

A DEEP REINFORCEMENT LEARNING
FRAMEWORK FOR FINANCIAL PORTFOLIO
MANAGEMENT

by

Jinyang Li

December 2019

Supervisor: David Saunders

A Master research paper
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Quantitative Finance

Abstract

In this research paper, we investigate into a paper named “A Deep Reinforcement Learning Framework for the Financial Portfolio Management Problem”. It is a portfolio management problem which is solved by deep learning techniques. The original paper proposes a financial-model-free reinforcement learning framework, which consists of the Ensemble of Identical Independent Evaluators (EIIE) topology, a Portfolio-Vector Memory (PVM), an Online Stochastic Batch Learning (OSBL) scheme, and a fully exploiting and explicit reward function. Three different instants are used to realize this framework, namely a Convolutional Neural Network (CNN), a basic Recurrent Neural Network (RNN), and a Long Short-Term Memory (LSTM). The performance is then examined by comparing to a number of recently reviewed or published portfolio-selection strategies. We have successfully replicated their implementations and evaluations. Besides, we further apply this framework in the stock market, instead of the cryptocurrency market that the original paper uses. The experiment in the cryptocurrency market is consistent with the original paper, which achieve superior returns. But it doesn't perform as well when applied in the stock market.

Acknowledgements

I would first like to thank my supervisor Prof. David Saunders of the Faculty of Mathematics at University of Waterloo. The door to his office was always open whenever I ran into a trouble spot or had a question about my research or writing. He consistently allowed this paper to be my own work, but steered me in the right direction whenever he thought I needed it. His guidance helped me in all the time of research and writing of this Master's research paper. I could not have imagined having a better advisor and mentor for my Master's study.

My sincere thanks also goes to Zhengyao Jiang, who is the author of the original paper that I have studied in this research paper, provided me an opportunity to discuss my confusions and offered great help to me during my implementation. Without his precious support it would not be possible to conduct this research.

Last but not the least, none of this could have happened without my family. My parents have paid the subscription fee for me to get the stock market data. They also offered their encouragement through phone calls every week – despite my own limited devotion to correspondence. My wife was always on my side and encouraged me whenever I felt frustrated. This Master's research paper stands as a testament to your unconditional love and encouragement.

Table of Contents

List of Tables	vi
List of Figures	vii
1 Introduction	1
2 Background and Related Work	3
2.1 Background	3
2.1.1 Portfolio Management Theory	3
2.1.2 Cryptocurrencies	5
2.1.3 Reinforcement Learning	6
2.2 Related Work	6
3 Problem Definition	8
3.1 Two Assumptions	8
3.2 Mathematical Formalism	9
3.3 Transaction Costs	10
4 Data Structure	13
4.1 Asset Selection	13
4.2 Price Tensor	14
5 Reinforcement Learning Framework	16
5.1 Reinforcement Learning Elements	16
5.2 Policy Networks	18
5.2.1 Network Topologies	19
5.2.2 Portfolio-Vector Memory	21
5.2.3 Online Stochastic Batch Learning	21
6 Numerical Experiments and Evaluation	23
6.1 In the Cryptocurrency Market	24
6.2 In the Stock Market	27
7 Conclusion and Future Work	31
Bibliography	33

Appendices	36
A Proofs of Theorems	37
A.1 Proof of the Formula for Transaction Remainder Factor	37
B Configurations for the Experiments	39
B.1 Experiment 1 Configurations	39
B.2 Experiment 2 Configurations	41
C GitHub Repository	44
C.1 Quickstart	44
C.2 Configuration File	45
C.3 Training the agent	45
C.4 Training results	47
C.5 Download Data	47
C.6 Backtest	48
C.7 Plotting	48
C.8 Table summary	48

List of Tables

2.1	General classification for the state-of-the-art online portfolio selection algorithms.	5
6.1	Training and Back-testing results for CNN, RNN and LSTM - <i>Back-test 1</i>	25
6.2	Performance of the three EIIEs, compared with traditional online selection algorithms - <i>Back-test 1</i>	26
6.3	Training and Back-testing results for CNN, RNN and LSTM - <i>Back-test 2</i>	29
6.4	Performance of the three EIIEs, compared with traditional online selection algorithms - <i>Back-test 2</i>	30

List of Figures

2.1	The Typical Framing of a Reinforcement Learning (RL) scenario	7
3.1	Illustration of trading between periods.	9
5.1	CNN Implementation of the EIIE	19
5.2	RNN (or LSTM) Implementation of the EIIE	20
6.1	Log-scale accumulated wealth from 2017-05-04 to 2017-07-01 - <i>Back-test 1</i> .	25
6.2	Log-scale accumulated wealth from 2019-10-14 to 2019-12-11 - <i>Back-test 2</i> .	27
6.3	Log-scale accumulated wealth (excluding Follow-the-Loser) from 2019-10-14 to 2019-12-11 - <i>Back-test 2</i>	28

Chapter 1

Introduction

The portfolio management problem is investigated in this research paper and we focus on a cutting-edge paper using deep machine learning techniques to solve it which is written by Jiang et al. (2017), here after referred to as “the original paper”. As we know, it is an optimization problem that continuously reallocates capital into different financial products, according to certain objective. The objective typically maximizes factors such as expect return, and minimizes the risk such as the standard deviation. The portfolio can be of wide range from tangible assets to intangible assets.

The original paper proposes a reinforcement learning network specially designed for the task of portfolio management. The core of the framework is the Ensemble of Identical Independent Evaluators (EIIE) topology. An Identical Independent Evaluators (IIE) is a neural network whose job is to analyze the history of an asset and then evaluates its potential growth for the immediate future. Three different types of IIEs are tested in the original paper, namely a Convolutional Neural Network (CNN), a basic Recurrent Neural Network (RNN) and a Long Short Term Memory (LSTM). The IIEs are trained on each asset and then assembled to a softmax layer for the new portfolio weights for the coming trading period. In order to take into account the effect of transaction costs, the historical portfolio weights are also inputs to the EIIE. For this purpose, the portfolio weights of each period are

recorded in a Portfolio Vector Memory (PVM). The EIIE is trained in an Online Stochastic Batch Learning (OSBL) scheme, which is compatible with both pre-trade training and online training during back-test or online trading.

This framework is not restricted to any particular market. The original paper tests its validity and profitability in the cryptocurrency exchange market. In this paper, we first implemented the experiments based on the same setting of Jiang’s paper. Then, we further evaluate this framework in the stock market. For both experiments, the performance of the three EIIEs (CNN, RNN and LSTM) is compared with some recently published or reviewed portfolio selection strategies (Li et al.; 2016; Li and Hoi; 2014). The EIIEs significantly beat all the other strategies when it is in the cryptocurrency market, but the performance is average in stock market.

The remainder of this paper is organized as follows. **Chapter 2** presents background information and related works in this field. **Chapter 3** defines the portfolio management problem that the framework is aiming to solve. **Chapter 4** is the implementation specifications, introducing the asset pre-selection process and the input price tensor. **Chapter 5** presents the reinforcement learning framework. The results of two experiments under the cryptocurrency and stock markets are illustrated in **Chapter 6**, as well as the evaluation and discussion. Finally, **Section 7** concludes the whole project and lists possible future works.

Chapter 2

Background and Related Work

In this chapter, we first talk about the background of portfolio management theory and the cryptocurrency market. Then we present related works that have been done using machine learning techniques.

2.1 Background

2.1.1 Portfolio Management Theory

Modern Portfolio Theory (MPT) was first introduced by Markowitz, who was later awarded a Nobel price for developing this theory. In a series of papers (Markowitz; 1952, 1959), the Markowitz model was proposed and developed. The model, also known as the mean-variance model, assists in the selection of the most efficient portfolio by analyzing various possible portfolios of given securities. It uses statistical analysis for measurement of risk and mathematical programming for selection of assets in an efficient manner. According to the model, it is possible to construct the efficient frontier of optimal portfolios offering the maximum possible expected return for a given level of risk.

Another type of theory called Capital Growth Theory (CGT) was proposed shortly afterwards by Kelly (1956); Hakansson and Ziembba (1995), which is primarily originated from

information theory. Instead of focusing on a single-period portfolio selection as it is in the MPT, CGT focuses on multiple-period or sequential portfolio selection, aiming to maximize the portfolio's expected growth rate or expected log return. While both theories solve the task of portfolio selection, the latter is fitted to the online scenario, which naturally consists of multiple periods and is the basis for the paper that we focus on.

Online portfolio selection, which sequentially selects a portfolio over a set of assets in order to achieve certain targets, is a natural and important task for asset portfolio management. Aiming to maximize the cumulative wealth, several categories of algorithms have been proposed to solve this task. According to Li and Hoi (2014), online portfolio selection algorithms can be classified into four categories, namely “Follow-the-Winner”, “Follow-the-Loser”, “Pattern-Matching Approaches” and “Meta-Learning Algorithms”.

