



# Identifying CSE market signals by integrating technical analysis with machine learning techniques

A Dissertation Submitted for the Degree of  
Master of Computer Science

M.D.S.L. Priyadarshana

University of Colombo School of Computing  
2021



---

## Declaration

I hereby declare that the thesis is my original work and it has been written by me in its entirety. I have duly acknowledged all the sources of information which have been used in the thesis. This thesis has also not been submitted for any degree in any university previously.

Student Name: M.D. Suresh L. Priyadarshana

Registration Number: 2018/MCS/070

Index Number: 18440709

01/12/2021



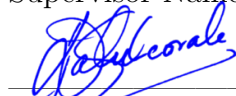
---

Signature of the Student & Date

This is to certify that this thesis is based on the work of Mr. M.D.S.L. Priyadarshana under my supervision. The thesis has been prepared according to the format stipulated and is of acceptable standard.

Certified by,

Supervisor Name: Dr. D. A. S. Atukorale



---

Signature of the Supervisor & Date 01/12/2021

I would like to dedicate this thesis to all the small investors in the  
colombo stock exchange.

# Acknowledgments

Completing this project has indeed been a challenge. As my understanding of the problem grows, the solution narrative changed, and in each iteration, there was new knowledge to acquire. I succeeded in this project because of many people's guidance, advice, support, and encouragement. I would like to thank all of them for their dedication.

First, I would like to thank my esteemed supervisor Dr. Ajantha Athukorala for his invaluable supervision, support and tutelage during the completion of this project.

Next, I would like to give my gratitude to my project coordinator Dr. B.H.R. Pushpananda for redirecting me to the correct way and provide me with guidance throughout the project time.

I would like to extend my gratitude to my teachers and lecturers who advise me in my academic life and all the staff of the University of Colombo School of Computing for the support they have given to me.

Also, I would like to thank my friends, colleagues, and seniors for the advice and feedback given about this project.

Last but not least, my heartfelt thanks to my family, my wife and two children who bared my long working hours and sleepless nights, and my parents who made who today I am.

# Abstract

Investing in publically listed companies in Colombo Stock Exchange is a trendy option among investors. These investments, though carry extremely higher risks, if decisions are taken correctly, investors can achieve extremely higher returns, potentially multiplying the initial investment. However, due to the market volatility, the investors have to keep their close eye on the market sentiments to get the maximum gain, which is not an option for those who don't have time for active trading.

Technical indicators are mathematical and statistical equations based on historical stock price and volume data visually represented as graphs to help investors understand market sentiments.

In this study, we model the problem of stock market investment as a game and try to solve it with a reinforcement learning algorithm. The neural network is a recurrent neural network that understands the market patterns within a 14 day moving window period.

The study includes multiple experiments, training the model with different training data sets, different episode counts, and different types of RNN strategies in the neural network, gated recurrent networks (GRU networks ) and long short-term memory networks ( LSTM networks ), and comparing their profits in different periods.

The study's findings suggest that we cannot create a 'train once use forever' model for stock market predictions. The model performs better when we train it with a recent batch of data, and the model has to be retrained periodically with the previous 200-day price points. Further, it suggests that GRU networks work significantly better than LSTMs and for training, using an episode count of around 500 is sufficient to train.

# Contents

<b>Acknowledgments</b>	<b>iii</b>
<b>Abstract</b>	<b>iv</b>
<b>Contents</b>	<b>v</b>
<b>List of Figures</b>	<b>viii</b>
<b>List of Tables</b>	<b>ix</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation . . . . .	2
1.2 The Problem . . . . .	2
1.2.1 Exact Computer Science Problem . . . . .	4
1.2.2 The Business Problem and the Opportunity . . . . .	4
1.3 Contribution . . . . .	4
1.4 Aims and Objectives . . . . .	5
1.5 Scope . . . . .	5
1.6 Structure of the Thesis . . . . .	6
<b>2 Literature Review</b>	<b>8</b>
2.0.1 Technical Analysis . . . . .	8
2.0.2 Technical Analysis to understand market sentiment . . . . .	8
2.0.3 Stock Market Technical Analysis as a Machine Learning problem. . . . .	9

2.0.4	Stock Market Technical Analysis as a Classification Machine Learning problem. . . . .	10
2.0.5	Stock Market Technical Analysis as a Reinforcement Learning problem . . . . .	11
2.0.6	Research Gap Identified . . . . .	11
<b>3</b>	<b>Methodology</b>	<b>13</b>
3.1	Problem Analysis . . . . .	13
3.2	Solution Approach . . . . .	17
3.2.1	AI component . . . . .	17
3.2.2	Implementation of AI Component . . . . .	20
3.2.3	Training the ML model . . . . .	25
<b>4</b>	<b>Results and Evaluation</b>	<b>28</b>
4.1	Introduction . . . . .	28
4.2	Experiments . . . . .	28
4.2.1	P-G16 . . . . .	29
4.2.2	P-G17 . . . . .	30
4.2.2.1	P-G17A . . . . .	30
4.2.2.2	P-G17B . . . . .	34
4.2.3	P-G24 . . . . .	37
4.2.4	P-G29 . . . . .	39
4.2.5	P-H01 . . . . .	40
4.2.6	P-H04 . . . . .	41
4.2.7	The optimal episode count, compare between P-H04 and P-H06 . . . . .	43
<b>5</b>	<b>Conclusion</b>	<b>47</b>
5.1	Introduction . . . . .	47
5.2	Problems Addressed . . . . .	47
5.3	Future Works . . . . .	48
5.3.1	Researches on Financial Market decisions. . . . .	48
	<b>Bibliography</b>	<b>50</b>

Appendix A : The Source Code	53
------------------------------	----



# List of Figures

1.1	Technical Indicators and Stock Prices of VONE . . . . .	3
3.1	Modeled Investor behavior . . . . .	13
3.2	Modeled Investor behavior . . . . .	14
3.3	MACD 5,12 . . . . .	15
3.4	RSI Calculation . . . . .	16
3.5	RI Calculation . . . . .	17
3.6	Reward Function . . . . .	18
3.7	Q Learning Algorithm . . . . .	19
3.8	Console output for GRU Network summary . . . . .	21
3.9	Console output for LSTM Network summary . . . . .	22
3.10	Source Data extracted from CSE Rest API . . . . .	23
3.11	Calculation of Technical Indicator values . . . . .	24
3.12	Standard VM Pool in GCP . . . . .	27
4.1	Test Episode #50 when trained in P-G17A test, with GRU network	31
4.2	Reward gain with episode in GRU network . . . . .	32
4.3	Test Episode #50 when trained in P-G17B test, with LSTM net- work . . . . .	35
4.4	Reward gain with episode in LSTM network . . . . .	36
4.5	Reward when the episode count is 500 ( in P-H04 ) . . . . .	43
4.6	Reward when the episode count is 1000 ( in P-H06 ) . . . . .	44
4.7	Profit/Loss when the episode count is 500 ( in P-H04 ) . . . . .	45
4.8	Profit/Loss when the episode count is 1000 ( in P-H06 ) . . . . .	46

# List of Tables

3.1	Configurations of the Laptop computer . . . . .	25
4.1	G16 performance with different time periods. . . . .	30
4.2	G17 performance with different time periods, with GRU network .	33
4.3	G17B performance with different time periods, with LSTM net- work . . . . .	37
4.4	G24 performance with different time periods. . . . .	38
4.5	G29 performance with different time periods. . . . .	39
4.6	H01 performance with different time periods. . . . .	41
4.7	H04 performance with different time periods. . . . .	42

# Chapter 1

## Introduction

The Colombo Stock Exchange (CSE) is the primary stock exchange in Sri Lanka. It has 296 companies representing 20 business sectors as of 25 January 2021, with a Market Capitalization of Rs. 3,699 Billion.

When it comes to investing, investing in shares of publicly-held companies is a popular choice among its options. Here an investor buys a smaller piece of a company. If the company value goes higher as time pass, his value on the company also goes higher, which call "the capital gain". The investor is also an owner of the company; he would own a percentage of the profit paid as a dividend. Stock market investing is categorized as a high-risk, high-reward investment option, and when used strategically, it is one the best place to grow wealth.

Compared to other investment options like bonds, real estate, or even directly investing in a company, a stock market investor get the advantage of diversification. Rather than buying a larger chunk of a single company, the investor can invest in multiple companies in smaller chunks in different sectors; he would get more opportunity to diluting his risk. This is called the stock portfolio.

The best strategy to play in the stock market is to buy and hold strategy to secure wealth. Here, an investor buys stocks and holds them for long terms, where he believes that long-term returns will be worth withstanding the short-term volatility. This works well for most of the growing economies where, overall, the stock prices also are in growth.

However, in Sri Lanka, we do not see such constant growth due to its financial instability and ever-changing policies.

---

In Sri Lanka, in CSE, for an investor to get the maximum advantage, when to enter and exit ( Buy / Hold / Sell ), the decision has to be taken correctly timing the market. The inability of doing so can result in losses and missed opportunities.

## 1.1 Motivation

If prices movements were considered for the last 12 years, the stock market has given investors multiple opportunities to buy and sell. For example, in 2012 May-July, in 2019 May - June period, and 2020 March-May due to COVID19 hit, periods stock market reached its minimal values giving investors ample opportunities to buy stocks at a lot discounted prices. Also, during 2011 Jan-Feb, 2014 Oct 2015 Jan and in last week of Jan 2021, the CSE reach its maximum values, where giving investors could sell their stocks at much higher prices.

For example, Expolanka Holdings PLC ( EXPO ) had its stock price around LKR1.70 in May 2020. By Feb 2021, in 9 months, due to the increased demand of its freight services, the price ran up to LKR64.6, giving investors a gain of around 3700%. Though this is an extreme example, any investor who invested in the stock market in May 2020 had his investment doubled by February 2021.

Even though there are such opportunities, many investors, especially show primary roles is not in the stock market, a doctor who uses the stock market to secure his wealth against inflation regularly miss these opportunities. They do not have the luxury of time to pay attention to all the stocks. This results in them having a smaller stock portfolio, a less diversified set of stocks, ultimately increasing the risk and reducing the potential profits.

## 1.2 The Problem

In stock market investments, there are two strategies in taking decisions, fundamental analysis and technical analysis. An investor would consider macro-economic and micro-economic factors to analyze the companies and the market in fundamental analysis. For example, an investor looking into the fundamental

factors would look at government regulations, tariffs, and how the company goes; its assets value, moats, cash flow. Understanding and modeling the variables affecting the market require a significant domain knowledge.

On the other hand, technical analysis is based on stock prices patterns and trading volumes patterns. An investor may use charts and graphs to visualize these patterns. These charts graphs are derived by applying statistical and mathematical equations to historical price/volumes. We call them technical indicators.



Figure 1.1: Technical Indicators and Stock Prices of VONE

---

### **1.2.1 Exact Computer Science Problem**

The problem with technical indicators is that there are many. And not the same technical indicator combination would be effective with every stock. For example, for stocks with low volatility, Moving Averages of periods 15, 26 and their convergence divergence will confidently predict the tendency; for stocks with higher volatility, we have to use MA's with lower periods. And depending on the entire market conditions ( usually reflected with All Share Price Index, ASPI ), the Relative Strength Index ( RSI ), a leading indicator, can be used to correctly predict the price movement before it happens. However, the same indicator can give wrong predictions in a different ASPI condition for the same stocks.

