greedy) during training. Empirically we choose the initial value of random action is set to 18% and decreasing for every episode. When is time to choose a action run a rand function and if the value is smaller than the current probability epsilon compute the weight of action state, otherwise choose a random stock. This logic is explained as: If rand()<epsilon then compute the α_t, else choose random stock m as action.

4.4 Policy Network

The policy functions π_θ will be built with three deep neural networks. The policy function is developed via a deep neural network that uses the input tensor (Shape $m \times 35 \times 4$) composed of:

- the m traded stocks (m = 10 stocks)
- the 4 matrix columns (processed OHLC)
- 35 previous time steps

A first convolution is realized resulting in a smaller tensor. Then, a second convolution is made resulting in 20 vector of shape ($m \times 1 \times 1$). The previous output vector is stacked. The last layer is a terminate convolution resulting in a unique m vector. Then, a cash bias is added and a softmax (see Equation 2.23) applied.

The neural network outcome is the action vector the agent will take. Then, the environment can compute the new vector of weights, following the new portfolio and finally the instant reward as seen in Figure: 4.2 .

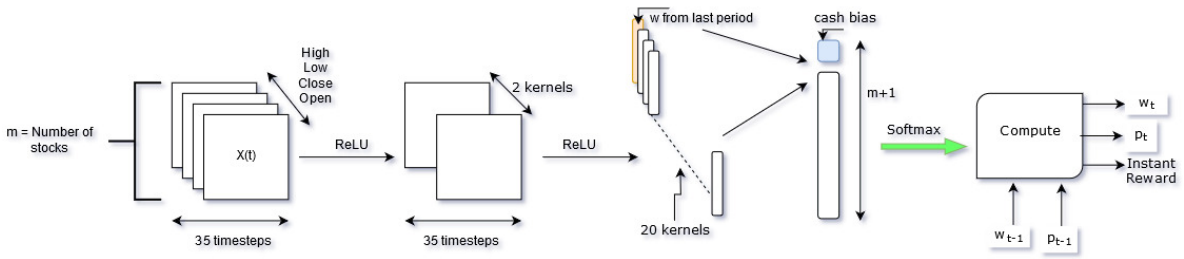


Figure 4.2: CNN Network Design (implemented from Jiang, Z., Xu, D., Liang, J. paper)

4.4.1 Portfolio-Vector Memory

In the network the result of portfolio weights from the previous trading period is used to reduce the transaction cost by restricting itself to major changes between successive portfolio vectors. The portfolio management agent is responsible for this. A dedicated network-specified Portfolio-Vector Memory (PVM) is introduced to the idea of replaying memory experience as shown by Mnih et al. [36] and Liu and Zou [37]. The PVM is a chronologically ordered heap of portfolio vectors. The PVM is initialized with uniform weights prior to any network training. A policy network loads the portfolio vector for the past period in $t-1$ from the memory location and overlaps the memory with its output at t at each stage. The values in memory also converge as the parameters of the policy grid converge in many training epochs. The sharing of a single storage stack allows a network to simultaneously train against data points in mini-batches, that greatly enhances the effectiveness of training.

4.4.2 Online Stochastic Batch Learning

The establishment of the network output-memory plauses mini-patch training, although the learning framework needs sequential inputs. In this training scheme however, the data points must be following time ordering within a batch, in contrast to supervised learning in which data points are unordered and mini-batches are random disjointed subsets of the sample space. Because data sets are time series, even with a significant overlapping interval, mini batch sets from various periods are also considered valid and distinctive.

The on-going nature of financial market and the size of a training sample set grow indefinitely, so new data is still being transferred to the agent. Fortunately, with the time distance between them, the correlation between two market price events is thought to decrease exponentially as shown by Winters [38] and Holt [39].

The price change during this period is added to the training set at the end of the $t-$th period. The policy network will be trained against N_b random mini-batches from this set once the agent has completed his orders during period $t+1$. A batch starting with period $t_b \leq t - n_b$ is selected with a geometrically distributed probability $P_\beta(t_b)$,

$$P_\beta(t_b) = \beta(1-\beta)^{t-t_b-n_b} \tag{4.5}$$

where $\beta \in (0,1)$ is the decline in probability that determines the form and gravity of new market events in the distribution of probabilities and n_b is the size of duration in mini batch.

4.5 The environment

The trading environment of our agent is called by giving an action at the time t. Then gives back the new portfolio at the next step (time t+1). Below are the parameters of the environment:

- **window length:** this is the number of time steps considered relevant from the past to build the input tensor
- **portfolio value:** this is the initial value of the portfolio
- **interest rate:** this is the rate of interest (in % of the traded stocks) the agent pays to perform the action
- **transaction cost:** this is the cost (in % of the traded stocks) the agent will pay to execute the action

4.6 Evaluation Performance

We will explain in this section the different types of agents that we use and the measurements that we display. There are 4 type of agents:

- **Agent_equal_weight**: set up for an agent who only plays an equally_weighted portfolio (baseline) between all stocks.
- **Agent_random_static:** set up for an agent who only play with random initial weights and keep them fix until the end.
- **Full_stock[m]:** set up for agents who play only on one stock.
- **The proposed Agent:** this is the proposed trading agent (policy network agent).

We choose some metrics to evaluate the performance of the portfolio value in each episode which were described in Section 3.1.2 of training period in the validation set:

- Sharpe Ratio
- Sortino Ratio
- Omega Ratio
- Maximum DrawDown
- Mean price
- Max-Min price

4.7 Implementation in Tensorflow

In this Section we will explained how we collect the data, set-up the environment and build the algorithm we proposed with Tensorflow and Python.

4.7.1 Input tensor

We use python numpy and pandas libraries to download the S&P 500 data used to produces the input tensors of the neural network. The data is provides by Yahoo Finance App and store as "csv" files and transposes to create a numpy nd-array.

Process 1 Environment Calculations

load the whole data
Initialize environment with given initial weights and given value of portfolio
Get the return of each stock for the day t
w_alloc=action, pf_alloc=pf_previous
Compute transaction cost
Convert weight vector into value vector, v_alloc = pf_alloc×w_alloc
Pay transaction costs, pf_alloc - cost
Market prices evolution, we go to the end of the day
Compute new value vector and portfolio value
Compute weight vector
Compute instantaneous reward, reward = $\frac{pf\_evol - pf\_previous}{pf\_previous}$
Proceed to the next day
Compute new state
return state,reward

4.7.2 The trading environment

In each step t, the trading agent gives as input the action he wants to do. So, he gies the new value of the weights of the portfolio. The environment compute the new value of the portfolio at the step $(t+1)$, and returns the reward associated with the action the agent took.

The Process 1 is shown the steps which required to compute reward and state at each step.

4.7.3 Description of the Actor

We create a class to incorporate the policy network agent. We use Tensorflow library, specifically tf.Placeholder() and tf.variable_scope() methods to define neural network as a class.

The Process 2 is shown the steps which required to define the CNN we use. Conv1, Conv2 and Conv3 uses Relu activation function, on the contrary Policy_Output as the last layers use a softmax activation Function.

Process 2 Neural Network Design

1: Create $X_t, w_{previous}, pf_{value\_previous}, daily\_r_t$
2: Create cash bias
3: Shape of the X_t == batchsize
4: First layer on the X_t tensor ("Conv1")
5: Feature maps ("Conv2")
6: w from last period ("Tensor3")
7: Feature map WITHOUT cash bias ("Conv3")
8: Feature map WITH cash bias ("Tensor4")
9: action = softmax(Tensor4) ("Policy_Output")
10: Compute the insta reward
11: Compute the equally reward
12: Find the max weight
13: Compute the Adjusted Reward
14: Call Adam Optimizer to maximize the Adjusted Reward

4.7.4 The Metrics for the evaluation

We keep list of portfolio value in every step of training to estimate the metrics and the performance of the agent during validation set. The results of training period will shown in Section 5 with the values for all metrics.

4.8 The Suggested Algorithm

We sum up our approach in this area and describe the optimization algorithm proposed. Our optimization approach is based on a CNN with a policy gradient as an RL method. Algorithm 3 draft our weights update from time step $t = 0$ to the end of the training set, while Table 4.1 records all algorithm parameters together with our assumed values.