- **Follow-the-Winner:** tries to asymptotically achieve the same growth rate as that of an optimal strategy, which is often based on the Capital Growth Theory.
- **Follow-the-Loser:** transfers the wealth from winning assets to losers, which seems contradictory to common sense but empirically often achieves significantly better performance.
- **Pattern-Matching Approaches:** tries to predict the next market distribution based on a sample of historical data and explicitly optimizes the portfolio based on the sampled distribution.
- **Meta-Learning Algorithms:** combines multiple strategies from the above three categories.

We selected some algorithms from the above categories, together with three benchmarks namely Best stock, Uniformly Constant Rebalanced Portfolios and Buy And Hold, to evaluate the performance of the EIIE neural networks. A table summarizing those algorithms is as below, see Table 2.1.

Classifications	Algorithms	Abbreviation
Benchmarks	Buy And Hold	ubah
	Best Stock	best
	Uniformly Constant Rebalanced Portfolios	ucrp
Follow-the-Winner	Universal Portfolios	up
	Exponential Gradient	eg
	Online Newton Step	ons
Follow-the-Loser	Anti Correlation	anticor
	Online Moving Average Reversion	olmar
	Passive Aggressive Mean Reversion	pamr
	Weighted Moving Average Passive Aggressive	wmamr
	Confidence Weighted Mean Reversion	cwmr
Pattern-Matching	Robust Median Reversion	rnr
	Nonparametric Histogram Log-optimal	bk
	Nonparametric Nearest Neighbor Log-optimal	bnn
Others	Correlation-driven Nonparametric Learning	cornk
	Constant Rebalanced Portfolio	m0

Table 2.1: General classification for the state-of-the-art online portfolio selection algorithms.

2.1.2 Cryptocurrencies

The original paper tests the framework in the cryptocurrency market. A cryptocurrency is a digital asset designed to work as a medium of exchange that uses strong cryptography to secure financial transactions, control the creation of additional units, and verify the transfer of assets (Narayanan et al.; 2016; Chohan; 2017; Schueffel; 2017). Cryptocurrencies use decentralized control as opposed to centralized digital currency and central banking systems (Szabo; 2015).

Bitcoin, first released as open-source software in 2009, is generally considered the first decentralized cryptocurrency. Since the release of bitcoin, over 6,000 altcoins, which are alternative variants of bitcoin, or other cryptocurrencies, have been created (Sagona-Stophel; 2015). The motivation of inventing so many variants is that there are flaws in Bitcoin and those invariants are invented to overcome these defects hoping their inventions will eventually replace Bitcoin. However, as of December 2019, Bitcoin takes 66.6% of the total market capital of all cryptocurrencies, which is nearly 198 billions in USD¹. Therefore, Bitcoin is

¹Crypto-currency market capitalizations, <http://coinmarketcap.com/>, accessed: 2019-12-14.

still the dominant cryptocurrency in the market and many other currencies can only be traded against Bitcoin. As a result, the original paper uses Bitcoin to be the unit when measuring the price of each cryptocurrency.

2.1.3 Reinforcement Learning

Reinforcement learning is an area of machine learning concerned with how software agents ought to take actions in an environment in order to maximize some notion of cumulative reward. Reinforcement learning is one of three basic machine learning paradigms, alongside supervised learning and unsupervised learning.

It differs from supervised learning in not needing labelled input/output pairs to be presented, and in not needing sub-optimal actions to be explicitly corrected (Kaelbling et al.; 1996). Instead the focus is on finding a balance between exploration (of uncharted territory) and exploitation (of current knowledge).

Reinforcement learning involves an agent and an environment: every time step, the agent chooses an action, and the environment will return a reward and transitions into the next state. A typical framework is shown in Figure 2.1. The agent's interaction with the environment is broken up into a series of episodes, each consisting of one or more time steps/actions. The process continues until a terminal state is reached, at which point the episode ends.

2.2 Related Work

Many machine learning approaches have been proposed for financial market trading. Most of them focus on stock price prediction or trends prediction, see Heaton et al. (2017); Niaki and Hoseinzade (2013); Freitas et al. (2009). The neural networks take the historical prices or indicators as input and then output the stock price or trend for the next period. Depending on the prediction, certain trading action will be taken, such as buy, sell or hold.

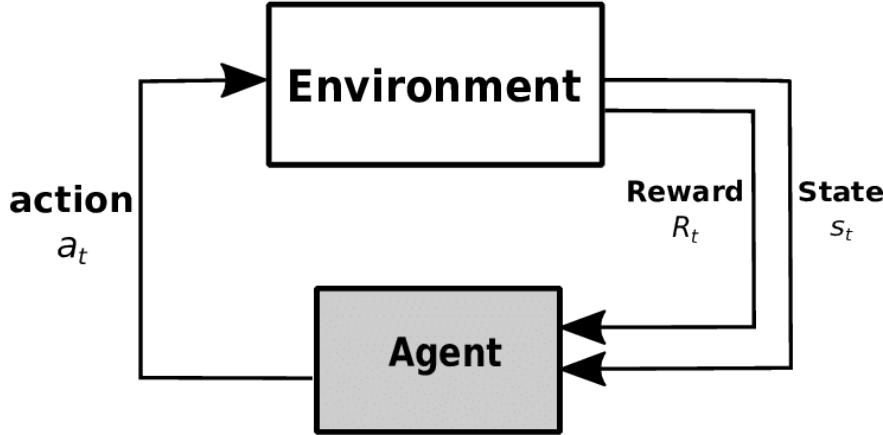


Figure 2.1: The Typical Framing of a Reinforcement Learning (RL) scenario

Those methods are straightforward and most are supervised learning, or essentially regression problems. The portfolio performance largely depends on the prediction accuracy, but it turns out the future market prices are difficult to predict. Moreover, there is still a layer between the neural network's output and trading action, which needs extra logic that can be based on human decisions. In this case, the whole approach is not purely machine learning and thus not very extendable or adaptable.

Using Reinforcement learning is a successful attempt when trying to apply model-free and fully machine learning schemes to algorithmic trading problems, without predicting future prices. Related work includes: Moody and Saffell (2001); Dempster and Leemans (2006); Cumming et al. (2015); Deng et al. (2016). However, those algorithms consider only the one asset case and are not applicable when it comes to portfolio management problems, where trading agents manage multiple assets.

One major problem with many reinforcement learning problem is the discrete action spaces that cannot be applied directly to the portfolio management problem, where actions are continuous. The market actions can be discretized but this comes with unknown risk. For example, the agent may invest all the capital in one asset, without spreading the risk to the rest of the market. As a result, instead of discrete action spaces, the agent in the original paper outputs a vector of portfolio weights as next action which is continuous.

Chapter 3

Problem Definition

This chapter defines the mathematical setting of the portfolio management problem. We first define the time discretization, then formulate the problem using mathematical symbols. Afterwards, transaction costs will be introduced as well as the two assumptions we imposed on our framework.

3.1 Two Assumptions

The validation of the framework is based on two assumptions.

1. Zero slippage: we assume that the volume of the market is high enough that each order can be filled with the market price at placement.
2. Zero money impact: the capital of the portfolio is small enough that it has no influence on the market.

These are the reason that later we select cryptocurrencies/stocks with high trading volume, in which case the above two assumptions that the framework is based on will be satisfied.

3.2 Mathematical Formalism

The whole trading process is divided into periods of equal length T , which is set to 30 mins in our later experiments. Shown in Figure 3.1, period t and $t+1$ are two consecutive periods. In each period, the price movement can be described using four features, namely opening price, closing price, highest price and lowest price. By the zero slippage assumption in Section 3.1, the closing price of period t is the opening price of period $t+1$.

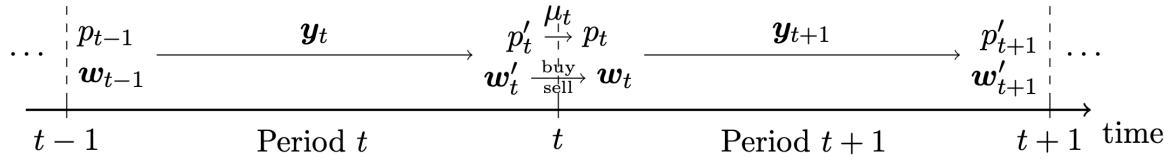


Figure 3.1: Illustration of trading between periods.

Considering a portfolio of m assets, we define the *price vector* as follows:

- $\mathbf{v}_t = (v_{0,t}, v_{1,t}, \dots, v_{m,t})$ is the closing prices of the t th period.
- $\mathbf{v}_t^{(hi)} = (v_{0,t}^{(hi)}, v_{1,t}^{(hi)}, \dots, v_{m,t}^{(hi)})$ is the highest prices of the t th period.
- $\mathbf{v}_t^{(lo)} = (v_{0,t}^{(lo)}, v_{1,t}^{(lo)}, \dots, v_{m,t}^{(lo)})$ is the lowest prices of the t th period.

We use Bitcoin as the cash and all other prices are quoted in cash, therefore the first element of each price vector will always be 1, that is $v_{0,t} = v_{0,t}^{(hi)} = v_{1,t}^{(lo)} = 1, \forall t$.

We then define the *relative price vector* of the t th trading period y_t as:

$$\mathbf{y}_t := \mathbf{v}_t \oslash \mathbf{v}_{t-1} \quad (3.1)$$

which is the element-wise division of \mathbf{v}_t by \mathbf{v}_{t-1} . The elements of y_t are the quotients of closing prices and opening prices for individual asset in the period. The relative price vector can be used to calculate the change in total portfolio value in a period.