So to get the best results, we may need to create vetted TE combinations for each stock. And then, we have to re-evaluate the same against the ASPI's different conditions. Doing this is a complicated and costly task to be done manually.

However, modeling this as a machine learning problem and using the historical data as the training data can achieve better results. After all the evaluations, this boils down to a classification problem. At a given time, given price/volume information, the application would classify whether the right decision is a Buy, Sell or Hold.

### **1.2.2 The Business Problem and the Opportunity**

For an investor, especially whose primary role is not in the stock market, say, for example, a doctor who uses the stock market to secure his wealth, it is not easy to focus his attention on many stocks daily. However, if he can be presented with a UI where he can see all his stocks and the signal ( Buy, Sell or Hold ) associated with the stock, with a justification that we can retrieve by solving problem 1, he would be greatly benefited.

## **1.3 Contribution**

The research project's main contribution is to develop a model to suggest suitable decisions for a stock with a confidence score; the higher the score, the higher the

---

system is confident about the price movement. The Open, Close, Max, Min prices and traded volumes are used as the algorithm's raw input, converted into technical indicator signals. Together, we use the same data of All Share Price Index (ASPI), converted into technical indicator signals to evaluate the price tendency with the market tendency.

## 1.4 Aims and Objectives

- Literature review, research gap identification and formation of research question.
- Retrieving stock market historical data for analysis.
- Evaluating technical indicator reliability against historical data.
- Building a model that can assess its signal with some confidence score given a stock and its pricing information.
- Building a web-based dashboard where users can select and view the status of their preferred stocks.
- Evaluate and finetune the system so it would maximise the profits of the users.

## 1.5 Scope

There are hundreds of technical indicators that can be used to analyse all the 289 registered companies in the Colombo stock exchange. However, here this research is limited to the following criteria.

- Only analyse a selected set of indicators.
  - Multiple lengths of MACD and their convergent points as the points of interest.
  - Parabolic Sar, parts of Ichimoku Kinko Hyo as trend indicators

- 
- Stochastic, Average Directional Index (ADX), and Ichimoku Kinko Hyo as momentum indicators
  - On-Balance-Volume, Chaikin Money Flow, and Klinger Volume Oscillator as volume indicators.
  - Only apply the analysis to 7 companies.
    - Following companies were selected.
      - \* ACCESS ENGINEERING PLC. - AEL.N0000
      - \* VALLIBEL ONE PLC. - VONE.N0000
      - \* TOKYO CEMENT COMPANY (LANKA) PLC - TKYO.N0000
      - \* SAMPATH BANK PLC. - SAMP.N0000
      - \* JOHN KEELLS HOLDINGS PLC. -JKH.N0000
      - \* HEMAS HOLDINGS PLC. - HHL.N0000
      - \* RICHARD PIERIS EXPORTS PLC. - REXP.N0000
    - companies that have had no stock splitting in history. Splitting is difficult to incorporate with the training model.
    - companies with enough volatility. Low volatile companies doesn't show much movement in the market to generate enough signals.

## 1.6 Structure of the Thesis

This thesis is divided into four main chapters, with specific details, source codes, charts and results, to give an overview of the project. Chapter 1 goes through a detailed description and understanding of the problem domain and the scope of the study; the following chapters consist of information as mentioned below.

The second chapter is the literature review of the problem domain that has been conducted alongside the project. The literature review takes an evolutionary approach. The stated studies under this chapter are the current knowledge, and it explains how we proceeded with different approaches eliminating one by



---

one until we approached the current approach.

In the third chapter, we explain the methodology we have taken to complete and achieve the targets of the quest. In this chapter, we explain the problem, how it is modelled into a game, and then explain the solution approach, data extraction, preprocessing, and the actual implementation of the ML model with the software implementation that supports it.

In the fourth chapter of this thesis, we evaluate the results obtained by this study. The project contained multiple experiments, and the document explains how each experiment changed the narrative of the research conclusions.

The final chapter, in the conclusion chapter, gives the final comments and thoughts about the study. Further, it suggests more research areas associated with the same study than with their market opportunities.

# Chapter 2

## Literature Review

### 2.0.1 Technical Analysis

In (Murphy, John J. 1996) and (Murphy, John J. 1999) , Murphy defines,

Technical analysis is the study of market action, primarily through the use of charts, to forecast future price trends.

In both references, murphy explains most of the technical indicators and how to use them.

### 2.0.2 Technical Analysis to understand market sentiment

It is a widespread argument whether technical analysis can be used to predict market movements and increase profit. Among the many arguments against the theory that the technical analysis cannot beat the market, i.e., get higher returns than the buy and hold strategy, Efficient Market Hypothesis (Fama, Eugene F. 1970) is one of the highest cited arguments. Here, Fama suggests, in an efficient market, the stocks are sold to their fair values. Unless there is new information related to the stock, the stock price cannot fluctuate. This declares no undervalued or overvalued stocks; therefore, neither technical analysis nor fundamental analysis can significantly return. Another such cited argument is the Random Walk Hypothesis (Fama, Eugene F. 1965). As the name suggests, even on top of the efficient market, the price movements can happen, but they

---

happen in a pure random notion, where it is impossible to predict.

Also, against technical analysis, the following arguments are frequently brought forward.

- Nature of Self-fulfilling prophecy
- The inability to use historic price data to predict future prices

There are scenarios where these arguments seem valid. On the brink of Covid-19, the market's downturn is a good scenario that one can argue as a Self-Fulfilling prophecy. In the early Covid19 hit, when the markets faced uncertainty, technical indicators started showing red. Seeing this, people rushed to sell their assets, further collapsing the markets. One may argue that indicators went red because the market is about to crash, and others may argue that the market crashed because indicators showed red. It is a chicken or the egg question.

While (Murphy, John J. 1999) Murphy tries to disprove these arguments theoretically, countless other research such as (Fernando, Pnd 2014) (Muruganandan, S. 2020) have been done to test the actual profitability of technical indicators in different equity markets, and they show conflicting results.

However, most of the researchers agree that considering the fees and costs associated with the transactions, it is challenging to beat the market in significant margins with simple technical trading rules.

### **2.0.3 Stock Market Technical Analysis as a Machine Learning problem.**

However, with the emerging interest in using machine learning, combining them to analyse the financial time series data became a trendy research topic.

Among many machine learning techniques used for this purpose: The Support Vector Machine (SVM) is one of the most popular research options. In (Tay, Francis E. H and Cao, Lijuan 2001) and (Tay, Francis E. H and Cao, Lijuan 2003), Tay and Cao compare the SVM approach with an Artificial Neural Network (ANN) and explore its suitability for predicting market prices. They conclude

---

SVM outperforms compared to ANN. In (Chang, Pei-Chann and Fan, Chin-Yuan and Liu, Chen-Hao 2009), Chang, Fan, and Liu take a different approach using the piecewise linear representation (PLR) method for pattern matching of candlestick charts from historical data. The identified patterns are used as segment 'temporarily turning points' of the historical stock data, which then be inputted into a backpropagation neural network (BPN) for supervised learning of the model. In (Rodríguez-González, Alejandro and Guldrás-Iglesias, Fernando and Colomo-Palacios, Ricardo and Gomez-Berbis, Juan Miguel and Jimenez-Domingo, Enrique and Alor-Hernandez, Giner and Posada-Gomez, Rubén and Cortes-Robles, Guillermo 2010), the González and his team introduce a new term, iRSI, a more accurate RSI value, calculated using artificial intelligence techniques. González's research is fascinating as there are times where RSI's output became questionable among traders, and the same strategy is evaluated inside this project. In (Shynkevich, Yauheniya and McGinnity, T.M. and Coleman, Sonya A. and Belatreche, Ammar and Li, Yuhua 2017), Shynkevich and his team try to evaluate technical indicators' effectiveness by testing them, taking different time windows as input. The pattern is studied using multiple performance metrics: prediction accuracy, winning rate, return per trade and sharp ratio.

#### **2.0.4 Stock Market Technical Analysis as a Classification Machine Learning problem.**

In (Dash, Rajashree and Dash, Pradipta Kishore 2016), Dash introduces a novel decision support system using a computational efficient functional link artificial neural network ( CEFLANN ) and a set of rule to generate the trading decision more efficiently. Dash models "the stock trading decision prediction" problem as a classification problem with three class values of buy, hold or sell signals. In (Borovkova, Svetlana and Tsiamas, Ioannis 2019), Borokova and Tsiamas are working to solve the same problem for high-frequency stock markets, specifically for intraday trading; buying and selling can happen on the same day. For this purpose, They had used a group of LSTM neural networks, and they had operated in a rolling window of one month of days for training the network, one week for

---

evaluating the performance. In (Basak, Suryoday and Kar, Saibal and Saha, Snehanshu and Khaidem, Luckyson and Dey, Sudeepa Roy 2019), Basak and his team use tree-based classifiers to predict the direction of stock market prices.

### **2.0.5 Stock Market Technical Analysis as a Reinforcement Learning problem**

In (Lee, Jae Won and Park, Jonghun and Jangmin and Lee, Jongwoo and Hong, Euyseok 2007), Lee with his team proposes a new trading framework that incorporates multiple Q-learning agents, allowing them to effectively divide and conquer the stock trading problem by defining necessary roles for cooperatively carrying out stock pricing and selection decisions. They tested the framework in the Korean stock market with outstanding performances in profit and risk management. In (Gao, Xiang 2018), Xiang combines the Q-learning agent's strategy with recurrent network strategies. It's questionable why he had used randomly generated price variations to test the capability of the strategy. But the implementation he has tried show the ability to use strategy in actual market conditions.

### **2.0.6 Research Gap Identified**

Most of the discussed researches associating ML try to predict prices' future behaviour, thus outperforming human investors.

However, this project tries to mimic human investor behaviour, and there, it tries to identify the market tendency and suggests the correct decisions.

Though technical indicators cannot predict where the stork price would be in more extended periods, they can be accurately used to identify the tendencies. Though there are uncountable amounts of fundamental factors affecting price movements, this project assumes the cumulative effect of all those factors controls the market interest. The tendency of the market interested can be uncovered via the market patterns.

In this study, we evaluated all the strategies mentioned here and concluded to proceed with a reinforcement learning algorithm based on an RNN model that is based on GRU or LSTM, that is based on (Gao, Xiang 2018). We also evaluate

---

(Lee, Jae Won and Park, Jonghun and Jangmin and Lee, Jongwoo and Hong, Euyseok 2007) approach to include multiple agents in helping taking decisions. Still, due to its added complexity, I decided to consider it in later stages hence out scope for this project.

# Chapter 3

## Methodology

### 3.1 Problem Analysis

We try to build a decision assisting system that mimics a usual technical investor behaviour during this project. To accomplish this, concerning a particular listed company, investor behaviour can be simplified and modelled as a finite state machine as below.