All experiments run for 8 episodes. In each episode we initialize the PVM with training parameters:

```
PVM(m, sample_bias, total_steps_train, batch_size, w_init_train)
```

and then proceed to reset the environment for all types of agents with the weight from PVM at the starting point for each batches. Afterwards, we compute portfolio value for

Algorithm 3 Portfolio optimization Algorithm

for $e = 0$ in range $(n\_episodes)$ **do**
  Initialize the PVM $(m, distribution, batchsize, totalsteps)$
  for $i = 1$ in range $(n\_batches)$ **do**
    Reset all the agents environments
    for $bs = 1$ in range $(batch\_size)$ **do**
      load the inputs from the previous loaded state
      Get price Vector $X_t, W_{previous}, Pf_{previous}$
      Compute Action
      given the s_t (equation: 3.17) and $\alpha_t(equation:\ 3.13)$ go to next step
      get the new states
      update the PVM
      store elements
      if $bs == batchsize - 1$ **then**
        online training end
      end if
    end for
    train the network to maximize the reward
    call actor.train($X_t, W_{previous}, pf_{previous}$, dailyReturn)
    call Adam optimizer (2.5.5)
    evaluates the performance of the different types of agents.
  end for
end for

each batch and send the list required to the actor to train the network for maximizing the reward.

`actor.train(X_t, W_previous, pf_value_previous, dailyReturn_t).`

Finally for each episode call the evaluations performance function to calculate the metrics and shown the portfolio value during validation set.

Table 4.1: Parameter values of localization algorithm

Parameter	Proposed Value	Description
e	8	Episodes number
$batch\_size$	35	Size of mini-batch during training
$n\_batches$	10	Number of Batch for every episode
pf_init_train	10000	Initial Investment amount
$ratio\_train$	60%	Percentage of training time
$ratio\_val$	20%	Percentage of validation time
$ratio\_test$	20%	Percentage of testing time
$Initial\_ratio\_greedy$	18%	Percentage of random actions
$ratio\_regul$	0.12	Regulation rate for max Weight
$learning$	0.09	Adam optimizer learning rate
$regularization$	$5e^{-9}$	L2 regularization coefficient applied to NN
$trading\_cost$	0.25/100	Cost per action
$interest\_rate$	0.02/250	Interest rate
$sample\_bias$	$5e^{-5}$	Beta geometric distribution for PVM

Chapter 5

Results

The results of our work are presented in this chapter. Firstly, we must make it clear that our approach estimates a good portfolio that overcomes the benchmark. The results are compared to some well developed strategies for portfolio decision. The main financial criterion compared with that is portfolio value and maximum draw-down additionally to the ratios of Sharpe, Sortino and Omega.

The efficiency of the Standard & Poor's 500 Indices is questioned in actual financial markets.

5.1 Standard & Poor's 500

Publicly traded companies are typically correlated by increasing the number of their remaining shares by the current share price in terms of their market value or market capitalisation (market capital) (Investopedia,2018) [40], or evenly:

$$MarketCap_{asset_{(i)}} = Volume_{asset_{(i)}} \times SharePrice_{asset_{(i)}} \tag{5.1}$$

The S&P 500 Standard is an American stock market index based on the market capitalizations of 500 large companies, (Investopedia, 2018) [41]. As stated by the Efficient Market Hypothesis (EMH) (Fama, 1970) [42] and the Capital Asset Pricing Model (CAPM) (Luenberger, 1997 [43]), the market index, S&P 500, is adequate and portfolio extract by its combining assets could not work to a higher standard. However, the CAPM and the EMH are not exactly fulfilled and trade opportunities can be used by appropriate strategies.

5.1.1 Companies And Data period

We choose some random companies from (S&P 500) index which are mentioned in Table 5.1 and Table 5.2. The data period we choose is from $2005 - January - 01$ until $2016 - September - 01$ 2937 days in total, where 60% is the training set (1762 days), followed by 20% validation set (588 days) and the final testing set is the remaining 20% (588 days).

5.2 Performance of the algorithm

5.2.1 Test A: Exploration

We run experiments for the 10 companies in Table 5.2 with epsilon greedy policy and without it. Figure 5.1 shows the portfolio value during training for 8 episodes and Table 5.3 shows the metrics we described in Section 3.1.2 for each episode with greedy policy.
Similarly, Figure 5.2 shows the portfolio value during training for 8 episodes and Table 5.4 the metrics we described in section 3.1.2 for each episode without e-greedy policy.

We see upward trend in every next episode in both cases. The number of episodes stop in 8 because after that number the portfolio value barely increased. The metrics noticed to be stagnant after some episodes which means there are not much room for improvement. The results are averages over 10 repeats with one run to have $10 \times 8 = 80$ episodes.
In Figure 5.3 we see the performance of different types of agents and that the proposed optimization approach outperform the results than our benchmark method based on equal-weight portfolio, and is very close to some random weight agent for the combinations of assets. The exact percentage distance between the 2 agent is show in Figure 5.4.

Similarly in Figure 5.5 we see the performance of different types of agents without e-greedy policy, again the proposed optimization approach in this case outperform both the results than our benchmark method based on equal-weight portfolio, and some random weight agent for the combinations of assets. The exact percentage distance between the 2 agent is show in Figure 5.6.

Table 5.1: Companies for testing

Name	Ticket	Market Cap
Sony Corporation	SNE	$61.77B
JPMorgan Chase & Co.	JPM	$345.40B
3M Company	MMM	$112.50B
International Business Machines	IBM	$122.02B
AT&T Inc.	T	$223.216B
Cisco Systems	CSCO	$205.69B
The Goldman Sachs Group	GS	$76.07B
The Walt Disney Company	DIS	$165.12B
Pfizer Inc.	PFE	$228.50B
Walmart Inc.	WMT	$281.98B

Table 5.2: Companies for testing (2)

Name	Ticket	Market Cap
Alphabet Inc.	GOOG	$775.69B
MO - Altria Group	MO	$92.45B
AMAZON	AMZN	$795.18B
APPLE Inc.	AAPL	$787.61B
BANK OF AMERICA Corporation	BAC	$274.41B
Chubb Limited	CB	$61.55B
Coca-Cola Company	KO	$207.29B
Bristol-Myers Squibb Company	BMY	$81.44B
Intel Corporations	INTC	$222.43B
Chevron Corporation.	CVX	$226.18B

The difference between sophisticated models and the naive approach is not statistically significant [44] and [45], they point out that really basic models perform quite well.

Finally, in Figure 5.7 we can see the course of each share from Table 5.2 if we only invested in it the full amount of capital.

Table 5.3: Metrics for portfolio value during training (A1)

Episode	Pf-value	MDD	Mean	Sharpe	Sortino	Omega
0	13592	1025	0.056	0.083	0.099	0.167
1	14025	1163	0.062	0.082	0.1	0.171
2	14100	1186	0.063	0.081	0.1	0.172
3	14168	1207	0.064	0.081	0.1	0.173
4	14191	1215	0.065	0.081	0.1	0.173
5	14251	1234	0.065	0.081	0.1	0.173
6	14276	1242	0.066	0.081	0.1	0.173
7	14290	1246	0.066	0.081	0.1	0.173
8	14292	1247	0.066	0.081	0.1	0.174