Let p_{t-1} denote the portfolio value at the beginning of period t , then we have,

$$p_t = p_{t-1} \mathbf{y}_t \cdot \mathbf{w}_{t-1} \quad (3.2)$$

where \mathbf{w}_{t-1} is the portfolio weight vector (hereinafter referred to as the *portfolio vector*) at the beginning of period t . By definition, all elements in the *portfolio vector* sum up to 1. Then the *rate of return* for period t is

$$\rho_t := \frac{p_t}{p_{t-1}} - 1 = \mathbf{y}_t \cdot \mathbf{w}_{t-1} - 1 \quad (3.3)$$

and the corresponding *corresponding logarithmic rate of return* is

$$r_t := \ln \frac{p_t}{p_{t-1}} = \ln \mathbf{y}_t \cdot \mathbf{w}_{t-1} \quad (3.4)$$

When there is no transaction cost, the final portfolio value will be

$$p_f = p_0 \exp \sum_{t=1}^{t_f+1} r_t = p_0 \prod_{t=1}^{t_f+1} \mathbf{y}_t \cdot \mathbf{w}_{t-1} \quad (3.5)$$

where p_0 is the initial investment amount. The goal of our framework is to maximize the final portfolio value for a given time frame.

3.3 Transaction Costs

In the real world financial market, transaction costs are not immaterial and may affect the portfolio significantly. Many strategies may have a great performance without transaction costs, but become terrible when they are included. Therefore, it is crucial for us to include transaction costs into our framework. In our framework, we assume that the costs mainly come from commission fee and we assume a constant commission rate using the recursive formula by Ormos and Urbán (2013).

The portfolio vector at the end of period t and before any action is taken is denoted by \mathbf{w}'_t . Due to the price move in the period, the weights \mathbf{w}'_t evolve into

$$\mathbf{w}'_t = \frac{\mathbf{y}_t \odot \mathbf{w}_{t-1}}{\mathbf{y}_t \cdot \mathbf{w}_{t-1}} \quad (3.6)$$

where \odot is the element-wise multiplication. The task for the portfolio manager is to redistribute the whole portfolio such that the initial weights at the beginning of period $t+1$ are changed from \mathbf{w}'_t to \mathbf{w}_t . As a result, a series of trading actions will take place at time $t+1$ as well as the involvement of commission fees. Suppose such redistribution action will shrink the portfolio value by a *transaction remainder factor* μ_t where $\mu_t \in (0, 1]$, the factor can be determined by

$$\mu_t = \frac{p_t}{p'_t} \quad (3.7)$$

where p'_t is the portfolio value at the end of period t before any action is taken and p_t at the beginning of period $t+1$ after transactions costs. Under this situation, (3.3) and (3.4) are now

$$\rho_t := \frac{p_t}{p_{t-1}} - 1 = \frac{\mu_t p'_t}{p_{t-1}} - 1 = \mu_t \mathbf{y}_t \cdot \mathbf{w}_{t-1} - 1 \quad (3.8)$$

$$r_t := \ln \frac{p_t}{p_{t-1}} = \ln \mu_t \mathbf{y}_t \cdot \mathbf{w}_{t-1} \quad (3.9)$$

and the final portfolio value becomes

$$p_f = p_0 \exp \sum_{t=1}^{t_f+1} r_t = p_0 \prod_{t=1}^{t_f+1} \mu_t \mathbf{y}_t \cdot \mathbf{w}_{t-1} \quad (3.10)$$

Once again, a graphical representation of the relationship among portfolio vectors can be found in Figure 3.1.

Let $c_s, c_p \in [0, 1)$ denote the commission rate for selling and purchasing respectively, μ_t

can be calculated by

$$\mu_t = \frac{1}{1 - c_p w_{t,0}} \left[1 - c_p w'_{t,0} - (c_s + c_p - c_s c_p) \sum_{i=1}^m (w'_{t,i} - \mu_t w_{t,i})^+ \right] \quad (3.11)$$

The proof of (3.11) will be given in Appendix A.

Chapter 4

Data Structure

Data is an important consideration before feeding into the neural network framework. Garbage in, garbage out. So we need a careful analysis on the treatment of data. In this chapter, we will go through how we select the assets and how the data are stored in the price tensor.

4.1 Asset Selection

In the original paper, the top 11 non-cash assets from the cryptocurrency market ranking by trading volume are selected to construct the portfolio. Together with the cash, Bitcoin, the size of the portfolio, $m + 1$, is 12. As for the stock market experiment, the top 11 stocks are selected according to their volume as well.

The reasons for the above selection is as follows:

1. Bigger volume implies better market liquidity of an asset, in turn meaning that our experiment satisfies the Assumption 1 in Section 3.1. It also suggests that the portfolio is less likely to have influence on the market, making the environment close fulfilling the Assumption 2.
2. The market of cryptocurrency is not stable. Some cryptocurrencies may suddenly jump or drop in volume in a short time. Therefore, the volume is measured on a longer

time-frame, where in the later experiments, 30-day average volumes are used.

The whole time frame is split into training sets and test sets. The former is used for offline training of the policy network while the later is used for back-testing as well as online training. Choosing the top volume assets just before the end of the whole time frame can be subject to the survivor bias. The trading volume of an asset is correlated to its popularity, which in turn is governed by its historic performance. Given the future ranking by volume will inevitably pass future price information to the experiments. In order to avoid this bias as much as possible, the average volume is taken just before the beginning of the back-tests.

4.2 Price Tensor

The data is fed into the neural network in the form of price tensors. The neural network will then generate the output of portfolio vectors. We will talk about the structure of the price tensor and its normalization scheme.

The price tensor \mathbf{X}_t is the input to the neural networks at the end of period t , which has a dimension of (f, n, m) where m is the number of preselected non-cash assets, n is the number of input periods before t , and f is the feature number. The features are chosen to be the closing price, highest price and lowest price in a period. Using the notation in Section 3.2, these are $v_{i,t}$, $v_{i,t}^{(hi)}$, $v_{i,t}^{(lo)}$ for asset i on period t .

An important step before directly feeding the data into the networks is normalization. The absolute price is normalized by the latest closing price in order to reflect the changes in prices which actually determine the performance of the portfolio management. The price tensor \mathbf{X}_t is defined as

$$\mathbf{X}_T = (\mathbf{V}_t, \mathbf{V}_t^{(hi)}, \mathbf{V}_t^{(lo)}) \quad (4.1)$$

where $\mathbf{V}_t, \mathbf{V}_t^{(hi)}, \mathbf{V}_t^{(lo)}$ are the normalized price matrices,

$$\begin{aligned}\mathbf{V}_t &= [\mathbf{v}_{t-n+1} \oslash \mathbf{v}_t, \mathbf{v}_{t-n+2} \oslash \mathbf{v}_t, \dots, \mathbf{v}_{t-1} \oslash \mathbf{v}_t, \mathbf{1}] \\ \mathbf{V}_t^{(hi)} &= [\mathbf{v}_{t-n+1}^{(hi)} \oslash \mathbf{v}_t, \mathbf{v}_{t-n+2}^{(hi)} \oslash \mathbf{v}_t, \dots, \mathbf{v}_{t-1}^{(hi)} \oslash \mathbf{v}_t, \mathbf{v}_t^{(hi)} \oslash \mathbf{v}_t] \\ \mathbf{V}_t^{(lo)} &= [\mathbf{v}_{t-n+1}^{(lo)} \oslash \mathbf{v}_t, \mathbf{v}_{t-n+2}^{(lo)} \oslash \mathbf{v}_t, \dots, \mathbf{v}_{t-1}^{(lo)} \oslash \mathbf{v}_t, \mathbf{v}_t^{(lo)} \oslash \mathbf{v}_t]\end{aligned}$$

with $\mathbf{1} = (1, \dots, 1)^T$ and \oslash being the element-wise division operator.

At the end of period t , the portfolio manager comes up with a portfolio vector w_t using merely the information from the price tensor \mathbf{X}_t and the previous portfolio vector w_{t-1} , according to some policy π . In other words, $w_t = \pi(\mathbf{X}_t, w_{t-1})$. At the end of period $t+1$, the logarithmic rate of return for the period due to decision w_t can be calculated with the additional information from the price change vector y_{t+1} , using Equation (3.9). In the language of reinforcement learning, r_{t+1} is the immediate reward to the portfolio management agent for its action w_t under the environment \mathbf{X}_t .

Chapter 5

Reinforcement Learning Framework

Extending what we have discussed in Subsection 2.1.3, we will investigate into how the original paper builds up the whole reinforcement learning framework. First, we will talk about how the portfolio problem defined before can be applied to reinforcement learning, specifically explains the representation of each element. Then, we will look into the policy network which is the core innovation from the original paper. Three different deep neural networks are used here to construct the policy function, namely CNN, RNN and LSTM. Three innovations will also be discussed, namely the network topologies, the portfolio-vector memory and the online stochastic batch learning scheme.