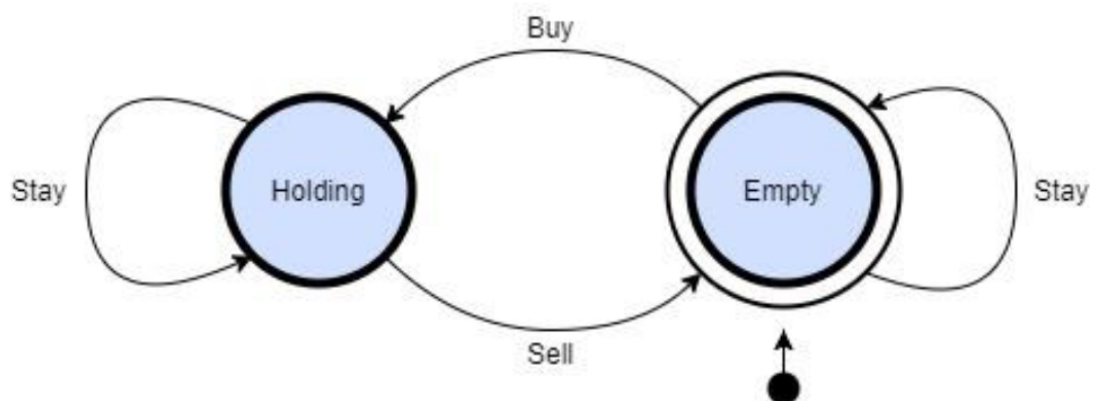


Figure 3.1: Modeled Investor behavior

---

## Empty State - Cash At hand

At the empty stage, the investor only has money in his hand. He has two actions: staying in this state or buying stocks and moving to Holding State. We assume the investor starts and ends his journey from this state.

## Holding State - Stocks At hand

At holding state, we assume the investor no longer has money to buy more stocks. So he has two actions: stay in the same state or sell his holding and move to the Empty state.

Though this is by far from the actual behaviour of a real investor, as he may be buying or selling iteratively. However, in this project, we assume this behaviour reduces the complexity and eliminates unwanted noise in the application.

## Applying the investor behaviour to actual data.



Figure 3.2: Modeled Investor behavior



As per the above figure, it can be seen that our investor can achieve the best gains if he buys the stocks in the first week of October for around LKR18/= per share and sells his holding by mid-November for a price around LKR23/=.

This would give him a profit of around 24% per his investment within a 2-month duration after reducing the transaction costs.

In addition to that, he get another sell high and buy low oppertunity in mid October and also in early November, that will help him increasing his gains further.

### MACD

Moving average convergence divergence (MACD) is a trend-following momentum indicator that shows the relationship between two moving averages of prices. The MACD is calculated by subtracting the longer period exponential moving average (EMA) from the shorter period EMA.

The shorter period EMA curve runs above the longer period when the stock has a bullish trend ( price increasing trend ). And otherwise when the stock has a bear trend ( decreasing trend ). And the locations where these curves are crossed are considered as buy or sell signals.



Figure 3.3: MACD 5,12

---

The main problem of MACD is that it is a lagging indicator. It identifies the trend reversal after it is done. While the reliability of MACD is higher, it doesn't help to reach the optimum gain due to the delay.

Usually, for calculating MACD, 12 days EMA and 26 days EMA is considered. However, depending on the stock's volatility, the 12-16 day EMAs signal could be too late, and some trading opportunities can go unnoticed.

And For identifying the signal, we mainly focus on the sign of MACD rather than the exact value. Therefore we use

$$\tanh MACD_{x,y}$$

At this stage, the project considers the following MACDs.

- $\tanh$  of MACD between 2 and 9
- $\tanh$  of MACD between 5 and 12
- $\tanh$  of MACD between 12 and 26

## RSI

The relative Strength Index (RSI) is a momentum oscillator that indicates the speed and change of price movements. The RSI value varies between 0 and 100, and when it is outside of 30-70 range, it usually considers the stock is oversold or overbought, so a trend reversal is expected.

The RSI is calculated with equations mentioned in [3.4](#) and [3.5](#)

$$RSI = 100 - \frac{100}{1 - RS}$$

Figure 3.4: RSI Calculation

---

$$RS = \frac{\text{Average of } N \text{ day's closes UP}}{\text{Average of } N \text{ day's closes Down}}$$

Figure 3.5: RI Calculation

Though it is a little unreliable, RSI is usually a leading indicator. It indicates the trend traversal before it happens.

### MACD + RSI Strategy

MACD + RSI strategy is one of the basic and most frequently used strategy in stock market. With it, an investor would enter the market ( buy action from empty state ) when the RSI gives an over sold signal with supported by a MACD signal line crossing, and exit the market ( sell the holding ) when the RSI gives an over-bought signal with supported by a MACD signal line crossing. This strategy is used as the primary strategy at the initial stage of the project.

## 3.2 Solution Approach

The complete solution is consist of two parts.

- AI component that builds the model and makes predictions
- Software solution that represents the results and helps analysis.

### 3.2.1 AI component

As per the finite state diagram, from the investor perspective, he has two states of being in and in each state; he has the option to stay in the same state or move to the next state. Depending on the state he selects, he is rewarded as an increment or decrement of the portfolio value.

This can be modelled as a game, where an agent can choose actions based on the

---

state and the environment inputs, and he's rewarded accordingly. This has the exact nature of a reinforcement learning problem.

Therefore I try solving the problem with deep Q-learning with experience. The experience of the agent is stored and randomly replayed at each time step for training.

### Reward Function

The reward function for each action the investor ( the agent ) can take in these states can be shown as below.

$$R_t = \begin{cases} 0 & \textit{stayempty} \\ p(t+1) - p(t) & \textit{stayholding} \\ p(t+1) - p(t) - c & \textit{buy} \\ -c & \textit{sell} \end{cases}$$

Figure 3.6: Reward Function

$c$  -transaction charges ( 1.34% x  $p(t)$  in Colombo Stock Exchange )

### Q-Learning Algorithm

Pseudo code of the Deep Q-Learning algorithm used in the agent is as follows

---

```
for episode = 1 to N
  for time step t = 1 to T
    With probability  $\epsilon$  select a random valid action at
    Otherwise select the valid action at that maximise predicted Q
    Given  $a_t$ , emulator returns reward  $r_t$  and new state  $s_{t+1}$ 
    Store  $(s_t, a_t, r_t, s_{t+1})$  in memory
    for memory  $(s_t, a_t, r_t, s_{t+1})$  in sampled minibatch
      perform gradient descent on  $Q(s_j, a_j)$  using target value
      if game ends at j
         $Q_{\text{target}}(s_j, a_j) = r_j$ 
      else
         $Q_{\text{target}}(s_j, a_j) = r_j + \gamma \max_a Q(s_{j+1}, a)$ 
      end for
    end for
  end for
end for
```

Figure 3.7: Q Learning Algorithm

---

## Recurrent Neural Networks

When analysing the indicator values, we are looking at data in a time series. Moreover, today's correct trade decision ( to buy, stay, or sell ) does not entirely depend on today's market conditions.

Instead, it is required to have some memory of past events. Due to that reason, this problem requires a recurrent neural network.

Therefore the model is created with an RNN component.

## Currently Testing Models

Currently the project is experimenting with

- A GRU model - A neural network with GRU ( Gated Recurrent Unit ) layers
- A LSTM model - A neural network with a LSTM ( Long Short Term Memory ) layers

### 3.2.2 Implementation of AI Component

As mentioned in the literature review, the implementation of the machine learning section is mainly based on Xiang's implementation of (Gao, Xiang 2018) research. The source code is available at <https://github.com/golsun/deep-RL-trading>.

The source code is modified to facilitate the following

- A new sampler was introduced 'company\_data\_sampler' replacing existed SinSampler and PairSampler, to feed actual market values rather than generating random market values. 'company\_data\_sampler' also converts the company pricing data into technical indicator values.
- Remove Convolutional Neural Network ( CNN ) and Multi-Layer Perception (MLP) related implementations, as in this project would only consider RNN based training.

---

Layer (type)	Output Shape	Param #
reshape (Reshape)	(None, 40, 2)	0
gru (GRU)	(None, 40, 32)	3456
gru_1 (GRU)	(None, 40, 32)	6336
gru_2 (GRU)	(None, 32)	6336
dense (Dense)	(None, 32)	1056
dense_1 (Dense)	(None, 32)	1056
dense_2 (Dense)	(None, 3)	99
Total params: 18,339		
Trainable params: 18,339		
Non-trainable params: 0		

Figure 3.8: Console output for GRU Network summary

---

```

Layer (type)                Output Shape                Param #
=====
reshape (Reshape)           (None, 40, 2)              0
lstm (LSTM)                  (None, 40, 32)             4480
lstm_1 (LSTM)                (None, 40, 32)            8320
lstm_2 (LSTM)                (None, 32)                 8320
dense (Dense)                (None, 32)                 1056
dense_1 (Dense)              (None, 32)                 1056
dense_2 (Dense)              (None, 3)                  99
=====
Total params: 23,331
Trainable params: 23,331
Non-trainable params: 0

```

Figure 3.9: Console output for LSTM Network summary



- 
- RandomMovingMarket extends the actual behaviour of Market to use random segments of the entire dataset for training. This is to mimic cross-validation behaviour in a usual machine learning implementation.

**Data collection** As this project emphasizes technical readings, the raw data we require are Open, Close, High, Low prices and Volume information. There are many approaches to extract this information, including web scrapping tools that parse data from HTML pages. However, in the research, we found a private API from the CSE website, which contain all the required information. From this Rest based API following information is extracted.

- Open - Stock price when the market is opened on the day
- Close - Stock price when the market is closed on the day.
- High - Highest price the market reached during a day.
- Low - Lowest price the market reached during a day.
- Volume - Total number of stocks traded during a day.

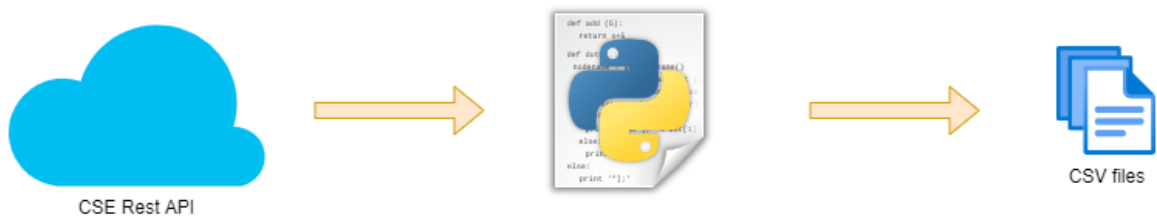


Figure 3.10: Source Data extracted from CSE Rest API

**Data Preprocessing and Preparation** Within this project scope, we are not primarily interested in the raw pricing information. Instead, we are required to transform these values into technical indicators. This transformation is done lazily when the data is feed into the training algorithm, as a part of the data sampler.