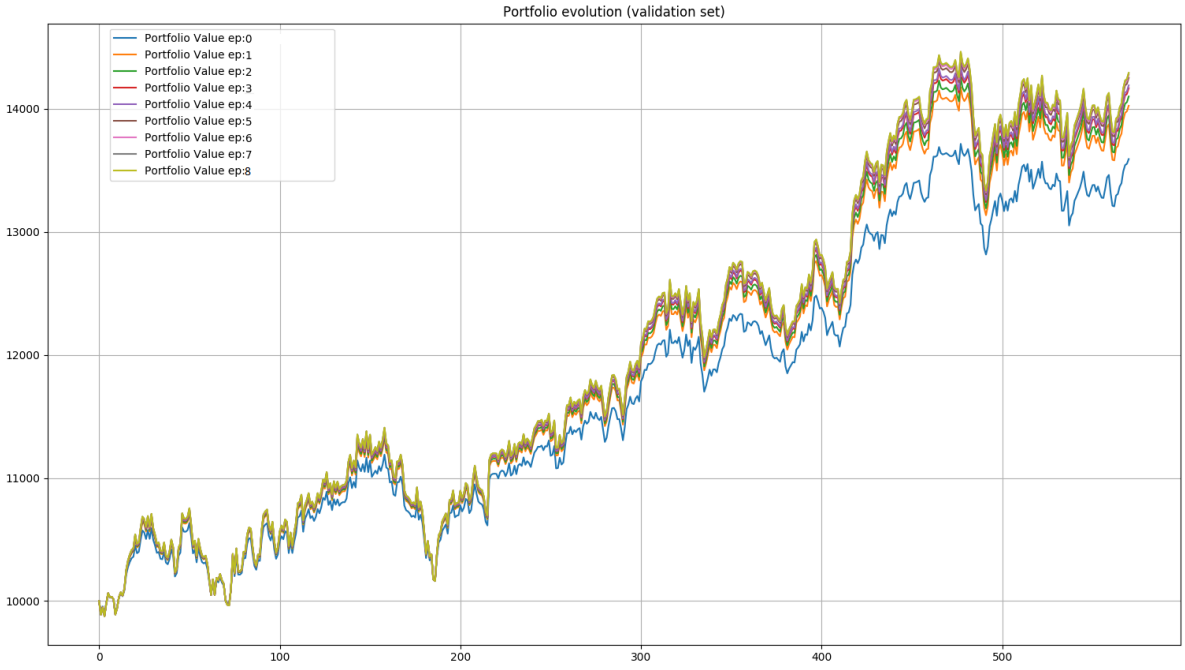


Figure 5.1: Portfolio value during training (A1) with exploration

Table 5.4: Metrics for portfolio value during training (A2)

Episode	Pf-value	MDD	Mean	Sharpe	Sortino	Omega
0	13603	1025	0.056	0.083	0.099	0.167
1	13977	1147	0.062	0.082	0.1	0.171
2	14170	1208	0.064	0.081	0.1	0.173
3	14191	1215	0.065	0.081	0.1	0.173
4	14164	1206	0.064	0.081	0.1	0.173
5	14200	1218	0.065	0.081	0.1	0.173
6	14247	1232	0.065	0.081	0.1	0.173
7	14272	1240	0.066	0.081	0.1	0.173
8	14280	1243	0.064	0.081	0.1	0.173

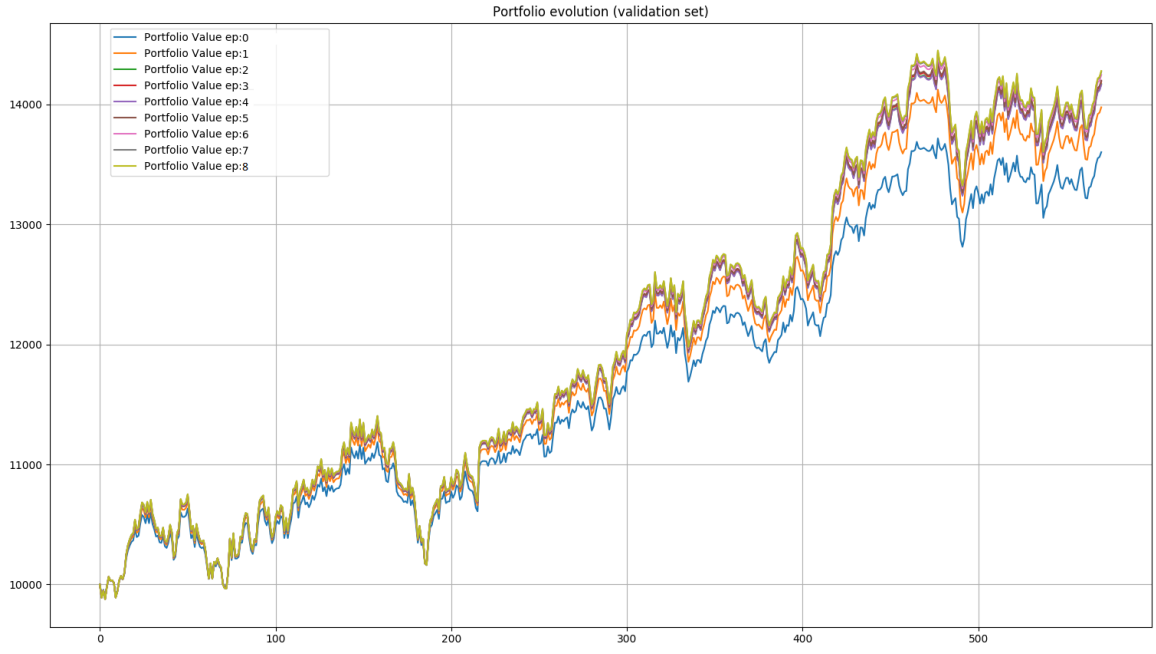


Figure 5.2: Portfolio value during training (A2) without exploration

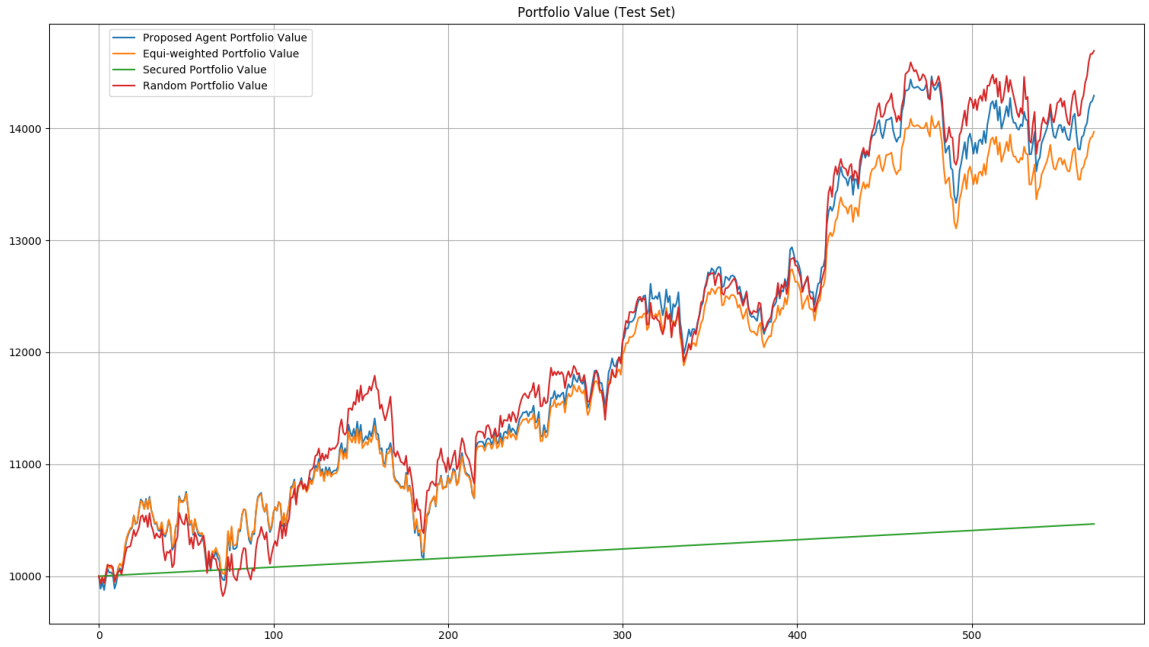


Figure 5.3: Portfolio value during testing (A1)

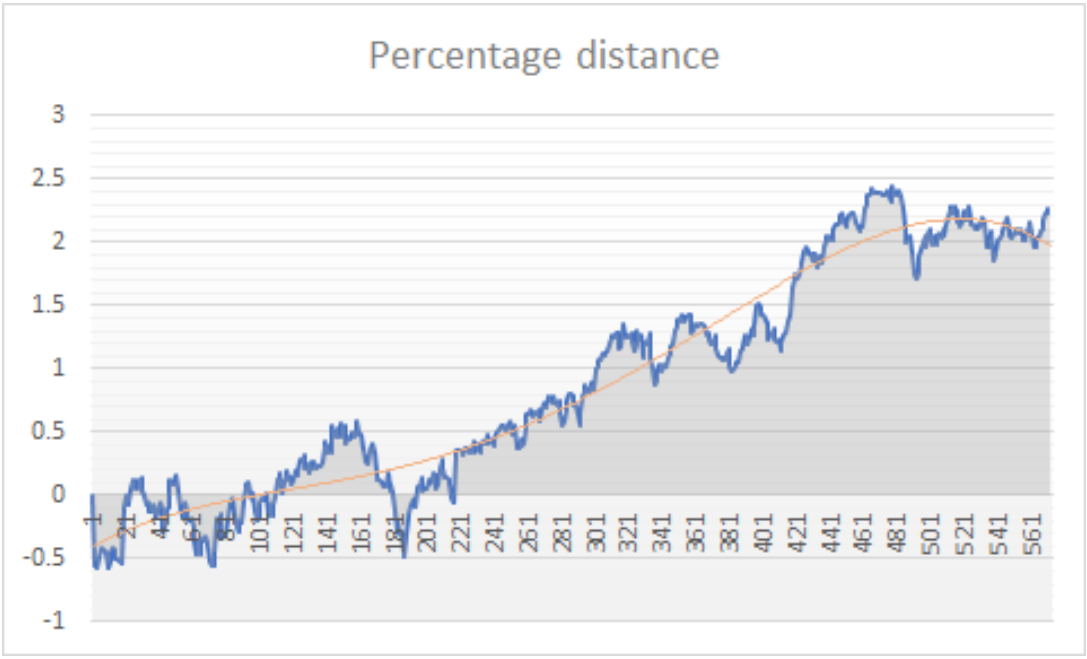


Figure 5.4: Distance between Proposed vs Equal-weighted Agents (A1)