5.1 Reinforcement Learning Elements

As shown in Figure 2.1, several elements are crucial to the framing of a reinforcement learning setup, which will be explained in detail.

The Agent and the Environment

The agent in this portfolio management problem is the software portfolio manager performing trading-actions in the environment of a financial market. This environment is comprised of all available assets in the market and the expectations of all market participants towards

them.

The State and the Action

In theory, the state at time t should be all the information up until t . However, it is impossible to get and store all such information considering such a large and complex environment. It is believed that all relevant information is reflected in the prices of the assets, which are already publicly available to the agent (Kirkpatrick II and Dahlquist; 2010; Lo et al.; 2000). Under this point of view, sub-sampling schemes for the order-history information are employed to be the state representation of the market environment. Besides, the action made at the beginning of period t will affect the reward of period $t + 1$, and as a result will affect the decision of its action. As a result, the state at t is represented as the pair of \mathbf{X}_t and w_{t-1} ,

$$\mathbf{s}_t = (\mathbf{X}_t, w_{t-1}) \quad (5.1)$$

The action a_t at time t is therefore

$$\mathbf{a}_t = \mathbf{w}_t \quad (5.2)$$

where $\mathbf{w}_0 = (1, 0, \dots, 0)^T$.

The Reward Function

The objective of the agent is to maximize the portfolio value p_f of Equation (3.10) at the end of $t_f + 1$ period. Or equivalently, the job is to maximize the average logarithmic accumulated return R ,

$$R(\mathbf{s}_1, \mathbf{a}_1, \dots, \mathbf{s}_{t_f}, \mathbf{a}_{t_f}, \mathbf{s}_{t_f+1}) = \frac{1}{t_f} \ln \frac{p_f}{p_0} = \frac{1}{t_f} \sum_{t=1}^{t_f+1} \ln \mu_t \mathbf{y}_t \cdot \mathbf{w}_{t-1} = \frac{1}{t_f} \sum_{t=1}^{t_f+1} r_t \quad (5.3)$$

Therefore, R is the accumulated reward, and r_t/t_f is the immediate reward for an individual period.

Deterministic Policy Gradient

The policy π is the strategy that the agent employs to determine the next action based on the current state. It maps states to actions, the actions that promise the highest reward. Mathematically, the policy can be represented by $\pi: S \rightarrow A$. By full exploitation, the policy can deterministically produce an action from a state. The optimal policy is obtained using a gradient ascent algorithm. To achieve this, a policy is specified by a set of parameters θ , and $a_t = \pi_\theta(s_t)$. The performance matrix of π_θ for the time interval $[0, t_f]$ is defined as the corresponding reward function of this interval,

$$J_{[0,t_f]}(\pi_\theta) = R(s_1, \pi_\theta(s_1), \dots, s_{t_f}, \pi_\theta(s_f), s_{t_f+1}) \quad (5.4)$$

The initial parameter is initialized by randomization, and will be continuously updated along the gradient direction with a learning rate λ ,

$$\theta \rightarrow \theta + \lambda \nabla_\theta J_{[0,t_f]}(\pi_\theta) \quad (5.5)$$

As a usual practice in reinforcement learning, the policy is updated upon mini-batches instead of the whole time frame as above. The true updating rule will then be, for each mini-batch $[t_{b_1}, t_{b_2}]$, do

$$\theta \rightarrow \theta + \lambda \nabla_\theta J_{[t_{b_1}, t_{b_2}]}(\pi_\theta) \quad (5.6)$$

5.2 Policy Networks

Neural networks are function approximators, which are particularly useful in reinforcement learning when the state space or action space are too large to be completely known. Typically,

it is used to approximate the policy function in this framework.

The policy function π_θ is constructed using three different neural networks, namely CNN, LSTM and RNN. Three important innovations are included in the framework that are crucial for the completeness and effectiveness. They are network topologies, the portfolio-vector memory and the stochastic mini-batch online learning scheme.

5.2.1 Network Topologies

The implementations of policy networks using CNN, RNN and LSTM are visualized using the figure extracted from the original paper, see Figure 5.1 and Figure 5.2. For all implementations, the input is the price tensor \mathbf{X}_t as well as the weight from last period \mathbf{w}_{t-1} . The output is the portfolio vector of the next period \mathbf{w}_t .

As one can see from both figures, the last hidden layers are the voting scores for all non-cash assets. The soft-max outcomes of these scores and a cash bias become the actual corresponding portfolio weights. Please keep in mind that \mathbf{w}_{t-1} is fed into the network just before the voting layer, differing from \mathbf{X}_t being fed into the first layer.

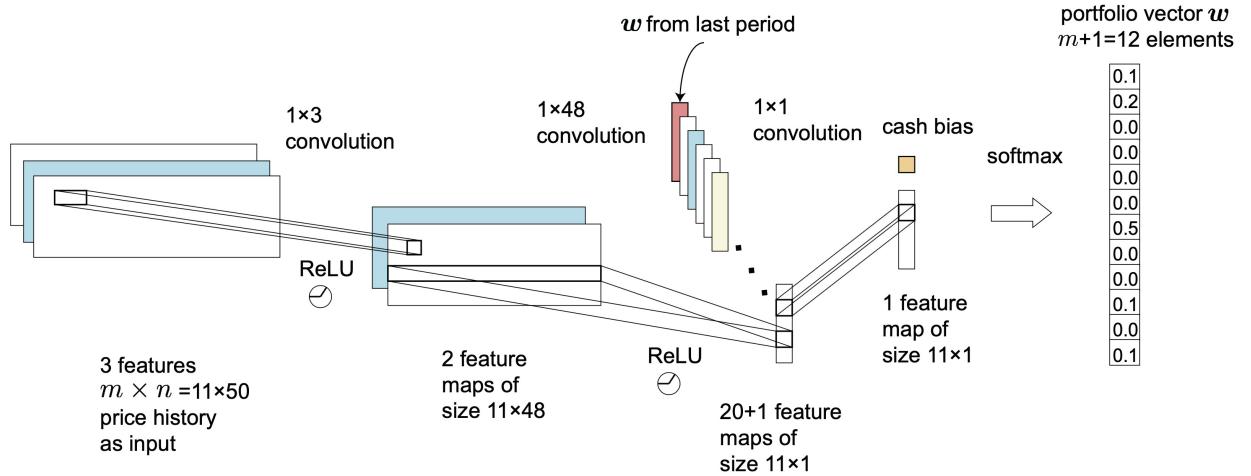


Figure 5.1: CNN Implementation of the EIIE

For the CNN implementation, The first dimensions of all the local receptive fields in all feature maps are 1, making all rows isolated from each other until the soft-max activation.

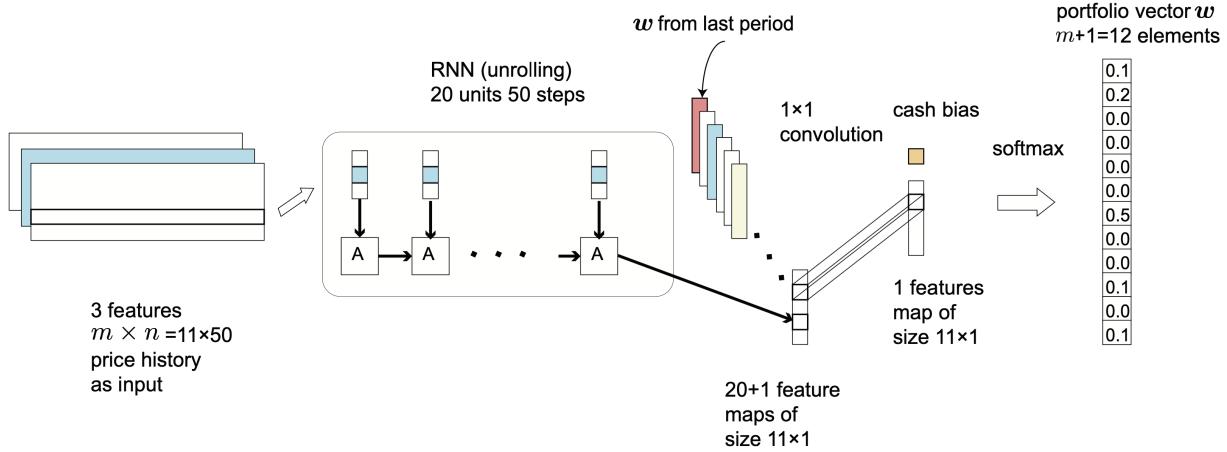


Figure 5.2: RNN (or LSTM) Implementation of the EIIE

Apart from weight-sharing among receptive fields in a feature map, which is a usual CNN characteristic, parameters are also shared between rows in an EIIE configuration. Each row of the entire network is assigned with a particular asset, and is responsible to submit a voting score to the soft-max on the growing potential of the asset in the coming trading period. The input to the network is a $(3 \times m \times n)$ price tensor, comprising the highest, closing, and lowest prices of m non-cash assets over the past n periods. The outputs are the new portfolio weights. The previous portfolio weights are inserted as an extra feature map before the scoring layer, for the agent to minimize transaction costs.