---

```

def load_data(file_path):
    data = pd.read_csv(file_path)
    data['Date'] = pd.to_datetime(data['Date'])

    TechIndicator = copy.deepcopy(data)

    TechIndicator['Momentum_1D'] = (TechIndicator['Close'] - TechIndicator['Close'].shift(1)).fillna(0)
    TechIndicator['RSI_14D'] = TechIndicator['Momentum_1D'].rolling(center=False, window=14).apply(rsi).fillna(0)

    TechIndicator['26_ema'] = TechIndicator['Close'].ewm(span=26, min_periods=0, adjust=True, ignore_na=False).mean()
    TechIndicator['12_ema'] = TechIndicator['Close'].ewm(span=12, min_periods=0, adjust=True, ignore_na=False).mean()
    TechIndicator['9_ema'] = TechIndicator['Close'].ewm(span=9, min_periods=0, adjust=True, ignore_na=False).mean()
    TechIndicator['5_ema'] = TechIndicator['Close'].ewm(span=5, min_periods=0, adjust=True, ignore_na=False).mean()
    TechIndicator['2_ema'] = TechIndicator['Close'].ewm(span=2, min_periods=0, adjust=True, ignore_na=False).mean()

    TechIndicator['MACD_2_9'] = (np.tanh((TechIndicator['2_ema'] - TechIndicator['9_ema']) * 1000))
    TechIndicator['MACD_5_12'] = (np.tanh((TechIndicator['5_ema'] - TechIndicator['12_ema']) * 1000))
    TechIndicator['MACD_12_26'] = (np.tanh((TechIndicator['12_ema'] - TechIndicator['26_ema']) * 1000))

    TechIndicator = TechIndicator.fillna(0)

> columns2Drop = [ ...
]
TechIndicator = TechIndicator.drop(labels=columns2Drop, axis=1)

> data_columns = [ ...
]

np_arr = TechIndicator[data_columns].to_numpy()
return (np_arr)

```

Figure 3.11: Calculation of Technical Indicator values

---

### 3.2.3 Training the ML model

#### Training the Model with Laptop Computer

Initially, the model was trained with a laptop computer with the following specs.

<b>Processor</b>	Intel®Core™i5 10210U
<b>RAM</b>	DDR4 16GB
<b>GPU</b>	Not Enabled

Table 3.1: Configurations of the Laptop computer

The training of the model with 200 price points took nearly 28 hours. This is a considerably large amount of time. Especially when it is required to make frequent changes to the network and test. Due to this reason, it was required to find an alternative approach to train the model.

Following approaches were considered during the project.

- Inforporating Laptop GPU for training - This was not possible due to a version incompatiiblity with TensorFlow and the GPU drivers.
- Using Google Colabotary
- Using Cloud servers for training the models.

#### Training the model with Google Colaboratory

Google Colaboratory ( or Google Colab for short ) is a Free SAAS product provided by google research that replicates Jupyter Notebook in the cloud. This allows anybody to write and execute python code through a browser, with no initial setup and free access to computing resources, including GPUs and TPUs.

---

Google Colab servers have a restriction that we cannot run processing for longer hours. Moreover, the algorithm took too much time, and it had to find an alternative mechanism to train the model.

### **Training the model with GPU driven Cloud VMs**

AWS SageMaker and GCP AI platform are tools that provide cloud-hosted Jupyter notebooks that allow executing python codes through a browser. And such services are not provided under the free tire. While they are costly to carry out the project, I configured a couple of VMs in AWS and GCP.

However, in training, it was noted that GPUs are not giving much of the gain for the Q-Learning algorithm I had used.

Both GPU backed VMs, and simple VMs with some CPU capacity were training the model roughly in the same period.

In further investigation, it was clear, the loop in the Q-Learning algorithm that evaluates the model for each price point again causes the delay and using even a GPU, we can get a smaller gain there.

### **Training the model with standard GCP Cloud VMs**

As the standard VMs also providing almost the same performance as GPU hosted machines, We switched to using GCP VMs to get it working. We created 9 VMs that ran from time to time and trained the machine learning model with different periods and configurations. This gave a significant cost reduction.

VM instances CREATE INSTANCE IMPORT VM REFRESH START / RESUME STOP SUSPEND RESET

**INSTANCES** INSTANCE SCHEDULE

VM instances are highly configurable virtual machines for running workloads on Google infrastructure. [Learn more](#)

**Filter** Enter property name or value

<input type="checkbox"/>	Status	Name ↑	Internal IP	External IP	Labels	Connect
<input type="checkbox"/>	●	gru-instance-4	10.190.0.2 (nic0)	None	key : expo	SSH ↓ ⋮
<input type="checkbox"/>	●	gru-processor	10.128.0.5 (nic0)	None		SSH ↓ ⋮
<input type="checkbox"/>	●	gru-processor-1	10.128.0.8 (nic0)	None		SSH ↓ ⋮
<input type="checkbox"/>	●	gru-processor-2	10.128.0.10 (nic0)	None		SSH ↓ ⋮
<input type="checkbox"/>	●	gru-processor-3	10.128.0.12 (nic0)	None		SSH ↓ ⋮
<input type="checkbox"/>	●	lstm-instance-1	10.128.0.9 (nic0)	None	ubuntu-box : ubuntu-box	SSH ↓ ⋮
<input type="checkbox"/>	●	lstm-instance-2	10.128.0.11 (nic0)	None	ubuntu-box : ubuntu-box	SSH ↓ ⋮
<input type="checkbox"/>	●	lstm-instance-3	10.128.0.13 (nic0)	None	ubuntu-box : ubuntu-box	SSH ↓ ⋮
<input type="checkbox"/>	●	ubuntu-instance-3	10.128.0.4 (nic0)	None	ubuntu-box : ubuntu-box	SSH ↓ ⋮

Figure 3.12: Standard VM Pool in GCP

# Chapter 4

## Results and Evaluation

### 4.1 Introduction

In this study, we had to conduct a series of experiments to narrow down the best combination of date period, the neural network model and the most efficient episode count to get the task correctly done.

These experiments are conducted in an evolutionary method where the result of each experiment clarifies a choice and in the next experiments assume the correctness of that assumption. While this is not the most optimum way to correctly evaluate the choices, due to the training overhead and the cost involved in multiple experiments for the same choice, we had to let every experiment control the project's narrative.

### 4.2 Experiments

The primary company used for the analysis was Access Engineering PLC ( AEL.N0000 ). The company's price points are available from 2012-03-27 to the current date. The existing pricing data was used as the training data with selecting different training periods, selecting different episode counts.

---

Training Set ID	Description
P-G16	200 price points, 500 episodes - (bug in reward function)
P-G17A	Same P-G16, the bug in reward function fixed, GRU network
P-G17B	Same P-G16, the bug in reward function fixed, LSTM network
P-G24	200 price points, 500 episodes (random samples)
P-G29	800 price points, 500 episodes (random samples)
P-H01	1800 price points, 500 episodes. ( no random )
P-H04	200 price points starting from 2019, 500 episodes.
P-H06	200 price points starting from 2019, 1000 episodes.

### 4.2.1 P-G16

The model was trained with price points starting from 2012-03-27, for 200 days period. This is roughly the first year Access Engineering PLC after its IPO. As the rewarding function had a bug in its implementation, the model had not considered the cash-holding action much and therefore; it had done more trading than it should.

The trained model's behaviour for different time bands can be seen as below. Here, in each case, we assume the agent invests LKR50,000 on the first day of the duration and check his investment's worth after the period.

<b>From</b>	<b>To</b>	<b>Final Worth</b>	<b>Profit/Loss</b>	<b>Profit/Loss Gain</b>
2012-07-01	2013-06-30	121432.8	71432.83	142.86%
2013-01-01	2013-12-31	57138.85	7138.85	14.27%
2013-07-01	2014-06-30	51353.94	1353.94	2.70%
2014-01-01	2014-12-31	54021.08	4021.08	8.04%
2014-07-01	2015-06-30	42323.19	-7676.80	-15.35%
2015-01-01	2015-12-31	31921.293	-18078.70	-36.15%
2015-07-01	2016-06-30	36112.44	-13887.55	-27.77%
2016-01-01	2016-12-31	45507.024	-4492.97	-8.98%
2016-07-01	2017-06-30	49506.08	-493.91	-0.98%
2017-01-01	2017-12-31	45311.36	-4688.63	-9.37%
2017-07-01	2018-06-30	36133.85	-13866.14	-27.73%
2018-01-01	2018-12-31	37411.735	-12588.26	-25.17%
2018-07-01	2019-06-30	42540.22	-7459.77	-14.91%
2019-01-01	2019-12-31	71004.19	21004.19	42.00%
2019-07-01	2020-06-30	72546.00	22546.00	45.09%
2020-01-01	2020-12-31	57077.32	7077.32	14.15%
2020-07-01	2021-06-30	63173.04	13173.04	26.34%

Table 4.1: G16 performance with different time periods.

## 4.2.2 P-G17

After fixing the issue in the reward function, the model was trained for the same date period. In this approach, we could see better, more reliable behaviour, that the model is trying to hold the cash or money more.

### 4.2.2.1 P-G17A

This training session conducted with setting the GRU network as the core Neural network. Following is how the agent’s reward is accumulated during the test period.

And it can be seen that even when the stock price is decreasing, the agent is capable of getting rewards.