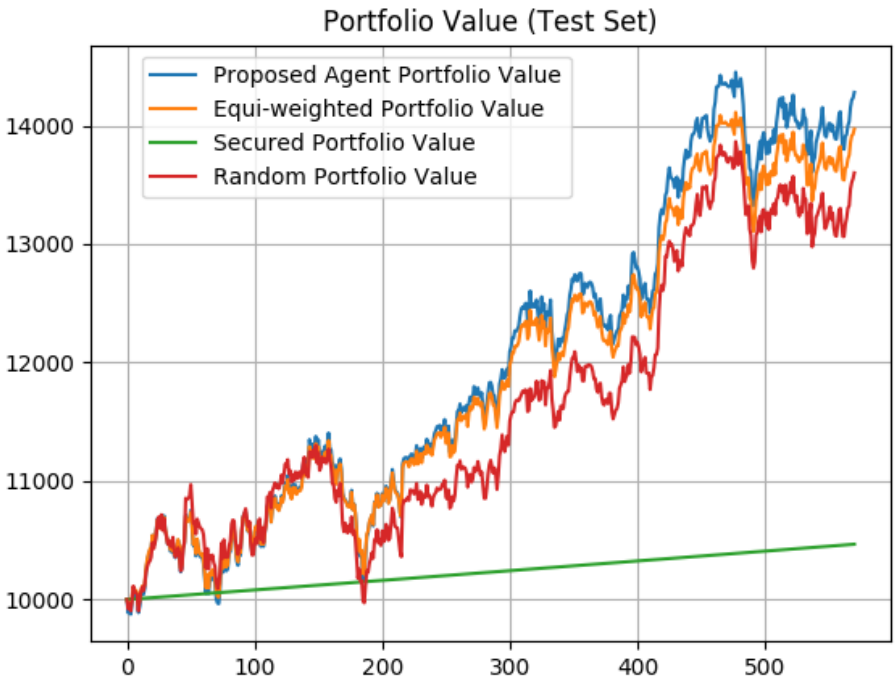


Figure 5.5: Portfolio value during testing (A2)

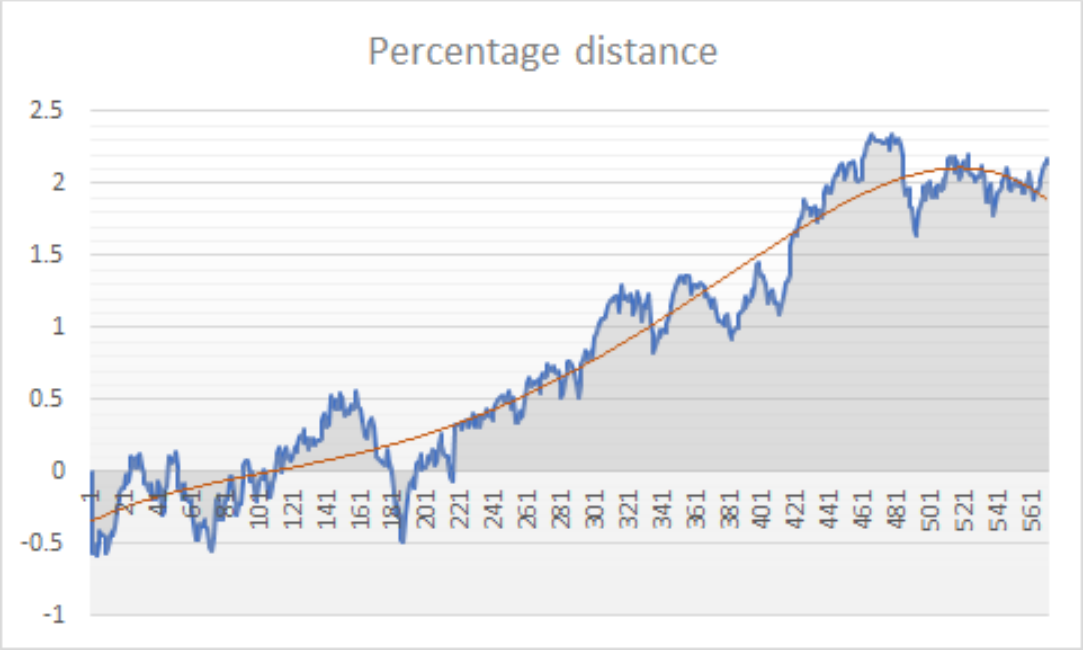


Figure 5.6: Distance between Proposed vs Equal-weighted Agents (A2)

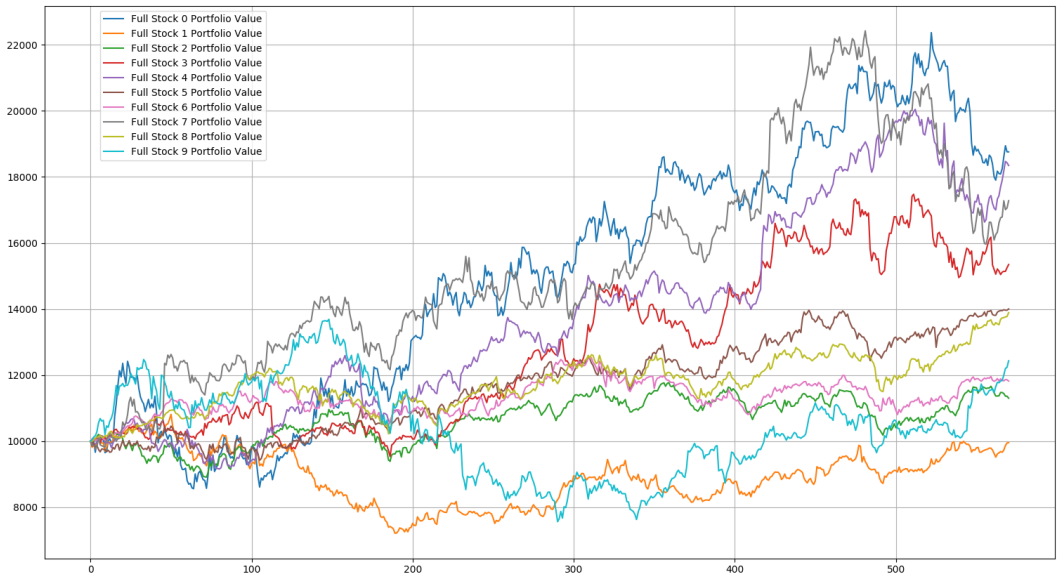


Figure 5.7: Portfolio value - Full invest in one stock (A)

5.2.2 Test B: Adam vs Adagrad Optimizer

In the second experiment, we test the 10 companies from Table 5.1 with adagrad and adam optimization Algorithms.

Adam: In the Table 5.5 we can see the metrics for each episodes during training. In Figure 5.8 we observe the portfolio value for each episode of training, combine with Figure 5.9 which show the weights evolution. Especially in episode 1 seven weights vanish for this episode but after that the percentage of each weight does not change significantly, and state stable until the then of training. In Figure 5.10 we see the performance of different types of agents with the Adam optimizer, again the proposed optimization approach in this case outperform both the results than our benchmark method based on equal-weight portfolio, and some random weight agent for the combinations of assets. The exact percentage distance between the 2 agent is show in Figure 5.11.