The RNN implementation is the same as that of LSTM, with the only difference being RNN replacing LSTM in the first layer. It is a recurrent realization the Ensemble of Identical Independent Evaluators (EIIE). In this version, the price inputs of individual assets are taken by small recurrent subnets. These subnets are identical LSTMs or RNNs.

An important feature in this framework is that the network will evaluate each of the m assets independently but the parameters are shared among those streams. These streams are like independent but identical networks of smaller scopes, separately observing and assessing individual non-cash assets. They only interconnect at the softmax function, just to make sure their outputting weights are non-negative and summing up to unity. This special architecture of Identical Independent Evaluators (IIE) ensembled together to build up the mega topology

is the reason that the authors named it as EIIE.

5.2.2 Portfolio-Vector Memory

Inspired by the idea of experience replay memory (Mnih et al.; 2016), the author introduced a dedicated Portfolio-Vector Memory (PVM) to store the network outputs, i.e. the weights in each step. The PVM is a stack of portfolios in chronological order. Before any network training, the PVM is initialized with uniform weights. In each training step, a policy network loads the portfolio vector of the previous period from the memory location at $t - 1$, and overwrites the memory at t with its output. As the parameters of the policy networks converge through many training epochs, the values in the memory also converge.

The advantages that PVM brings to the framework are significant:

1. PVM allows the network to be trained simultaneously against data points within mini-batches, enormously improving training efficiency.
2. In the case of RNN versions of the networks, inserting previous outputs after the recurrent blocks avoids passing the gradients back to the deep RNN structures, circumventing the gradient vanishing problem.

5.2.3 Online Stochastic Batch Learning

When this framework is applied to real-time trading, it is essential for the agent to update its policy when new data comes. The ever-ongoing nature of financial markets motivates the inclusion of online training. In this framework, the author proposed the so called Online Stochastic Batch Learning (OSBL) scheme.

At time t , the price movement information of period t is available and added to the training set. After the agent takes trading orders at the beginning of period $t + 1$, the policy network will be trained against N_b randomly chosen mini-batches from this set. Each batch will be starting from t_b with size n_b . The starting period is chosen from a geometric

distribution, i.e. $t_b \sim GEO(\beta)$ or the probability $P_\beta(t_b)$ is,

$$P_\beta(t_b) = \beta(1 - \beta)^{t-t_b-n_b}, \quad t_b \leq t - n_b \quad (5.7)$$

where $\beta \in (0, 1]$ is the decay rate reflecting your beliefs on the importance of recent market events and n_b is the number of periods in each mini-batch.

Chapter 6

Numerical Experiments and Evaluation

The deep reinforcement learning framework is examined and evaluated with two experiments. One is in the cryptocurrency market as it is in the original paper, the other one is in the stock market. We use a dataset different from the original paper in the first experiment and then totally change the asset to be selected from the equity market in the second experiment.

We first explain the performance measures that are used to evaluate the performance of a particular portfolio selection strategy. Then the results of the two experiments are discussed.

Performance Measures

The following measures and factors will be used in our later discussions:

1. Final Accumulated Portfolio Value (fAPV): the accumulated portfolio value over the whole time span of the back-test, with the initial value being 1.
2. Sharpe ratio: risk adjusted mean return, defined as the difference between the return of the investment and the risk-free return, divided by the standard deviation of the investment..

$$SP = \frac{\mathbb{E}_t[\rho_t - \rho_F]}{\sqrt{\text{Var}_t(\rho_t - \rho_F)}}$$

where ρ_t are the periodic returns defined in Equation (3.8) and ρ_F is the the rate of

return of a risk-free asset.

3. Maximum Draw-down (MDD): the biggest loss from a peak to a trough, mathematically

$$D = \max_t \left(\max_{\tau > t} \frac{p_t - p_\tau}{p_t} \right)$$

where $t \in (0, T]$

6.1 In the Cryptocurrency Market

Below is the time range for **Experiment 1**. All times below are in Coordinated Universal Time (UTC). All training sets start at 0 o'clock. All price data are accessed through Poloniex's official Application Programming Interface (API)¹. Back-testing time range is the same as the testing time range, the only difference is that the former is online after training is complete and the latter is offline during the training process.

- **Experiment 1** Time Range: 2015-07-01 to 2017-07-01
 - Training Time Range: 2015-07-01 to 2017-05-03
 - Best-testing (Testing) Time Range: 2017-05-04 to 2017-07-01

More configurations for **Experiment 1** can be found in Appendix B.1, which lists all the hyper-parameters' values.

Figure 6.1 plots the APV against time in the back-test respectively for the CNN, RNN and LSTM EIIE networks, two selected benchmarks and two model-based strategies. The benchmarks Best Stock and UCRP are two good representatives of the market. We can see all EIIEs beat the market throughout the entirety of the back-tests.

Table 6.1 is the internal comparison between the three different EIIEs with different neural networks. One can observe a great reduction in log mean when including the transaction

¹<https://poloniex.com/support/api/>

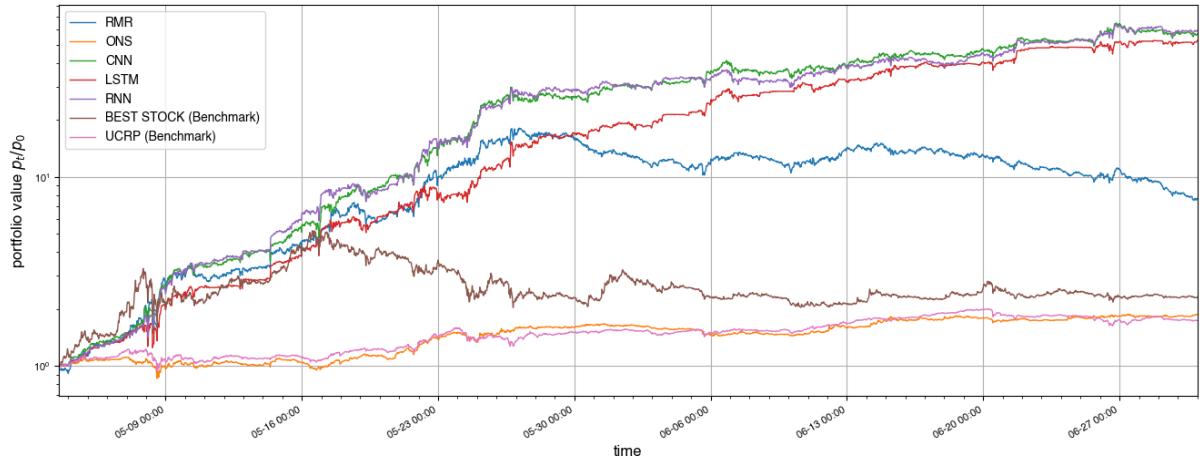


Figure 6.1: Log-scale accumulated wealth from 2017-05-04 to 2017-07-01 - *Back-test 1*

Network	Test			Backtest			Training Time(s)
	fAPV	Log Mean	Log Mean Free	fAPV	Log Mean		
CNN	8.938228607	0.000789027	0.003126783	56.98793309	0.001456354		590
RNN	25.48135948	0.001166407	0.002652416	59.49014754	0.001471834		1203
LSTM	42.89720917	0.001354037	0.002863562	52.48585405	0.001426709		3751

Table 6.1: Training and Back-testing results for CNN, RNN and LSTM - *Back-test 1*

costs, which restates the importance of considering transaction costs in the real financial world. For all three networks, the test fAPV is less than that in back-testing, indicating the power of online learning. Among them, LSTM performs the best in terms of testing fAPV, while RNN is best with respect to back-testing fAPV. The out-performance of RNN over CNN and LSTM in back-testing is not significant though.

Algorithms	MDD	fAPV	SR	- Days	- Periods	- Weeks	+ Days	+ Periods	+ Weeks
CNN	0.279	56.988	0.084	659	1,379	13	2,117	1,397	2,763
RNN	0.232	59.490	0.088	679	1,373	17	2,097	1,403	2,759
LSTM	0.255	52.486	0.083	514	1,551	10	2,262	1,187	2,766
Best Stock	0.608	2.310	0.025	1,403	1,432	1,163	1,373	1,328	1,613
UCRP	0.234	1.739	0.031	1,055	1,264	650	1,721	1,512	2,126
UBAH	0.268	1.457	0.020	1,119	1,282	890	1,657	1,494	1,886
ANTICOR	0.187	4.892	0.054	870	1,327	409	1,906	1,449	2,367
OLMAR	0.608	4.320	0.036	1,342	1,451	1,220	1,434	1,319	1,556
PAMR	0.980	0.041	-0.048	2,018	1,589	2,030	758	1,186	746
WMAMR	0.563	1.821	0.021	1,409	1,486	1,166	1,367	1,287	1,610
CWMR	0.986	0.028	-0.054	2,049	1,607	2,051	727	1,168	725
RMR	0.584	7.676	0.046	1,233	1,451	1,027	1,543	1,318	1,749
ONS	0.229	1.864	0.035	1,133	1,366	837	1,643	1,410	1,939
UP	0.235	1.718	0.031	1,068	1,267	649	1,708	1,509	2,127
EG	0.235	1.726	0.031	1,065	1,269	648	1,711	1,507	2,128
BK	0.782	0.574	-0.003	1,701	1,403	1,830	1,075	1,373	946
CORNK	0.978	0.027	-0.085	2,359	1,573	2,729	417	1,203	47
M0	0.289	1.890	0.028	1,156	1,288	639	1,620	1,488	2,137

Table 6.2: Performance of the three EIIEs, compared with traditional online selection algorithms - *Back-test 1*

As one can see from Table 6.2, all three EIIEs outperform the other algorithms in terms of fAPV and SR. The Anticor algorithm has the lowest MDD. By including a fixed commission rate, many traditional algorithms perform terribly with fAPV even less than the initial value 1. We also include the counting of negative and positive periods, days and weeks, whose result shows the continuous profitability of the three EIIEs. The only situation when the reinforcement learning algorithms lose is when counting negative/positive periods but the scoring gap is small.