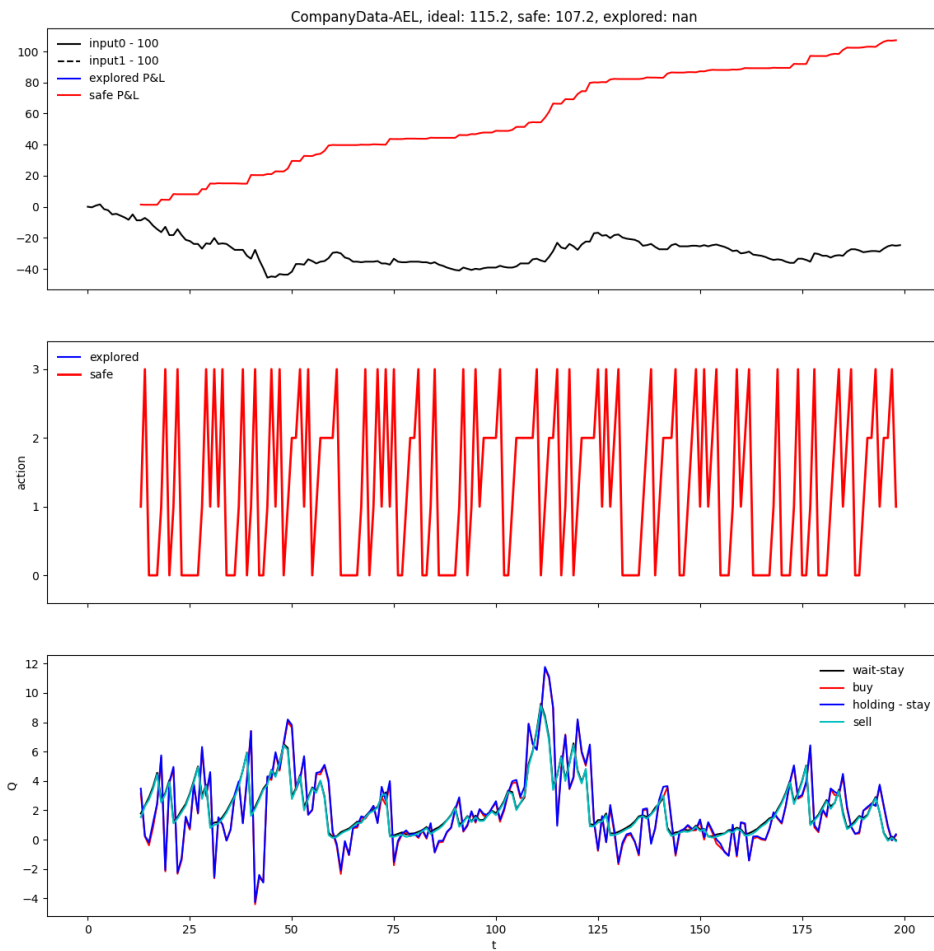


Figure 4.1: Test Episode #50 when trained in P-G17A test, with GRU network

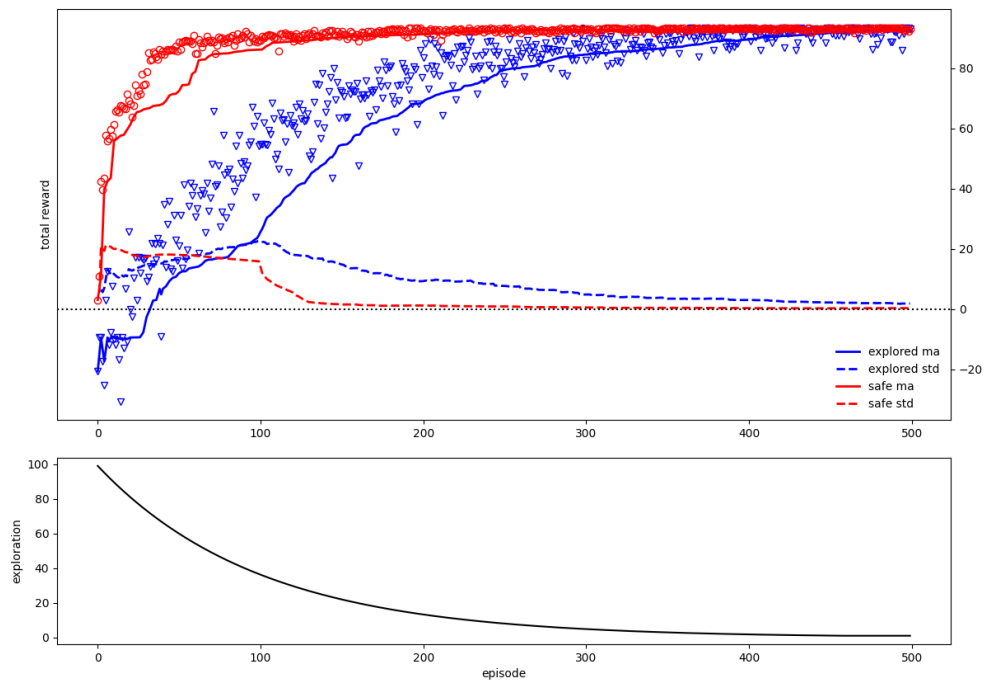


Figure 4.2: Reward gain with episode in GRU network

Then we use the trained model to suggest the trading decisions in the following durations. Note: Here, the model has only seen the price points between 2012-03-27 and 2013-01-22.

And we assume the agent invests LKR50,000 on the first day of the duration and check his investment's worth after the period, after taking buying selling decisions based his predictions.

<b>From</b>	<b>To</b>	<b>Final Worth</b>	<b>Profit/Loss</b>	<b>Profit/Loss Gain</b>
2012-07-01	2013-06-30	132246.06	82246.06	164.49%
2013-01-01	2013-12-31	58207.97	8207.97	16.41%
2013-07-01	2014-06-30	59094.73	9094.73	18.18%
2014-01-01	2014-12-31	67232.21	17232.21	34.46%
2014-07-01	2015-06-30	32115.63	-17884.36	-35.76%
2015-01-01	2015-12-31	23118.76	-26881.23	-53.76%
2015-07-01	2016-06-30	34813.66	-15186.33	-30.37%
2016-01-01	2016-12-31	48817.44	-1182.55	-2.36%
2016-07-01	2017-06-30	53193.45	3193.45	6.38%
2017-01-01	2017-12-31	44175.41	-5824.58	-11.64%
2017-07-01	2018-06-30	39674.16	-10325.83	-20.65%
2018-01-01	2018-12-31	37073.63	-12926.36	-25.85%
2018-07-01	2019-06-30	42927.06	-7072.93	-14.14%
2019-01-01	2019-12-31	79521.48	29521.48	59.04%
2019-07-01	2020-06-30	55977.08	5977.08	11.95%
2020-01-01	2020-12-31	57533.21	7533.21	15.06%
2020-07-01	2021-06-30	63574.22	13574.22	27.14%

Table 4.2: G17 performance with different time periods, with GRU network

Though the model can secure a significantly higher reward of 164.49% during the training period ( the data it has already seen ), the reward is much smaller in later date periods.

---

Especially during 2015 to early 2019, the agent fails to secure a positive reward.

#### **4.2.2.2 P-G17B**

This training session conducted with setting the LSTM network as the core Neural network. Following is how the agent's reward is accumulated during the test period.

And it can be seen that even when the stock price is decreasing, the agent is capable of getting rewards.

Then we use the trained model to suggest the trading decisions in the following durations. Note: Here, the model has only seen the price points between 2012-03-27 and 2013-01-22.

And we assume the agent invests LKR50,000 on the first day of the duration and check his investment's worth after the period, after taking buying selling decisions based his predictions.

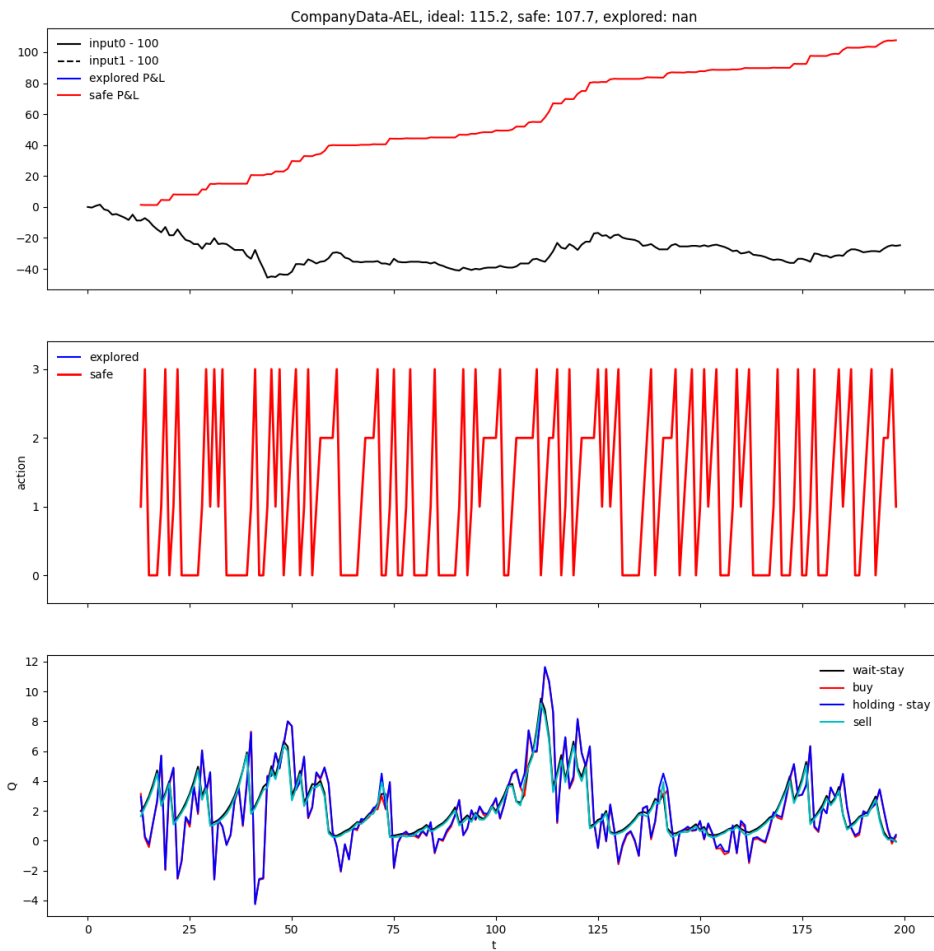


Figure 4.3: Test Episode #50 when trained in P-G17B test, with LSTM network

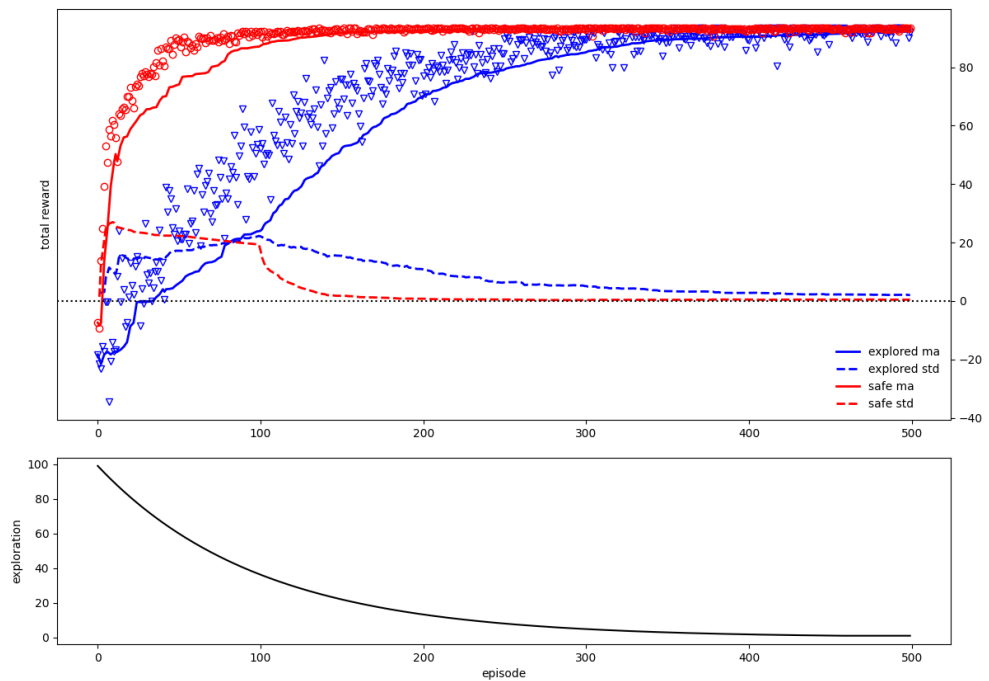


Figure 4.4: Reward gain with episode in LSTM network

<b>From</b>	<b>To</b>	<b>Final Worth</b>	<b>Profit/Loss</b>	<b>Profit/Loss Gain</b>
2012-07-01	2013-06-30	107315.4	57315.47	114.63%
2013-01-01	2013-12-31	55021.44	5021.44	10.04%
2013-07-01	2014-06-30	54167.53	4167.53	8.33%
2014-01-01	2014-12-31	58654.21	8654.21	17.30%
2014-07-01	2015-06-30	38948.3	-11051.62	-22.10%
2015-01-01	2015-12-31	34114.78	-15885.21	-31.77%
2015-07-01	2016-06-30	40653.53	-9346.46	-18.69%
2016-01-01	2016-12-31	55208.57	5208.57	10.41%
2016-07-01	2017-06-30	53195.32	3195.32	6.39%
2017-01-01	2017-12-31	41898.47	-8101.52	-16.20%
2017-07-01	2018-06-30	40999.47	-9000.52	-18.00%
2018-01-01	2018-12-31	36883.56	-13116.43	-26.23%
2018-07-01	2019-06-30	39389.86	-10610.13	-21.22%
2019-01-01	2019-12-31	61748.46	11748.46	23.49%
2019-07-01	2020-06-30	44121.58	-5878.41	-11.75%
2020-01-01	2020-12-31	45847.58	-4152.41	-8.30%
2020-07-01	2021-06-30	67224.63	17224.63	34.44%

Table 4.3: G17B performance with different time periods, with LSTM network

With LSTM network, the model can secure a higher reward of 114.63% during the training period ( the data it has already seen ), comparing to the GRU network , we can it is however lower to the GRU’s values.