The Adam algorithm demonstrated the capacity to identify high-potential stocks which maximizes results. However, it has a little potential to change the position through the trading process. He finds quickly the optimal portfolio value and we have not big changes during training as shown in 5.5. The change of weights is very poor. It seems the policy is "Training Sensitive" but not "State Sensitive".

Table 5.5: Metrics for portfolio value during training (Adam optimizer)

Episode	Pf-value	MDD	Mean	Sharpe	Sortino	Omega
0	12591	1251	0.043	0.062	0.073	0.123
1	13424	1371	0.056	0.062	0.076	0.134
2	13030	1465	0.050	0.062	0.074	0.127
3	13035	1468	0.050	0.062	0.074	0.127
4	13028	1464	0.050	0.062	0.074	0.127
5	13014	1457	0.050	0.062	0.074	0.127
6	13020	1460	0.050	0.062	0.074	0.127
7	13039	1470	0.050	0.062	0.074	0.127
8	13051	1476	0.050	0.061	0.074	0.127

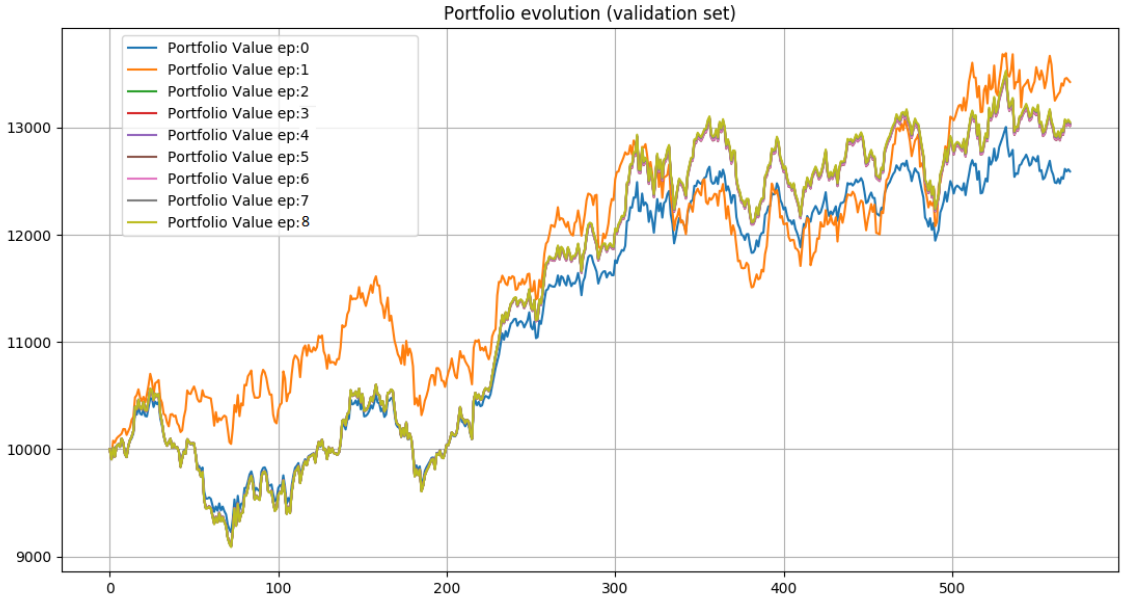


Figure 5.8: Portfolio value during testing (Adam optimizer)

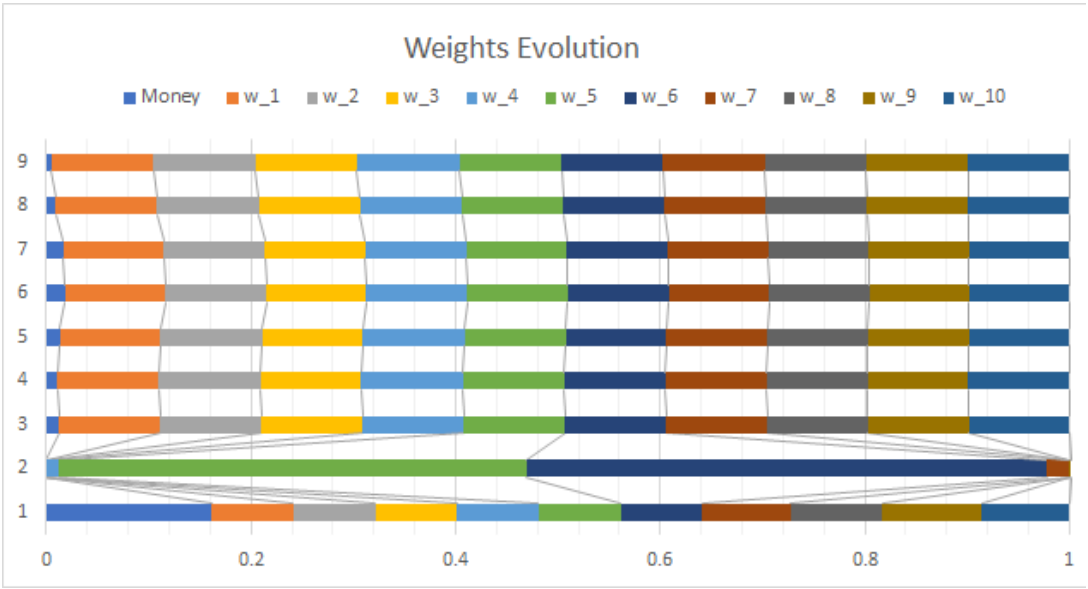


Figure 5.9: Weights evolution for each episode during training (Adam)

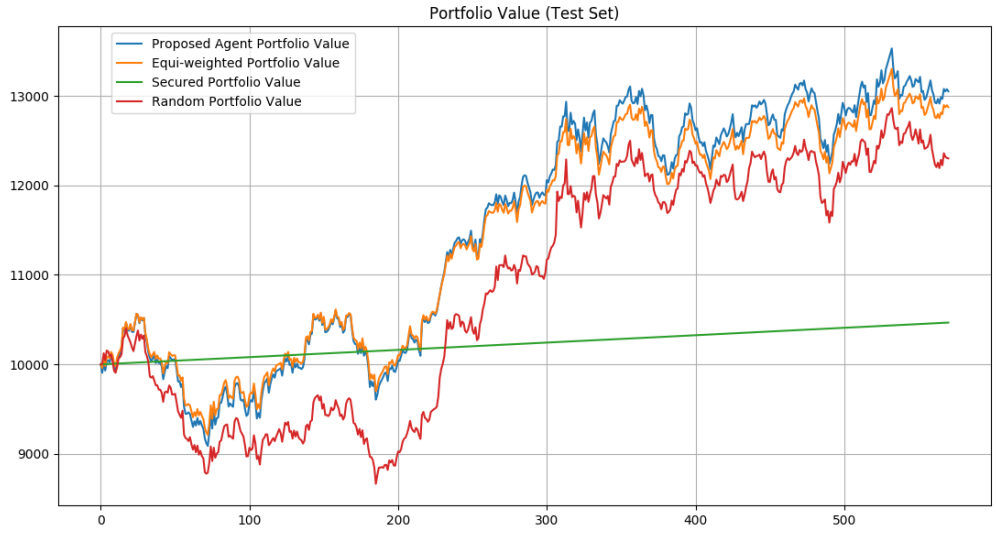


Figure 5.10: Portfolio value during testing (Adam)