6.2 In the Stock Market

In **Experiment 2**, the asset is selected from the stock market. Data is extracted through the Kibot's official Application Programming Interface (API)². The time range is set to be the past two years.

- **Experiment 2** Time Range: 2017-12-12 to 2019-12-11
 - Training Time Range: 2017-12-12 to 2019-10-13
 - Back-testing (Testing) Time Range: 2019-10-14 to 2019-12-11

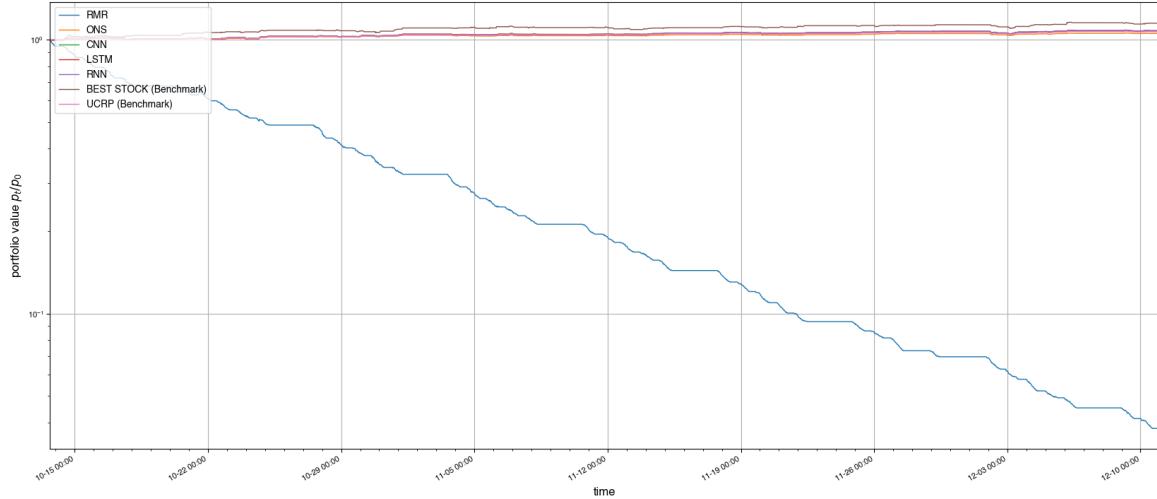


Figure 6.2: Log-scale accumulated wealth from 2019-10-14 to 2019-12-11 - *Back-test 2*

More configurations for **Experiment 2** can be found in Appendix B.2, which lists all the hyper-parameters' values.

Figure 6.2 and Figure 6.3 plot the APV against time in the three back-tests respectively for the CNN, RNN and LSTM EIIE networks, two selected benchmarks and two model-based strategies. Figure 6.2 includes a type of Follow-the-Loser algorithm (RMR) that performs terrible in this case. Other Follow-the-Loser algorithms are also tested and similar outcomes

²http://www.kibot.com/api/historical_data_api_sdk.aspx

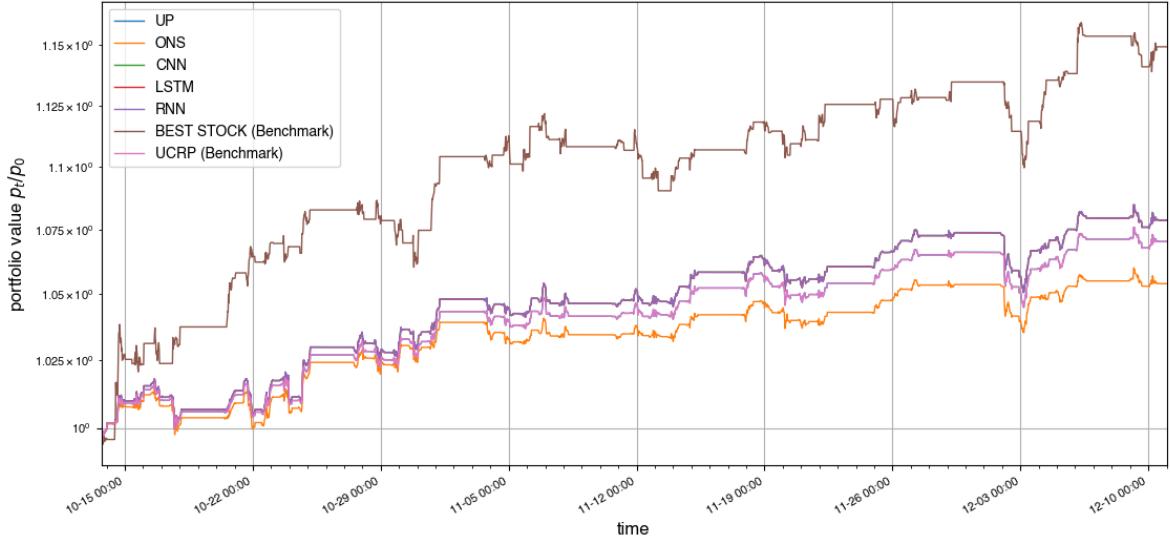


Figure 6.3: Log-scale accumulated wealth (excluding Follow-the-Loser) from 2019-10-14 to 2019-12-11 - *Back-test 2*

are achieved, hence omitted plotting. As we know, Follow-the-Loser algorithm assumes that the under-performing assets will revert and outperform others in the subsequent periods. The terrible performance by the Follow-the-Loser algorithms may indicate the out-performing stocks remain strong in the time range and the under-performing stocks show less reversion behavior. This explanation is also validated by Figure 6.3, where the Best Stock algorithm performs the best and outperforms all others significantly. We know that the Best Stock algorithm is a special Buy and Hold algorithm that puts all capital on the stock with best performance in hindsight. The bet on that best stock being always the winner wins this time. This leaves the room for future developments. We may increase the testing set portion or adjust the decay rate β to assign more importance to recent market event. Besides, we need more investigations in the stock selection process before training, as it is purely based on the ranking of the average trading volume.

Table 6.3 is the internal comparison between the three different EIIEs with different neural networks. Interesting things happen here. We notice that the performance measures in the table are close to each other among the three networks, except the training time which

Network	Test			Backtest		Training Time(s)
	fAPV	Log Mean	Log Mean Free	fAPV	Log Mean	
CNN	1.081315756	2.82504E-05	2.85495E-05	1.078906536	2.74374E-05	613
LSTM	1.081547618	2.83314E-05	2.86326E-05	1.078754127	2.73863E-05	4629
RNN	1.081599355	2.83488E-05	2.86426E-05	1.078838147	2.74136E-05	1905

Table 6.3: Training and Back-testing results for CNN, RNN and LSTM - *Back-test 2*

is determined by the nature of the underlying network. The log mean doesn't decrease as much as that in Table 6.1. Upon further investigation into the back-testing, we find that a nearly uniform portfolio is kept in the whole back-testing process. The weight for each non-cash asset is about $\frac{1}{m}$. In this case, all three EIIEs perform similarly and can be seen as the Uniform Constant Rebalanced Portfolio (UCRP), which is validated by the nearly same trends in Figure 6.3. The minor influence of transaction costs on log mean can also be explained by the less frequent rebalancing. This result sheds light into what we have learnt in ActSc 972, where we have discussed about how efficient the $\frac{1}{N}$ strategy is. DeMiguel et al. (2007) compared the performance of many different portfolio selection methods to the equal asset weights. They found the equally weighted strategy outperformed the other strategies out-of-sample on a variety of financial datasets. Even this naive algorithm is not the best but the Best Stock algorithm, it still ranks top among other algorithms. The reinforcement learning process gradually makes our agent follow this equally weighted strategy.

Table 6.4, once again, validates our explanation above. The three EIIEs perform similarly to UCRP and even a little bit better in terms of fAPV. Even though they are not as good as the Best Stock algorithm, the gap in between is tiny. Follow-the-Loser algorithms have the worst performance of all³.