### 4.2.3 P-G24

As the agent consistently failed to predict the price actions accurately for the period between 2015 to early 2019, it was assumed that the agent had trained only the prices between 2012-03-27 and 2013-01-22, and there is a missing pattern the agent is not trained for.

Therefore, the implementation is modified not to use a limited period when selecting price points. Instead, it was modified to pick 200 price points from the

entire price point history randomly.

Again here also we assume the agent invests LKR50,000 on the first day of the duration and check his investment's worth after the period, after taking buying selling decisions based his predictions.

<b>From</b>	<b>To</b>	<b>Final Worth</b>	<b>Profit/Loss</b>	<b>Profit/Loss Gain</b>
2012-07-01	2013-06-30	46684.35	-3315.64	-6.63%
2013-01-01	2013-12-31	50322.46	322.46	0.64%
2013-07-01	2014-06-30	49820.44	-179.55	-0.35%
2014-01-01	2014-12-31	49166.00	-833.99	-1.66%
2014-07-01	2015-06-30	48812.30	-1187.69	-2.37%
2015-01-01	2015-12-31	48379.61	-1620.38	-3.24%
2015-07-01	2016-06-30	50289.33	289.33	0.57%
2016-01-01	2016-12-31	49280.63	-719.36	-1.43%
2016-07-01	2017-06-30	48360.55	-1639.44	-3.27%
2017-01-01	2017-12-31	48225.14	-1774.85	-3.54%
2017-07-01	2018-06-30	48551.64	-1448.35	-2.89%
2018-01-01	2018-12-31	48676.02	-1323.97	-2.64%
2018-07-01	2019-06-30	49277.06	-722.93	-1.44%
2019-01-01	2019-12-31	59338.89	9338.89	8.67%
2019-07-01	2020-06-30	56504.48	6504.48	3.00%
2020-01-01	2020-12-31	52870.22	2870.22	5.74%
2020-07-01	2021-06-30	55096.98	5096.98	0.19%

Table 4.4: G24 performance with different time periods.

This approach ended up in failure. The agent failed to secure profits even for the duration that it had previously had got positive rewards.



#### 4.2.4 P-G29

At P-G24, we had selected only 200 random price points where there are 2000+ price points. Only 10% from the actual dataset. So we assumed the model fails to identify patterns when fed with a significantly smaller percentage of data.

At P-G29, we use the same strategy of randomly picking the price points, but here we increased the amount to 800

From	To	Final Worth	Profit/Loss	Profit/Loss Gain
2012-07-01	2013-06-30	50625.39	625.39	1.25% %
2013-01-01	2013-12-31	49276.56	-723.43	-1.44% %
2013-07-01	2014-06-30	49259.89	-740.10	-1.48% %
2014-01-01	2014-12-31	66581.95	16581.95	33.16% %
2014-07-01	2015-06-30	65297.29	15297.29	30.59% %
2015-01-01	2015-12-31	47772.87	-2227.12	-4.45% %
2015-07-01	2016-06-30	45811.87	-4188.12	-8.37% %
2016-01-01	2016-12-31	46538.62	-3461.37	-6.92% %
2016-07-01	2017-06-30	48700.45	-1299.54	-2.59% %
2017-01-01	2017-12-31	51313.22	1313.22	2.62% %
2017-07-01	2018-06-30	42463.40	-7536.59	-15.07% %
2018-01-01	2018-12-31	52598.74	2598.74	5.19% %
2018-07-01	2019-06-30	60430.41	10430.41	20.86% %
2019-01-01	2019-12-31	45060.95	-4939.04	-9.87% %
2019-07-01	2020-06-30	45604.30	-4395.69	-8.79% %
2020-01-01	2020-12-31	45994.21	-4005.78	-8.01% %
2020-07-01	2021-06-30	45752.82	-4247.17	-8.49% %

Table 4.5: G29 performance with different time periods.

At this stage, we could see it is still failing to gain higher rewards. Even The model had marginally secured positive rewards during 2017, 2018, and early 2019 periods, but it had still failed to secure gains where the stock had been

---

rallying, especially like durations between 2020-07-01 and 2021-06-30.

#### **4.2.5 P-H01**

The randomisation component was removed from the model to isolate why it fails when presented with a broader range of price points. Then we trained the model with 1800 price points. It was trained with price points from 2012-03-27 to 2019-09-19, so it will include all years between 2012 to 2019 where mixed results were shown.

The trained model's behaviour for different time bands can be seen as below. Here, again, in each case, we assume the agent invests LKR50,000 on the first day of the duration and check his investment's worth after the period.

From	To	Final Worth	Profit/Loss	Profit/Loss Gain
2012-07-01	2013-06-30	51893.57	1893.57	3.78%
2013-01-01	2013-12-31	39082.42	-10917.57	-21.83%
2013-07-01	2014-06-30	41959.34	-8040.65	-16.08%
2014-01-01	2014-12-31	61562.84	11562.84	23.12%
2014-07-01	2015-06-30	62402.78	12402.78	24.80%
2015-01-01	2015-12-31	36758.57	-13241.42	-26.48%
2015-07-01	2016-06-30	38317.64	-11682.35	-23.36%
2016-01-01	2016-12-31	51288.14	1288.14	2.57%
2016-07-01	2017-06-30	53174.74	3174.74	6.34%
2017-01-01	2017-12-31	49765.93	-234.06	-0.46%
2017-07-01	2018-06-30	37648.03	-12351.96	-24.70%
2018-01-01	2018-12-31	39659.91	-10340.08	-20.68%
2018-07-01	2019-06-30	58814.09	8814.09	17.62%
2019-01-01	2019-12-31	63248.80	13248.80	26.49%
2019-07-01	2020-06-30	57506.61	7506.61	15.01%
2020-01-01	2020-12-31	49799.50	-200.49	-0.40%
2020-07-01	2021-06-30	40104.28	-9895.71	-19.79%

Table 4.6: H01 performance with different time periods.

In this approach, also we see mixed results. Moreover, in most periods, the agent has not secured positive rewards. and in some durations, its lost has been around 25%. At this stage we concluded that agent performs poorly when the model is trained with wider range of price points.

#### 4.2.6 P-H04

At **P-G17** where the model is trained with price points between 2012-03-27 and 2013-01-22, we could see the agent is securing higher rewards in the 2013-2014 period. Therefore to test how the agent performs when the model is trained with data from a recent year, we trained the model with price points between 2019-01-02 and 2019-10-30 ( 200 price points).

---

Then the trained model evaluated assuming the agent invests LKR50,000 on the first day of the duration and check his investment's worth after the period. However, we are more interested in how the agent behaves between 2019-07 to the current date.

<b>From</b>	<b>To</b>	<b>Final Worth</b>	<b>Profit/Loss</b>	<b>Profit/Loss Gain</b>
2012-07-01	2013-06-30	47256.05	-2743.94	-5.48%
2013-01-01	2013-12-31	47401.17	-2598.82	-5.19%
2013-07-01	2014-06-30	47649.02	-2350.97	-4.70%
2014-01-01	2014-12-31	55105.71	5105.71	10.21%
2014-07-01	2015-06-30	43055.24	-6944.75	-13.88%
2015-01-01	2015-12-31	50022.15	22.15	0.04%
2015-07-01	2016-06-30	52815.46	2815.46	5.63%
2016-01-01	2016-12-31	48108.90	-1891.09	-3.78%
2016-07-01	2017-06-30	48009.36	-1990.63	-3.98%
2017-01-01	2017-12-31	49020.60	-979.39	-1.95%
2017-07-01	2018-06-30	40772.74	-9227.25	-18.45%
2018-01-01	2018-12-31	45459.09	-4540.90	-9.08%
2018-07-01	2019-06-30	65145.18	15145.18	30.29%
2019-01-01	2019-12-31	182456.09	132456.09	264.91%
2019-07-01	2020-06-30	203225.87	153225.87	306.45%
2020-01-01	2020-12-31	68470.88	18470.88	36.94%
2020-07-01	2021-06-30	43580.71	-6419.28	-12.83%

Table 4.7: H04 performance with different time periods.

Here we can see the agent has secured significant rewards during and near its trained period ( with the data it has seen and ) in 2019-01 to 2020-07, it has reached around 300% rewards.

---

### 4.2.7 The optimal episode count, compare between P-H04 and P-H06

Following figures shows how the reward is increased with different episode counts. And below we can see that when we reaches to episode count to 500, the reward reaches to its maximum, and there onwards, we cannot see an increment. So we can deduct that the optimum episode count is around 500 and we can use that value for our future implementations.

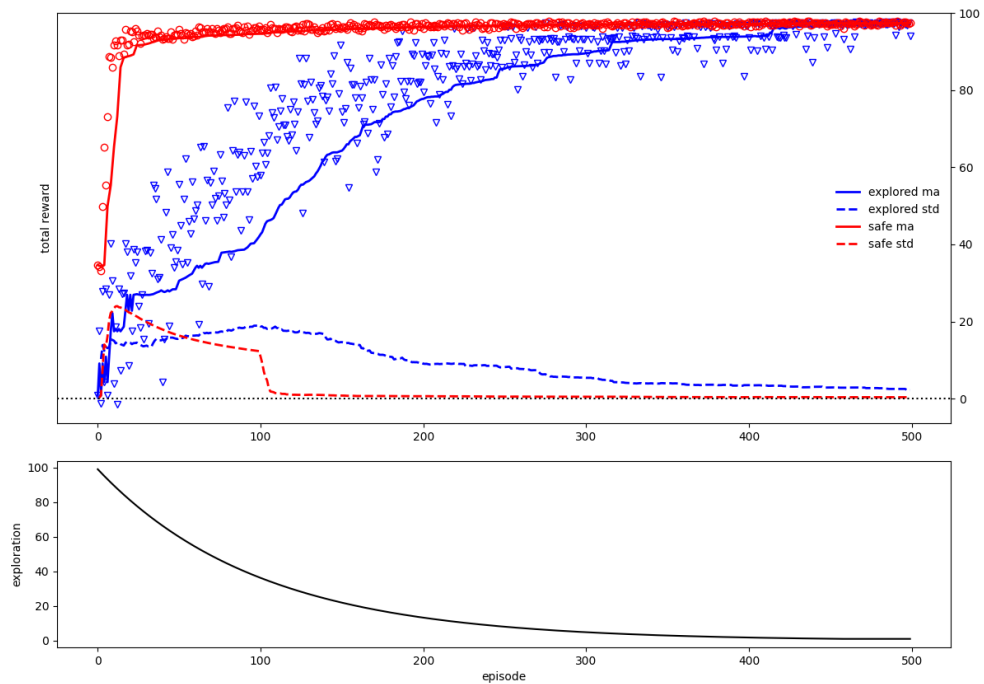


Figure 4.5: Reward when the episode count is 500 ( in P-H04 )

Following figures shows how the overall profit is changed with different episode counts.