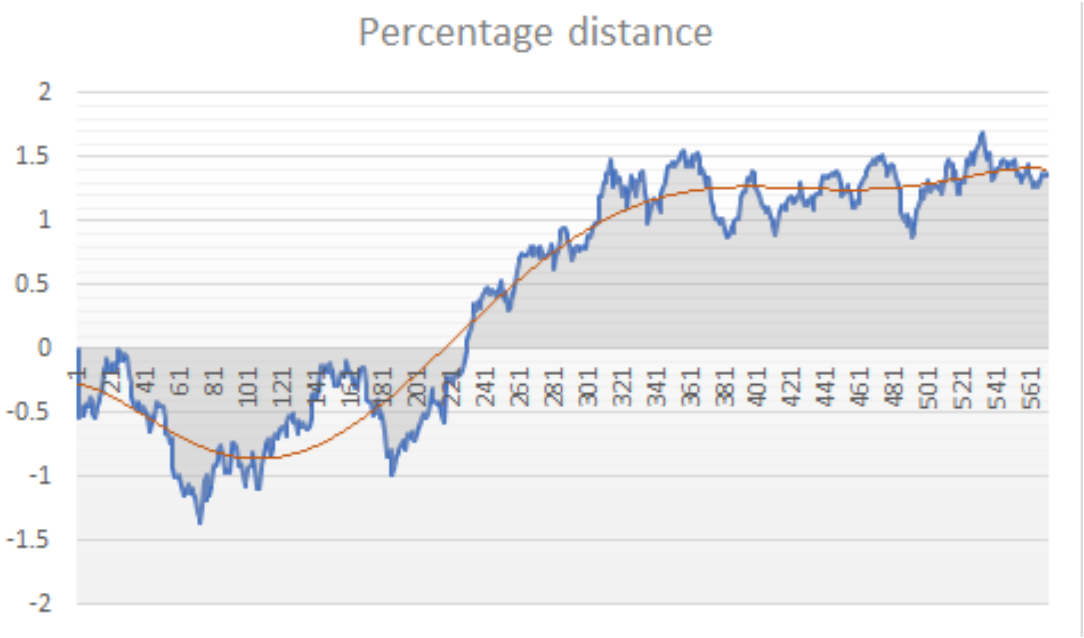


Figure 5.11: Distance between Proposed vs Equal-weighted Agents (Adam)

Adagrad: This section describe the testing of the 10 companies from Table 5.1 with Adagrad optimization algorithms. In the Table 5.6 we can see the metrics for each episodes during training. In Figure 5.12 we observe the portfolio value for each episode of training, combine with Figure 5.13 which show the weights evolution. It has more linear progression comparatively with the Adam weights evolution in Figure 5.13, but appear the weakness to eliminate the "money". In Figure 5.14 we see the performance of different types of agents with the Adagrad optimizer, in this case the proposed optimization approach is less "'optimal-to-equal" from benchmark method based on equal-weight portfolio, and some random weight agent for the combinations of assets. The exact percentage distance between the 2 agent is show in Figure 5.15.
The Adagrad algorithm demonstrated the capacity to identify high-potential stocks which maximizes results. However, change the position over time , but not always to better outcome. He has inability to decrease the "money" and to find quickly the optimal portfolio value 5.6. The change of weights is very poor. It seems the policy is "Training Sensitive" but not "State Sensitive".

Finally, in Figure 5.16 we can see the course of each share from Table 5.1 if we only invested in it the full amount of capital.

Table 5.6: Metrics for portfolio value during training (Adam optimizer)

Episode	**Pf-value**	**MDD**	**Mean**	**Sharpe**	**Sortino**	**Omega**
0	12574	1278	0.043	0.062	0.073	0.122
1	12605	1359	0.043	0.059	0.070	0.118
2	12735	1360	0.045	0.061	0.072	0.123
3	12826	1388	0.046	0.062	0.074	0.125
4	12897	1386	0.047	0.063	0.075	0.128
5	12861	1363	0.047	0.063	0.075	0.127
6	12881	1377	0.047	0.063	0.075	0.128
7	12873	1362	0.047	0.063	0.075	0.128
8	12878	1348	0.047	0.064	0.076	0.128

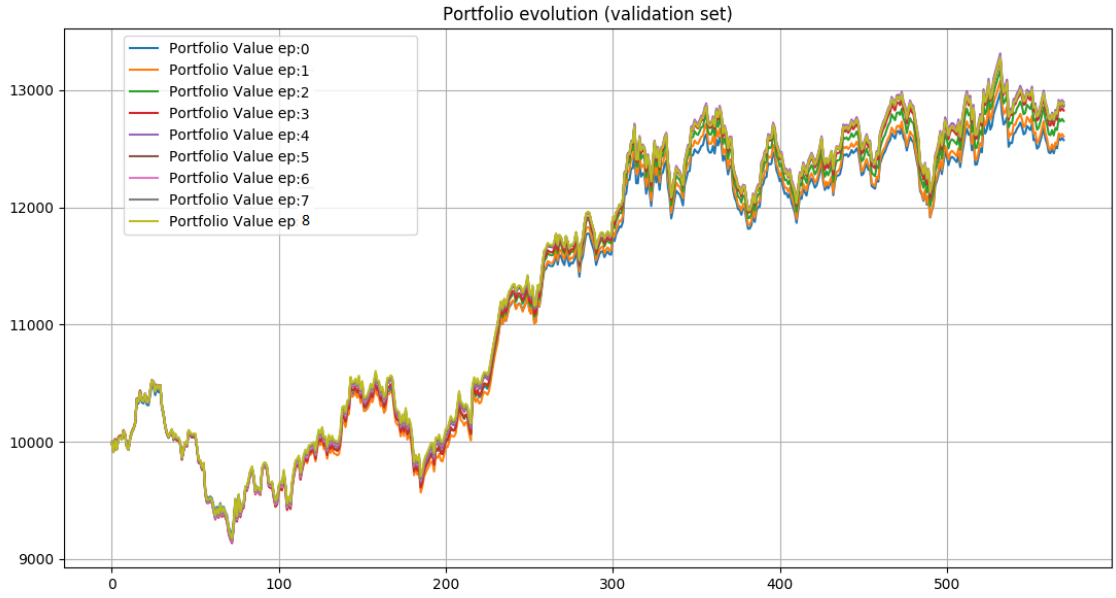


Figure 5.12: Portfolio value during testing (Adagrad optimizer)

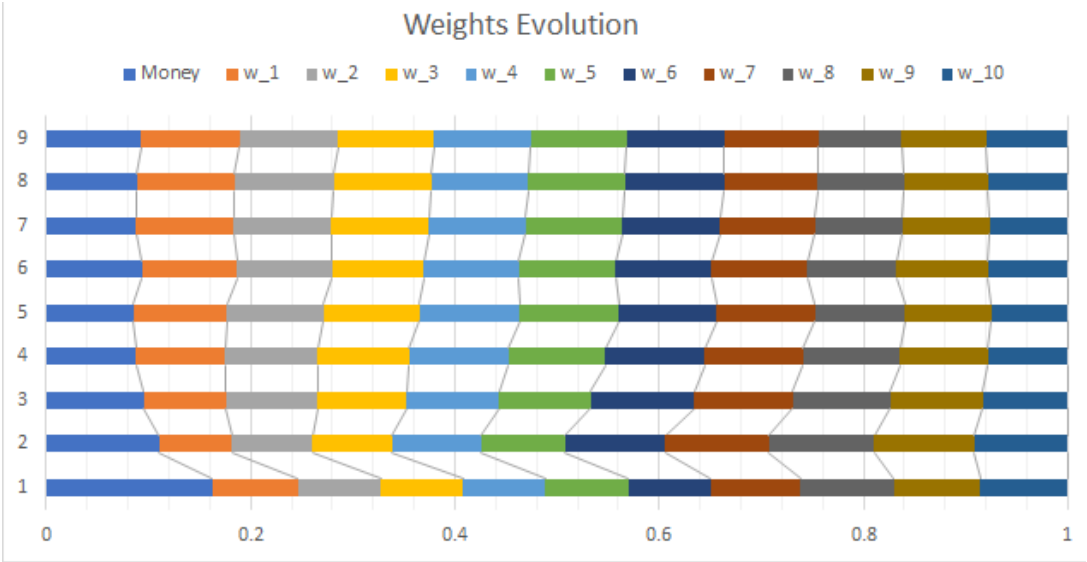


Figure 5.13: Weights evolution for each episode during training (Adagrad)