³some of the Follow-the-Loser algorithms can not been implemented because the Σ matrix is singular

Algorithms	MDD	fAPV	SR	- Days	- Periods	- Weeks	+ Days	+ Periods	+ Weeks
CNN	0.023	1.079	0.042	659	1,387	1,645	1,381	703	2,364
RNN	0.023	1.079	0.042	679	1,388	1,690	1,380	690	2,364
LSTM	0.023	1.079	0.042	514	1,388	1,784	1,380	682	2,364
Best Stock	0.031	1.149	0.042	1,403	851	319	1,278	367	2,349
UCRP	0.021	1.070	0.041	1,055	854	649	1,379	663	2,343
UBAH	0.021	1.072	0.042	1,119	840	578	1,384	663	2,345
ANTICOR	0.442	0.560	-0.243	870	2,345	1,674	105	363	10
OLMAR	0.960	0.040	-0.443	1,342	2,221	837	4	216	0
PAMR	0.956	0.044	-0.515	2,018	2,224	1,110	5	132	0
WMAMR	0.673	0.326	-0.278	1,409	2,238	1,105	15	273	0
ONS	0.019	1.054	0.033	1,133	906	674	1,327	640	2,222
UP	0.021	1.071	0.041	1,068	854	649	1,379	663	2,343
EG	0.021	1.070	0.041	1,065	854	649	1,379	663	2,343
M0	0.013	1.021	0.023	1,156	1,396	2,125	1,372	643	1,950

Table 6.4: Performance of the three EIIEs, compared with traditional online selection algorithms - *Back-test 2*

Chapter 7

Conclusion and Future Work

The original paper proposed a deep reinforcement learning framework to solve the financial portfolio management problem. The core architecture is the EIIE meta topology, which is able to accommodate many types of weight-sharing neural network structures in the lower level. Additional innovations include the use of Portfolio Vector Memory (PVM) and the implementation of the Online Stochastic Batch Learning (OSBL) scheme. The former stores portfolio vectors in a separate memory such that transaction costs can be taken into account, the agent can avoid oversized reallocations between consecutive actions and the framework can be trained in parallel within batching. The latter governs the online learning process so that the agent can continuously digest constant incoming market information while trading.

The framework is implemented, trained and back-tested in two different markets, i.e. the cryptocurrency market and the stock market. In all experiments, the framework was realized using three different underlining networks, a CNN, a RNN and a LSTM. The profitability of all surpasses traditional portfolio-selection methods, as demonstrated in the paper by the outcomes of **Experiment 1**. Among the three EIIE networks, LSTM has lower scores than that of CNN and RNN. The gap in performance between the two RNN species under the same framework might be explained by the belief that history repeats itself. Not being designed to forget its input history, a vanilla RNN is more able than a LSTM to exploit

repetitive patterns in price movements for higher yields. The gap might also be due to lack of fine-tuning in hyper-parameters for the LSTM. In the experiments, the same set of structural hyper-parameters was used for both basic RNN and LSTM.

Surprisingly, when it comes to the stock market, the magic of this framework disappeared and fell back to be equivalent to the Uniformly Constant Rebalanced Portfolios algorithm or the equally weighted strategy. The reinforcement learning gradually learns to sell high and buy low in order to take advantage of any mean reversion, even though this reversion is not significant in back-testing.

The future work that can be done to further examine or improve the framework is listed as follows:

1. Modify the framework to deal with situations when the two assumption in Section 3.1 are violated.
2. Try other reward functions in order to include awareness of longer-term market reactions.
3. Include technical indicators in the input tensor, e.g. exponential moving average (EMA), volume weighted average price (VWAP), relative strength index (RSI), etc.
4. Hyper-parameter tuning for stock marketS to see if the performance can be improved.
5. Select assets more carefully rather than just depending on the average trading volume.

Bibliography

- Chohan, U. W. (2017). Cryptocurrencies: a brief thematic review.
- Cumming, J., Alrajeh, D. D. and Dickens, L. (2015). *An investigation into the use of reinforcement learning techniques within the algorithmic trading domain*, PhD thesis, Master's thesis, Imperial College London, United Kingdoms, 2015.
- DeMiguel, V., Garlappi, L. and Uppal, R. (2007). Optimal versus naive diversification: How inefficient is the $1/n$ portfolio strategy?, *The Review of Financial Studies* **22**(5): 1915–1953.
- Dempster, M. A. and Leemans, V. (2006). An automated FX trading system using adaptive reinforcement learning, *Expert Systems with Applications* **30**(3): 543–552.
- Deng, Y., Bao, F., Kong, Y., Ren, Z. and Dai, Q. (2016). Deep direct reinforcement learning for financial signal representation and trading, *IEEE transactions on neural networks and learning systems* **28**(3): 653–664.
- Freitas, F. D., De Souza, A. F. and de Almeida, A. R. (2009). Prediction-based portfolio optimization model using neural networks, *Neurocomputing* **72**(10-12): 2155–2170.
- Hakansson, N. H. and Ziembba, W. T. (1995). Capital growth theory, *Handbooks in operations research and management science* **9**: 65–86.
- Heaton, J., Polson, N. and Witte, J. H. (2017). Deep learning for finance: Deep portfolios, *Applied Stochastic Models in Business and Industry* **33**(1): 3–12.

Jiang, Z., Xu, D. and Liang, J. (2017). A deep reinforcement learning framework for the financial portfolio management problem, *arXiv preprint arXiv:1706.10059* .

Kaelbling, L. P., Littman, M. L. and Moore, A. W. (1996). Reinforcement learning: A survey, *Journal of artificial intelligence research* **4**: 237–285.

Kelly, J. L. (1956). A new interpretation of information rate, *The Kelly Capital Growth Investment Criterion: Theory and Practice*, World Scientific, pp. 25–34.

Kirkpatrick II, C. D. and Dahlquist, J. A. (2010). *Technical analysis: The complete resource for financial market technicians*, FT press.

Li, B. and Hoi, S. C. (2014). Online portfolio selection: A survey, *ACM Computing Surveys (CSUR)* **46**(3): 35.

Li, B., Sahoo, D. and Hoi, S. C. (2016). Olps: a toolbox for on-line portfolio selection, *The Journal of Machine Learning Research* **17**(1): 1242–1246.

Lo, A. W., Mamaysky, H. and Wang, J. (2000). Foundations of technical analysis: Computational algorithms, statistical inference, and empirical implementation, *The journal of finance* **55**(4): 1705–1765.

Markowitz, H. (1952). Portfolio selection, *The journal of finance* **7**(1): 77–91.

Markowitz, H. (1959). Portfolio selection: Efficient diversification of investments, *NY: John Wiley* .

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D. and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning, *International conference on machine learning*, pp. 1928–1937.

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

Narayanan, A., Bonneau, J., Felten, E., Miller, A. and Goldfeder, S. (2016). *Bitcoin and cryptocurrency technologies: A comprehensive introduction*, Princeton University Press.

Niaki, S. T. A. and Hoseinzade, S. (2013). Forecasting S&P 500 index using artificial neural networks and design of experiments, *Journal of Industrial Engineering International* **9**(1): 1.

Ormos, M. and Urbán, A. (2013). Performance analysis of log-optimal portfolio strategies with transaction costs, *Quantitative Finance* **13**(10): 1587–1597.

Sagona-Stophel, K. (2015). Bitcoin 101 white paper, *PDF*. Thomson Reuters. Retrieved **20**.

Schueffel, P. (2017). The concise fintech compendium, *Fribourg: School of Management Fribourg, Switzerland* **3**.

Szabo, A. I. N. (2015). If banks want benefits of blockchains they must go permissionless, *International Business Times* **9**.

APPENDICES

Appendix A

Proofs of Theorems

A.1 Proof of the Formula for Transaction Remainder Factor

During the portfolio reallocation from \mathbf{w}'_t to \mathbf{w}_t , some or all of the amount of asset i needs to be sold, if $p'_t w'_{t,i} > p_t w_{t,i}$, or $w'_{t,i} > \mu_t w_{t,i}$. The total amount of cash obtained by all selling is

$$(1 - c_s) p'_t \sum_{i=1}^m (w'_{t,i} - \mu_t w_{t,i})^+ \quad (\text{A.1})$$

where $0 \leq c_s < 1$ is the commission rate for selling, and $\mathbf{v}^+ = \text{ReLU}(\mathbf{v})$ is the element-wise rectified linear function. This money and the original cash reserve $p'_t w'_{t,0}$ taken away from the new reserve $\mu_t p'_t w'_{t,0}$ will be used to buy new assets,

$$(1 - c) \left[w'_{t,0} + (1 - c_s) \sum_{i=1}^m (w'_{t,i} - \mu_t w_{t,i})^+ - \mu_t w_{t,0} \right] = \sum_{i=1}^m (\mu_t w_{t,i} - w'_{t,i})^+ \quad (\text{A.2})$$

where $0 \leq c_p < 1$ is the commission rate for purchasing, and p'_t has been cancelled out on both sides. Using the identity $(a - b)^+ - (b - a)^- = a - b$, and the fact that $w'_{t,0} + \sum_{i=1}^m w'_{t,i} =$

$w_{t,0} + \sum_{i=1}^m w_{i,t}$, Equation (A.2) is simplified to

$$\mu_t = \frac{1}{1 - c_p w_{t,0}} \left[1 - c_p w'_{t,0} - (c_s + c_p - c_s c_p) \sum_{i=1}^m (w'_{t,i} - \mu_t w_{t,i})^+ \right] \quad (\text{A.3})$$

μ_t can not be solved analytically, but can be solved recursively. The following is the numerical method to solve for μ_t :

The sequence $\{\tilde{\mu}_t^{(k)}\}$ is defined as

$$\left\{ \tilde{\mu}_t^{(k)} \mid \tilde{\mu}_t^{(0)} = \mu_0 \quad \& \quad \tilde{\mu}_t^{(k)} = f(\tilde{\mu}_t^{(k-1)}) \right\} \quad (\text{A.4})$$

where $f(\mu)$ is,

$$f(\mu) := \frac{1}{1 - c_p w_{t,0}} \left[1 - c_p w'_{t,0} - (c_s + c_p - c_s c_p) \sum_{i=1}^m (w'_{t,i} - \mu w_{t,i})^+ \right] \quad (\text{A.5})$$

and μ_0 is the initial guess.