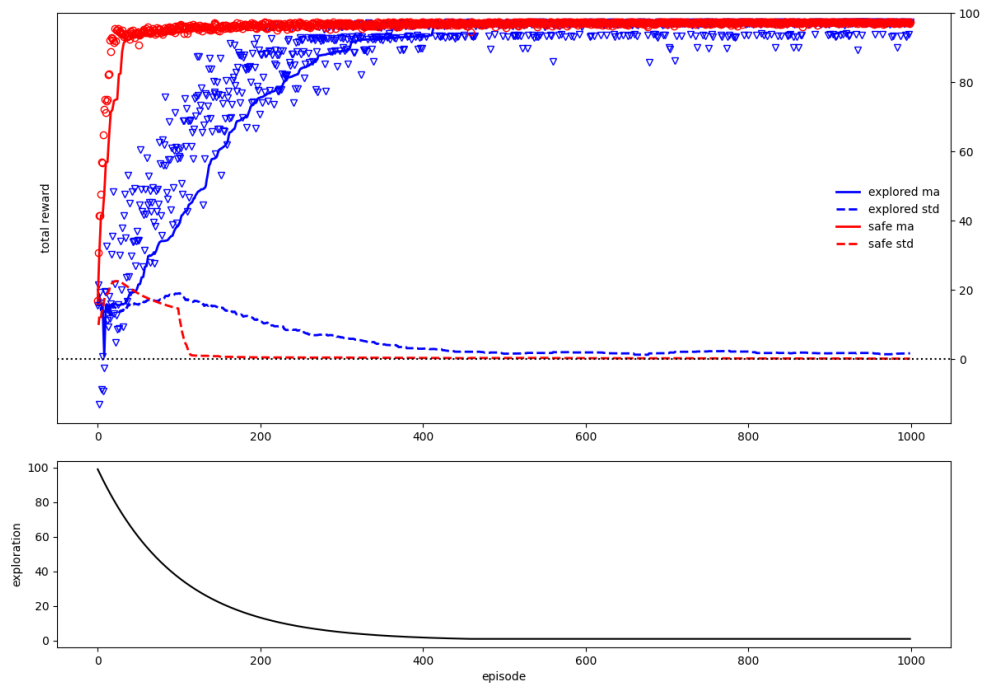


Figure 4.6: Reward when the episode count is 1000 ( in P-H06 )

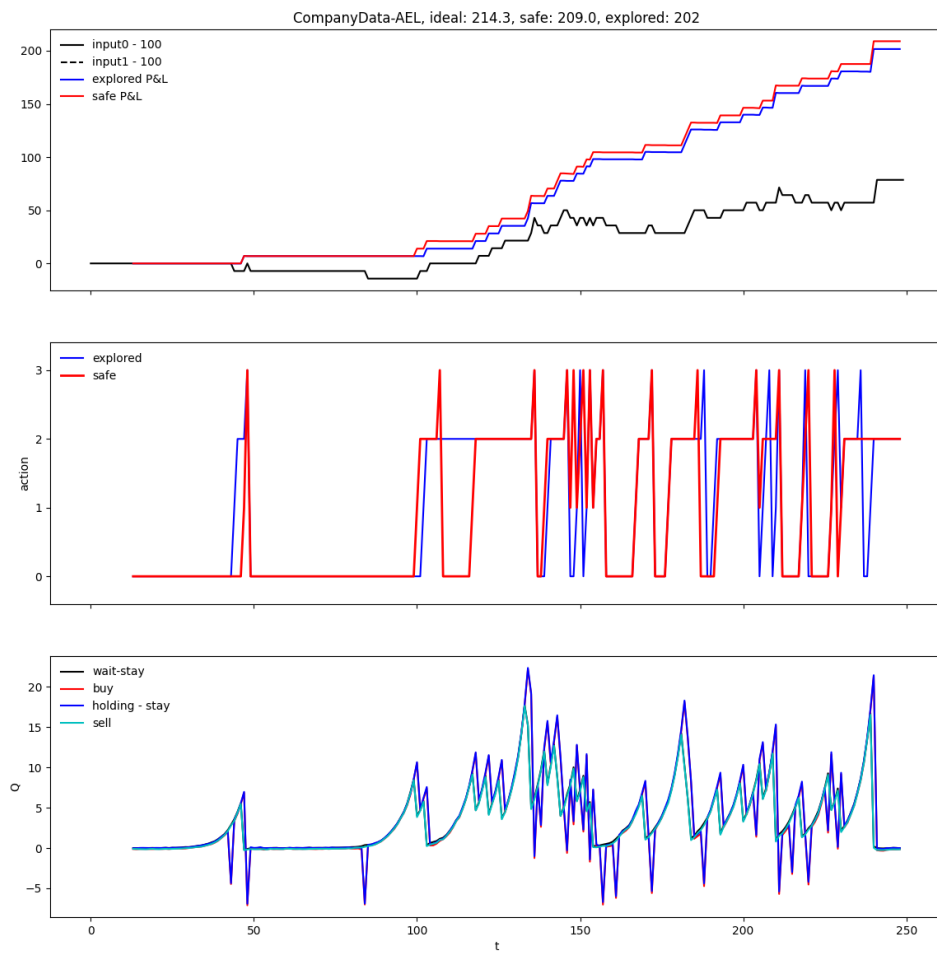


Figure 4.7: Profit/Loss when the episode count is 500 ( in P-H04 )

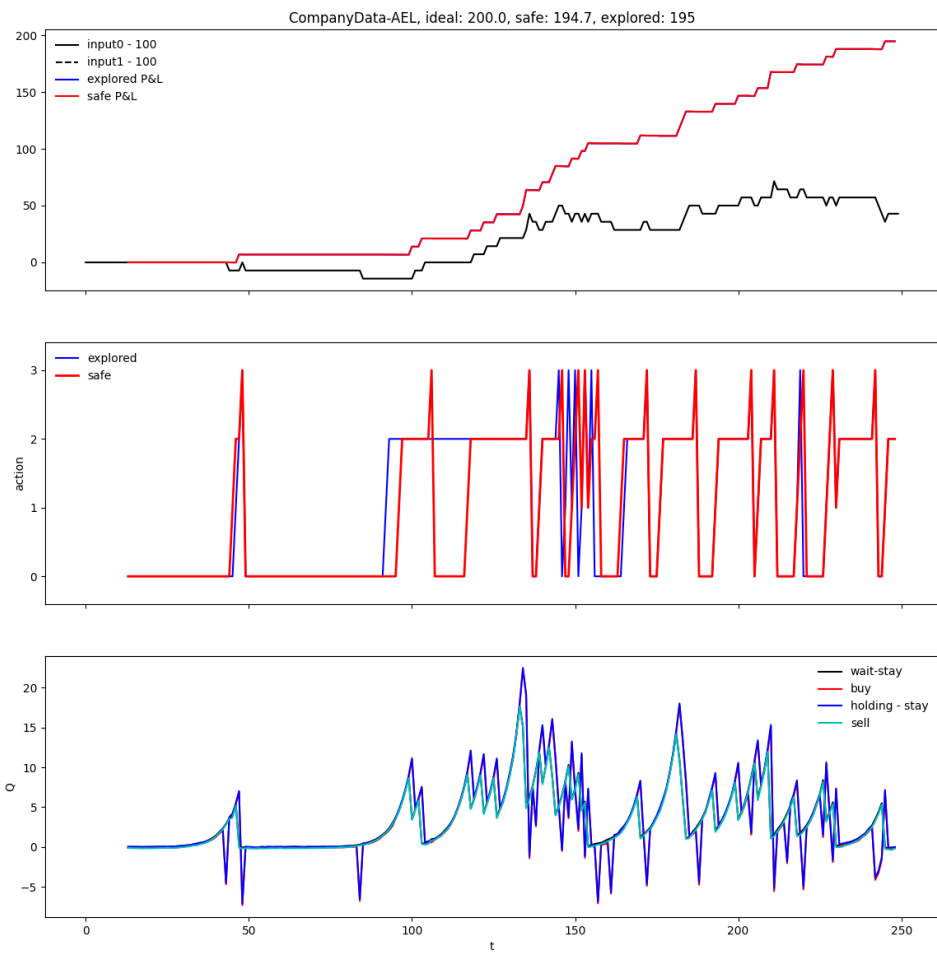


Figure 4.8: Profit/Loss when the episode count is 1000 ( in P-H06 )



# Chapter 5

## Conclusion

### 5.1 Introduction

In this study, we tried to find an approach to predict market behaviour using technical indicator signals. The study was carried out with three main steps comprehensive literature view, implementation of a machine learning model to identify patterns and make predictions and the visual framework for users to see the predications from the system and do the analysis. This chapter wraps this document by providing the final quest comments, thoughts and future works to extend this study.

### 5.2 Problems Addressed

If we scrutinize the gained results for different combinations, we cannot create a train once use forever model. Even the models return significantly higher results in specific periods; when applied to another, it shows losses.

Following observations are noted in the study:

- Models trained with a recent date period tend to give better yields.
- Trying to train the model for the entire price do not improve the model performance.

---

The explanation for these observations right now is behavioural changes of people. By observing the graph patterns, we observe the investor interests and behaviours. When the government policies, geo-economic factors, company strategies change over time, investor perception of the company also changes. I authorize that these changes affect the buying patterns making a trained model for a particular duration invalid. So the solution would be, just as people learn continuously, we must train the model from time to time to sync with current trends.

## 5.3 Future Works

Research on investments in financial markets, in stock markets, as well as a vast domain of research. Moreover, it is a huge market. Many companies research market data and help investors making timely decisions. However, in the Colombo Stock Exchange, I see a minimal number of such research carried out, probably due to the less demand. I see this as an opportunity.

### 5.3.1 Researches on Financial Market decisions.

Investing in a financial asset is not a single decision. In maintaining a financial asset portfolio, hundreds of factors, considerations, and reductions are made before investing in a financial asset.

- Deciding the asset class invest on based on market conditions. ( the real estate, stocks, bonds )
- Shortlisting assets to pay close attention.
- Selecting the asset from the pool to invest at a give moment.
- Right date/time to enter or exit the market.

- 
- Bid price/ask price to maximize the gain.
  - Predicting market crashes.
  - etc.

Each of these decisions contain many factors considered, where evaluating them and automating require their own researches.