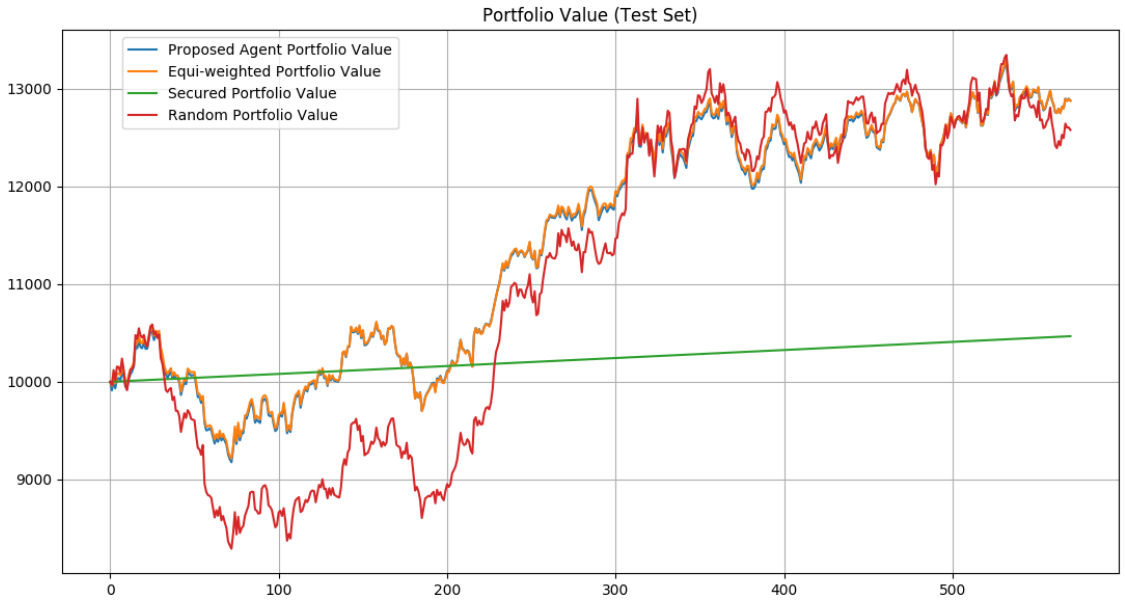


Figure 5.14: Portfolio value during testing (Adagrad)

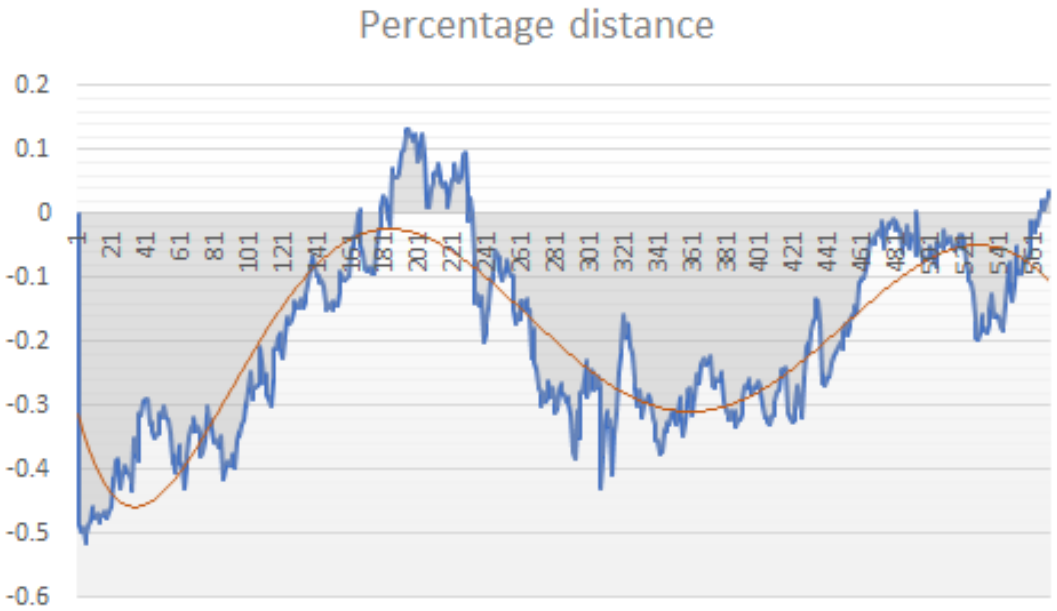


Figure 5.15: Distance between Proposed vs Equal-weighted Agents (Adagrad)

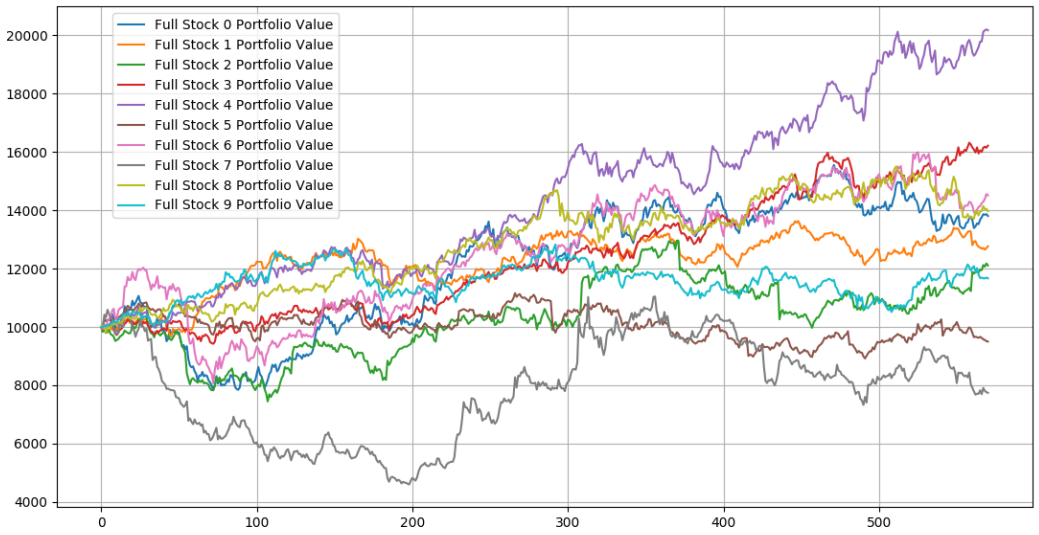


Figure 5.16: Portfolio value - Full invest in one stock (B)

Chapter 6

Conclusion

6.1 Conclusion

A deep deterministic reinforcement approach to portfolio optimization is described in this thesis, which has been implemented and incorporated using Tensorflow and Python. The optimization algorithm takes as input historical prices from 10 stocks from S&P index and estimates a good portfolio with combination of weights that are updated in every step. We make use of a Convolution Neural Network for policy function with Policy gradient as the RL method. The proposed approach to optimization gives more lucrative results than our equivalent portfolio benchmark method [44] and [45]. We conclude our research by assessing our results with the objectives set out in the introduction and discussing their impact on both reinforcement learning and the optimization of portfolios. We also suggest some possible ideas for extension of our approach in the form of future work.

6.2 Future Work

The lack of interpretability as shown by Rico-Martinez etc. [46] and the lack of thorough testing of depths (i.e. deep networks) of the architecture used discourages practitioners from using these solutions even though the strategies developed improve on performing. As a result, the learned strategies should be explored and interpreted through the opening of the deep "black box".

- Since the financial signals are uncertain, such insecurity should be shaped and integrated in the process of decision-making. Bayesian inference can also be used to train probabilistic models which are capable of capturing environmental uncertainty, including Variation inference as shown by Titsias and Lawrence [47], and Vlassis et al. [48].

- A interesting approach is to construct some indicators whose typical used in technical analysis of financial Series. Strategy performance resulting from conditional optimization and the use of several possible signal indicators, could help in guiding towards better decisions [49] or technical indicators with Multi-Objective Evolutionary Algorithms [50].

- Most extensions of Markowitz extensions not only perform poorly against the naive 1/N rule (that invests equally across N assets) in simulations, but also in many real data sets they lose money on a risk-adjusted basis [51]. An optimal combination of 1/N naive rule and one of the four advanced strategies such that the Markowitz rule, the Jorion rule [52], the MacKinlay and Pastor rule [53] and the Kan and Zhou rule [54] will enhance the performance of the agent.