Appendix B

Configurations for the Experiments

B.1 Experiment 1 Configurations

```
1 {
2     "layers": [
3         [
4             {"filter_shape": [1, 2], "filter_number": 3, "type": "ConvLayer"},
5             {"filter_number": 10, "type": "EIIIE_Dense", "regularizer": "L2",
6                 "weight_decay": 5e-9},
7             {"type": "EIIIE_Output_WithW", "regularizer": "L2", "weight_decay": 5e-8}
8         ],
9         "training": {
10            "steps": 80000,
11            "learning_rate": 0.00028,
12            "batch_size": 109,
13            "buffer_biased": 5e-5,
```

```
13     "snap_shot":false,
14     "fast_train":true,
15     "training_method":"Adam",
16     "loss_function":"loss_function6"
17 },
18
19 "input":{
20     "window_size":31,
21     "coin_number":11,
22     "global_period":1800,
23     "feature_number":3,
24     "test_portion":0.08,
25     "online":false,
26     "start_date":"2015/07/01",
27     "end_date":"2017/07/01",
28     "volume_average_days":30,
29     "portion_reversed": false
30 },
31
32 "trading":{
33     "trading_consumption":0.0025,
34     "rolling_training_steps":85,
35     "learning_rate":0.00028,
36     "buffer_biased":5e-5
37 }
38 }
```

The above is the configuration for **Experiment 1** with CNN. LSTM (RNN: “type” change to “EIIE_RNN”) configuration only differs in the layer input as bellows:

```
1 { "layers": [
2     {
3         "dropouts": null,
4         "neuron_number": 20,
5         "type": "EIIE_LSTM"
6     },
7     {
8         "regularizer": "L2",
9         "type": "EIIE_Output_WithW",
10        "weight_decay": 5e-08
11    }
12 ]}
```

B.2 Experiment 2 Configurations

```
1 {
2     "layers": [
3         [
4             {"filter_shape": [1, 2], "filter_number": 3, "type": "ConvLayer"},
5             {"filter_number": 10, "type": "EIIE_Dense", "regularizer": "L2", "weight_decay": 5e-9},
6             {"type": "EIIE_Output_WithW", "regularizer": "L2", "weight_decay": 5e-8}
7         ],
8     ]
9 }
```

```

8   "training": {
9     "steps": 80000,
10    "learning_rate": 0.00028,
11    "batch_size": 109,
12    "buffer_biased": 5e-5,
13    "snap_shot": false,
14    "fast_train": true,
15    "training_method": "Adam",
16    "loss_function": "loss_function6"
17  },
18
19  "input": {
20    "window_size": 31,
21    "coin_number": 11,
22    "global_period": 1800,
23    "feature_number": 3,
24    "test_portion": 0.08,
25    "online": false,
26    "start_date": "2017/12/12",
27    "end_date": "2019/12/11",
28    "volume_average_days": 30,
29    "portion_reversed": false
30  },
31
32  "trading": {
33    "trading_consumption": 0.0025,
34    "rolling_training_steps": 85,

```

```
35     "learning_rate": 0.00028,  
36     "buffer_biased": 5e-5  
37   }  
38 }
```

The above is the configuration for **Experiment 2** with CNN. LSTM (RNN: “type” change to “EIIE_RNN”) configuration only differs in the layer input as follows:

```
1 {  "layers": [  
2   {  
3     "dropouts": null,  
4     "neuron_number": 20,  
5     "type": "EIIE_LSTM"  
6   },  
7   {  
8     "regularizer": "L2",  
9     "type": "EIIE_Output_WithW",  
10    "weight_decay": 5e-08  
11  }  
12 ]
```

Appendix C

GitHub Repository

The implementation for this paper is stored at the repository on GitHub, <https://github.com/jackielili19/PGPortfolio/tree/Stock>. One may find all the codes, figures, tables and data there. Below is only the user guide.

Our experiments are all trained and back-testing on MacBook Pro 2018 without using GPU.

C.1 Quickstart

1. Edit pgportfolio/net_config.json
2. Generate an agent:

```
python main.py --mode=generate --repeat=1
```

3. Download the data:

```
python main.py --mode=download_data
```

4. Train the agent:

```
python main.py --mode=train --processes=1
```

5. Compare the result with other algorithms:

```
python main.py --mode=plot --algos=crp,1 --labels=crp,nnagent  
python main.py --mode=table --algos=1,ons --labels=nntrader,ons
```

See below for details on each step.

C.2 Configuration File

`pgportfolio/net_config.json` contains all the configuration parameters. The software can be configured by modifying this file and without any changes to the code.

The parameters are classified into four categories: Network Topology, Market Data, Training and Trading.

C.3 Training the agent

In order to train the agent perform the following steps:

1. (*Optional*) Modify the configuration in `pgportfolio/net_config.json` according to your desired agent configuration.

2. From the main folder, run:

```
python main.py --mode=generate --repeat=n
```

where `n` is a positive integer indicating the number of replicas you would like to train. This will create `n` subfolders in the `train_package` folder. Each subfolder contains a copy of the `net_config.json` file. The random seed of each subfolder runs from 0 to `n-1`. Please note that agents with different random seeds can have very different performances.

3. (*Optional*) Download the data with the command:

```
python main.py --mode=download_data
```

4. Train your agents with the command:

```
python main.py --mode=train --processes=1
```

* This will start training the 'n' agents one at a time.

Do not start more than 1 processes if you want to download data online.

* '--processes=m' starts 'm' parallel training processes

* '--device=gpu' can be added if your tensorflow supports GPU.

- On GTX1080Ti you should be able to run 4-5 training processes simultaneously.

- On GTX1060 you should be able to run 2-3 training processes simultaneously.

5. Each training run is composed of 2 phases: **Training** and **Backtest**.

- During the **Training** phase, the agent is trained on the training fraction of the global data matrix. The log looks like this:

```
average time for data accessing is 0.0015480489730834962
```

```
average time for training is 0.009850282192230225
```

```
=====
```

```
step 2000
```

```
-----
```

```
the portfolio value on test set is 2.118205
```

```
log_mean is 0.00027037683
```

```
loss_value is -0.000270
```

```
log mean without commission fee is 0.000341
```

- After training is completed, the **Backtest** phase begins. This uses a rolling training window, i.e. it performs online learning in supervised learning. The log looks like this:

```
the step is 536
total assets are 4.314677 BTC
```

6. Once training and backtest are completed, you can check the result summary of the training in `train_package/train_summary.csv`
7. Tune the hyper-parameters based on the summary, and go to 1 again.

C.4 Training results

Once training is completed, each subfolder in `train_package` will contain several output artifacts:

- `programlog`: a log file generated during training. This contains the same information that was visualized in output during training and backtesting.
- `tensorboard`: a folder containing the events for the tensorboard. You can visualize its content by running `tensorboard`: e.g. `tensorboard --logdir=train_package/1`.
- `netfile.*`: the model checkpoints. These can be used to restore a previously trained model.
- `train_summary.csv`: a file with summary information like: network configuration, portfolio value on validation set and test set etc.

C.5 Download Data

To prefetch data to the local database without starting a training run:

```
python main.py --mode=download_data
```

The program will use the configurations in pgportfolio/net_config.json to select coins and download necessary data to train the network.

* Download speed could be very slow and sometimes even have errors in China.

* If you can cannot download data, please check the first release where there is a Data.db file. Copy the file into the database folder. Make sure the online in input in net_config.json is false and run the example. Note that using the this file, you shouldn't make any changes to the input data configuration (for example start_date, end_date or coin_number) otherwise the results may not be correct.

C.6 Backtest

To execute a backtest with rolling training (i.e. online learning in supervised learning) on the target model run:

```
python main.py --mode=backtest --algo=1
```

- --algo could be either the name of a traditional method or the index of the training folder

C.7 Plotting

To plot the results run:

```
python main.py --mode=plot --algos=crp,1 --labels=crp,nnagent
```

- --algos: comma separated list of traditional algorithms and agent indexes
- --labels: comma separated list of names that appear in the plot legend

C.8 Table summary

You can present a summary of the results typing:

```
python main.py --mode=table --algos=1,ons --labels=nntrader,ons
```

- `--algos` and `--labels` are the same as in the plotting case. Labels indicate the row indexes.
- use `--format` arguments to change the format of the table, which could be `raw` `html` `csv` or `latex`. The default one is `raw`.