References

[1] Investopedia: Financial concepts: The optimal portfolio (2010) [online] https://www.investopedia.com/university/concepts/concepts7.asp. 1

[2] Byrne, P., Lee, S.: Risk reduction and real estate portfolio size. Managerial and Decision Economics **22** (10 2001) 369–379 3

[3] Kennedy, D.: Stochastic financial models. Chapman and Hall/CRC. (2016) 5

[4] Silver, D.: Markov decision processes. (2015a) [online] http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching_files/MDP.pdf. 8, 10

[5] Bellman, R.: Dynamic Programming. 1 edn. Princeton University Press, Princeton, NJ, USA (1957) 10

[6] Szepesvari, C.: Algorithms for Reinforcement Learning. Volume 4. (01 2010) 11

[7] Sutton, R.S., Barto, A.G.: Reinforcement learning - an introduction. Adaptive computation and machine learning. MIT Press (1998) 12

[8] Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, D.: Continuous control with deep reinforcement learning. CoRR **abs/1509.02971** (2015) 12

[9] Duchi, J., Hazan, E., Singer, Y.: Adaptive subgradient methods for online learning and stochastic optimization. J. Mach. Learn. Res. **12** (July 2011) 2121–2159 19

[10] Zeiler, M.D.: ADADELTA: an adaptive learning rate method. **abs/1212.5701** (2012) 20

[11] Hinton, G.: Rmsprop algorithm (2010) [online] http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf. 20

[12] Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. **abs/1412.6980** (2014) 21

[13] Fischer, T.G.: Reinforcement learning in financial markets - a survey. FAU Discussion Papers in Economics 12/2018, Erlangen (2018) 23

[14] Markowitz, H.: Portfolio selection*. The Journal of Finance **7**(1) (1952) 77–91 24, 29

[15] Wilmott, P.: Paul Wilmott Introduces Quantitative Finance. 2 edn. Wiley-Interscience, New York, NY, USA (2007) 26

[16] Tsay, R.: Analysis of financial time series. 2. ed. edn. Wiley series in probability and statistics. Wiley-Interscience, Hoboken, NJ (2005) 26

[17] Elias S. W. Shiu A S.A, P.: Luenberger, david g., 1997, investment science. North American Actuarial Journal **3**(2) (1999) 150–150 26

[18] Sharpe, W.F.: The sharpe ratio. The Journal of Portfolio Management **21**(1) (1994) 49–58 27

[19] Zhang, X.P.S., Wang, F.: Signal processing for finance, economics, and marketing: Concepts, framework, and big data applications. IEEE Signal Processing Magazine **34** (2017) 14–35 27

[20] Sortino, F.A., Price, L.N.: Performance measurement in a downside risk framework. The Journal of Investing **3**(3) (1994) 59–64 28

[21] Keating, C., Shadwick, W.: A universal performance measure. Journal of Performance Measurement **6** (01 2002) 59–84 28

[22] Chekhlov, A., Uryasev, S., Zabarankin, M.: Drawdown measure in portfolio optimization. International Journal of Theoretical and Applied Finance (IJTAF) **08** (01 2005) 13–58 29

[23] Mansini, R., Ogryczak, W., Speranza, M.G. In: Portfolio Optimization with Transaction Costs. Springer International Publishing, Cham (2015) 47–62 31

[24] Metelli, A.M., Pirotta, M., Restelli, M.: Compatible reward inverse reinforcement learning. In Guyon, I., Luxburg, U.V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., Garnett, R., eds.: Advances in Neural Information Processing Systems 30. Curran Associates, Inc. (2017) 2050–2059 33

[25] Ng, A.Y., Russell, S.J.: Algorithms for inverse reinforcement learning. In: Proceedings of the Seventeenth International Conference on Machine Learning. ICML '00, San Francisco, CA, USA, Morgan Kaufmann Publishers Inc. (2000) 663–670 33

[26] Shen, S., Jiang, H., Zhang, T.: Stock market forecasting using machine learning algorithms. (2012) 34

[27] Moody, J., Saffell, M.: Learning to trade via direct reinforcement. IEEE Transactions on Neural Networks **12**(4) (July 2001) 875–889 34

[28] Almahdi, S., Yang, S.: An adaptive portfolio trading system: A risk-return portfolio optimization using recurrent reinforcement learning with expected maximum drawdown. Expert Systems with Applications **87** (06 2017) 35

[29] Deng, Y., Bao, F., Kong, Y., Ren, Z., Dai, Q.: Deep direct reinforcement learning for financial signal representation and trading. IEEE Transactions on Neural Networks and Learning Systems **28** (2017) 653–664 35

[30] Moody, J.E., Saffell, M.: Reinforcement learning for trading systems and portfolios. In: KDD 1998. (1998) 36

[31] Neuneier, R.: Optimal asset allocation using adaptive dynamic programming. In: in Advances in Neural Information Processing Systems, MIT Press (1996) 952–958 36

[32] Necchi, P.: Policy gradient algorithms for asset allocation problem. (2016) [online] https://github.com/pnecchi/Thesis/blob/master/MS_Thesis_Pierpaolo_Necchi.pdf. 36

[33] Jiang, Z., Xu, D., Liang, J.: A deep reinforcement learning framework for the financial portfolio management problem (06 2016) [online], https://arxiv.org/abs/1706.10059. 37

[34] LeCun, Y., Bottou, L., Orr, G.B., Müller, K.R.: Efficient backprop. In: Neural Networks: Tricks of the Trade, London, UK, UK, Springer-Verlag (1998) 9–50 39

[35] Leslie Evans, J., H. Archer, S.: Diversification and the reduction of dispersion: An empirical analysis. The Journal of Finance **23** (12 1968) 40

[36] Mnih, V., Badia, A.P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., Kavukcuoglu, K.: Asynchronous methods for deep reinforcement learning. In Balcan, M.F., Weinberger, K.Q., eds.: Proceedings of The 33rd International Conference on Machine Learning. Volume 48 of Proceedings of Machine Learning Research., New York, New York, USA, PMLR (20–22 Jun 2016) 1928–1937 46

[37] Liu, R., Zou, J.: The effects of memory replay in reinforcement learning. 2018 56th Annual Allerton Conference on Communication, Control, and Computing (Allerton) (2018) 478–485 46

[38] R. Winters, P.: Forecasting sales by exponentially weighted moving averages. Management Science **6** (04 1960) 324–342 47

[39] C. Holt, C.: Forecasting seasonals and trends by exponential weighted moving averages. International Journal of Forecasting **20** (03 2004) 5–10 47

[40] Investopedia: Market value (2018) [online] https://www.investopedia.com/terms/m/marketvalue.asp. 53

[41] Investopedia: Standard & poor's 500 index - s&p 500. (2018) [online] https://www.investopedia.com/terms/s/sp500.asp. 53

[42] Malkiel, B.G., Fama, E.F.: Efficient capital markets: A review of theory and empirical work*. The Journal of Finance **25**(2) (1970) 383–417 53

[43] Luenberger, D.G.: Investment Science. Oxford University Press (1997) 53

[44] DeMiguel, V., Garlappi, L., Uppal, R.: Optimal Versus Naive Diversification: How Inefficient is the 1/N Portfolio Strategy? The Review of Financial Studies **22**(5) (12 2007) 1915–1953 56, 69

[45] Hwang, I., Xu, S., In, F.: Naive versus optimal diversification: Tail risk and performance. European Journal of Operational Research **265**(1) (2018) 372 – 388 56, 69

[46] Rico-Martinez, R., Anderson, J.S., Kevrekidis, I.G.: Continuous-time nonlinear signal processing: a neural network based approach for gray box identification. In: Proceedings of IEEE Workshop on Neural Networks for Signal Processing. (Sep. 1994) 596–605 69

[47] Titsias, M., Lawrence, N.D.: Bayesian gaussian process latent variable model. In Teh, Y.W., Titterington, M., eds.: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics. Volume 9 of Proceedings of Machine Learning Research., Chia Laguna Resort, Sardinia, Italy, PMLR (13–15 May 2010) 844–851 70

[48] Vlassis, N., Ghavamzadeh, M., Mannor, S., Poupart, P. In: Bayesian Reinforcement Learning. Springer Berlin Heidelberg, Berlin, Heidelberg (2012) 359–386 70

[49] Schiltz, J., Boissaux, M.: Practical weight-constrained conditioned portfolio optimization using risk aversion indicator signals. LSF Research Working Paper Series 11-12, Luxembourg School of Finance, University of Luxembourg (2011) 70

[50] Silva, A., Neves, R., Horta, N.: Portfolio optimization using fundamental indicators based on multi-objective ea. IEEE/IAFE Conference on Computational Intelligence for Financial Engineering, Proceedings (CIFEr) (10 2014) 158–165 70

[51] Tu, J., Zhou, G.: Markowitz meets talmud: A combination of sophisticated and naive diversification strategies. Journal of Financial Economics **99**(1) (2011) 204 – 215 70

[52] Jorion, P.: Bayes-stein estimation for portfolio analysis. Journal of Financial and Quantitative Analysis **21** (09 1986) 279–292 70

[53] Pastor, D.: Portfolio selection and asset pricing models. The Journal of Finance **55**(1) (2000) 179–223 70

[54] Kan, R., Zhou, G.: Optimal portfolio choice with parameter uncertainty. Journal of Financial and Quantitative Analysis **42** (09 2007) 621–656 70