# Bibliography

- Basak, Suryoday and Kar, Saibal and Saha, Snehanshu and Khaidem, Luckyson and Dey, Sudeepa Roy (2019). Predicting the direction of stock market prices using tree-based classifiers, *The North American Journal of Economics and Finance* **47**: 552–567. [11](#)
- Borovkova, Svetlana and Tsiamas, Ioannis (2019). An ensemble of LSTM neural networks for high-frequency stock market classification, *Journal of Forecasting* **38**(6). [10](#)
- Chang, Pei-Chann and Fan, Chin-Yuan and Liu, Chen-Hao (2009). Integrating a Piecewise Linear Representation Method and a Neural Network Model for Stock Trading Points Prediction, *IEEE Transactions on Systems, Man, and Cybernetics, Part C* **39**: 80–92. [10](#)
- Dash, Rajashree and Dash, Pradipta Kishore (2016). A hybrid stock trading framework integrating technical analysis with machine learning techniques, *The Journal of Finance and Data Science* **2**(1): 42–57. [10](#)
- Fama, Eugene F. (1965). Random Walks in Stock Market Prices, *Financial Analysts Journal* **51**(1): 75–80. Publisher: CFA Institute. [8](#)
- Fama, Eugene F. (1970). Efficient Capital Markets: A Review of Theory and Empirical Work, *The Journal of Finance* **25**(2): 383–417. Publisher: [American Finance Association, Wiley]. [8](#)
- Fernando, Pnd (2014). Profitability of technical trading strategies in emerging Sri Lankan stock market, *Kelaniya Journal of Management* **2**(2): 32. [9](#)

- Gao, Xiang (2018). Deep reinforcement learning for time series: playing idealized trading games. [11](#), [20](#)
- Lee, Jae Won and Park, Jonghun and Jangmin and Lee, Jongwoo and Hong, Euyseok (2007). A Multiagent Approach to Q-Learning for Daily Stock Trading, *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans* **37**(6): 864 – 877. [11](#), [12](#)
- Murphy, John J. (1996). *The visual investor: how to spot market trends*, John Wiley, New York. [8](#)
- Murphy, John J. (1999). *Technical Analysis of the Financial Markets: A Comprehensive Guide to Trading Methods and Applications*, SUB UPD EX edition edn, New York Institute of Finance, New York. [8](#), [9](#)
- Muruganandan, S. (2020). Testing the Profitability of Technical Trading Rules across Market Cycles: Evidence from India, *Colombo Business Journal* **11**(1): 24. [9](#)
- Rodríguez-González, Alejandro and Guadrás-Iglesias, Fernando and Colomo-Palacios, Ricardo and Gomez-Berbis, Juan Miguel and Jimenez-Domingo, Enrique and Alor-Hernandez, Giner and Posada-Gomez, Rubén and Cortes-Robles, Guillermo (2010). Improving Trading Systems Using the RSI Financial Indicator and Neural Networks, *in* Hutchison, David and Kanade, Takeo and Kittler, Josef and Kleinberg, Jon M. and Mattern, Friedemann and Mitchell, John C. and Naor, Moni and Nierstrasz, Oscar and Pandu Rangan, C. and Steffen, Bernhard and Sudan, Madhu and Terzopoulos, Demetri and Tygar, Doug and Vardi, Moshe Y. and Weikum, Gerhard and Kang, Byeong-Ho and Richards, Debbie (ed.), *Knowledge Management and Acquisition for Smart Systems and Services*, Vol. 6232, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 27–37. Series Title: Lecture Notes in Computer Science. [10](#)
- Shynkevich, Yauheniya and McGinnity, T.M. and Coleman, Sonya A. and Belatreche, Ammar and Li, Yuhua (2017). Forecasting price movements us-

## BIBLIOGRAPHY

---

- ing technical indicators: Investigating the impact of varying input window length, *Neurocomputing* **264**: 71–88. [10](#)
- Tay, Francis E. H and Cao, Lijuan (2001). Application of support vector machines in financial time series forecasting, *Omega* **29**(4): 309–317. [9](#)
- Tay, Francis E. H and Cao, Lijuan (2003). Support vector machine with adaptive parameters in financial time series forecasting, *IEEE Transactions on Neural Networks* **14**(6): 1506–1518. [9](#)

# Appendix A : The Source Code

agents.py

---

```
from lib import *
import tensorflow as tf

class Agent:

    def __init__(self, model,
                 batch_size=32, discount_factor=0.95):

        self.model = model
        self.batch_size = batch_size
        self.discount_factor = discount_factor
        self.memory = []

    def remember(self, state, action, reward, next_state, done,
                next_valid_actions):
        self.memory.append((state, action, reward, next_state, done,
                            next_valid_actions))
```

---

```

def replay(self):
    batch = random.sample(self.memory, min(len(self.memory),
        self.batch_size))
    for state, action, reward, next_state, done, next_valid_actions
        in batch:
        q = reward
        if not done:
            q += self.discount_factor *
                np.nanmax(self.get_q_valid(next_state,
                    next_valid_actions))
        self.model.fit(state, action, q)

def get_q_valid(self, state, valid_actions):
    q = self.model.predict(state)
    q_valid = [np.nan] * len(q)
    for action in valid_actions:
        q_valid[action] = q[action]
    return q_valid

def act(self, state, exploration, valid_actions):
    if np.random.random() > exploration:
        q_valid = self.get_q_valid(state, valid_actions)
        if np.nanmin(q_valid) != np.nanmax(q_valid):
            return np.nanargmax(q_valid)
    return random.sample(valid_actions, 1)[0]

```



---

```
def save(self, fld):
    makedirs(fld)

    attr = {
        'batch_size':self.batch_size,
        'discount_factor':self.discount_factor,
        #'memory':self.memory
    }

    pickle.dump(attr, open(os.path.join(fld,
        'agent_attr.pickle'),'wb'))
    self.model.save(fld)

def load(self, fld):
    path = os.path.join(fld, 'agent_attr.pickle')
    print(path)
    attr = pickle.load(open(path,'rb'))
    for k in attr:
        setattr(self, k, attr[k])
    self.model.load(fld)

def add_dim(x, shape):
    return np.reshape(x, (1,) + shape)
```

---

```
class QModelKeras:
    # ref: https://keon.io/deep-q-learning/

    def init(self):
        pass

    def build_model(self):
        pass

    def __init__(self, state_shape, n_action):
        self.state_shape = state_shape
        self.n_action = n_action
        self.attr2save = ['state_shape', 'n_action', 'model_name']
        self.init()

    def save(self, fld):
        mkdirs(fld)

        with open(os.path.join(fld, 'model.json'), 'w') as json_file:
            json_file.write(self.model.to_json())

        self.model.save_weights(os.path.join(fld, 'weights.hdf5'))

        attr = dict()
        for a in self.attr2save:
```

---

```
        attr[a] = getattr(self, a)
pickle.dump(attr, open(os.path.join(fld,
        'Qmodel_attr.pickle'),'wb'))

def load(self, fld, learning_rate):
    json_str = open(os.path.join(fld, 'model.json')).read()
    self.model = keras.models.model_from_json(json_str)
    self.model.load_weights(os.path.join(fld, 'weights.hdf5'))
    self.model.compile(loss='mse',
        optimizer=keras.optimizers.Adam(lr=learning_rate))

    attr = pickle.load(open(os.path.join(fld, 'Qmodel_attr.pickle'),
        'rb'))

    for a in attr:
        setattr(self, a, attr[a])

def predict(self, state):
    q = self.model.predict(
        add_dim(state, self.state_shape)
    )[0]

    if np.isnan(max(q)):
        print('state'+str(state))
        print('q'+str(q))
        raise ValueError
```

---

```
    return q

def fit(self, state, action, q_action):
    q = self.predict(state)
    q[action] = q_action

    self.model.fit(
        add_dim(state, self.state_shape),
        add_dim(q, (self.n_action,)),
        epochs=1, verbose=0)

class QModelMLP(QModelKeras):
    # multi-layer perception (MLP), i.e., dense only

    def init(self):
        self.qmodel = 'MLP'

    def build_model(self, n_hidden, learning_rate, activation='relu'):

        model = keras.models.Sequential()
        model.add(keras.layers.Reshape(
            (self.state_shape[0]*self.state_shape[1],),
            input_shape=self.state_shape))
```

---

```

for i in range(len(n_hidden)):
    model.add(keras.layers.Dense(n_hidden[i],
        activation=activation))
    #model.add(keras.layers.Dropout(drop_rate))

model.add(keras.layers.Dense(self.n_action, activation='linear'))
model.compile(loss='mse',
    optimizer=keras.optimizers.Adam(lr=learning_rate))
self.model = model
self.model_name = self.qmodel + str(n_hidden)

class QModelRNN(QModelKeras):
    """
    https://keras.io/getting-started/sequential-model-guide/#example
    note param doesn't grow with len of sequence
    """

    def _build_model(self, Layer, n_hidden, dense_units, learning_rate,
        activation='relu'):

        model = keras.models.Sequential()
        model.add(keras.layers.Reshape(self.state_shape,
            input_shape=self.state_shape))
        m = len(n_hidden)

```

---

```

for i in range(m):
    model.add(Layer(n_hidden[i],
                    return_sequences=(i<m-1)))
for i in range(len(dense_units)):
    model.add(keras.layers.Dense(dense_units[i],
                                  activation=activation))
model.add(keras.layers.Dense(self.n_action, activation='linear'))
model.compile(loss='mse',
              optimizer=keras.optimizers.Adam(lr=learning_rate))
self.model = model
self.model_name = self.qmodel + str(n_hidden) + str(dense_units)

```

```

class QModelLSTM(QModelRNN):
    def init(self):
        self.qmodel = 'LSTM'
    def build_model(self, n_hidden, dense_units, learning_rate,
                    activation='relu'):
        Layer = keras.layers.LSTM
        self._build_model(Layer, n_hidden, dense_units, learning_rate,
                           activation)

```

```

class QModelGRU(QModelRNN):
    def init(self):

```

---

```

        self.qmodel = 'GRU'

def build_model(self, n_hidden, dense_units, learning_rate,
                activation='relu'):
    Layer = keras.layers.GRU
    self._build_model(Layer, n_hidden, dense_units, learning_rate,
                      activation)

class QModelConv(QModelKeras):
    """
    ref: https://keras.io/layers/convolutional/
    """
    def init(self):
        self.qmodel = 'Conv'

    def build_model(self,
                    filter_num, filter_size, dense_units,
                    learning_rate, activation='relu', dilation=None, use_pool=None):

        if use_pool is None:
            use_pool = [True]*len(filter_num)
        if dilation is None:
            dilation = [1]*len(filter_num)

    model = keras.models.Sequential()

```

---

```
model.add(keras.layers.Reshape(self.state_shape,
                               input_shape=self.state_shape))

for i in range(len(filter_num)):
    model.add(keras.layers.Conv1D(filter_num[i],
                                   kernel_size=filter_size[i], dilation_rate=dilation[i],
                                   activation=activation, use_bias=True))
    if use_pool[i]:
        model.add(keras.layers.MaxPooling1D(pool_size=2))

model.add(keras.layers.Flatten())
for i in range(len(dense_units)):
    model.add(keras.layers.Dense(dense_units[i],
                                  activation=activation))
model.add(keras.layers.Dense(self.n_action, activation='linear'))
model.compile(loss='mse',
              optimizer=keras.optimizers.Adam(lr=learning_rate))

self.model = model

self.model_name = self.qmodel + str([a for a in
                                     zip(filter_num, filter_size, dilation, use_pool)
                                     ])+ ' + '+str(dense_units)
```



---

```

class QModelConvRNN(QModelKeras):
    """
    https://keras.io/getting-started/sequential-model-guide/#example
    note param doesn't grow with len of sequence
    """

    def _build_model(self, RNNLayer, conv_n_hidden, RNN_n_hidden,
                     dense_units, learning_rate,
                     conv_kernel_size=3, use_pool=False, activation='relu'):

        model = keras.models.Sequential()
        model.add(keras.layers.Reshape(self.state_shape,
                                       input_shape=self.state_shape))

        for i in range(len(conv_n_hidden)):
            model.add(keras.layers.Conv1D(conv_n_hidden[i],
                                          kernel_size=conv_kernel_size,
                                          activation=activation, use_bias=True))
            if use_pool:
                model.add(keras.layers.MaxPooling1D(pool_size=2))
        m = len(RNN_n_hidden)
        for i in range(m):
            model.add(RNNLayer(RNN_n_hidden[i],
                              return_sequences=(i<m-1)))
        for i in range(len(dense_units)):
            model.add(keras.layers.Dense(dense_units[i],

```

---

```

        activation=activation))

model.add(keras.layers.Dense(self.n_action, activation='linear'))
model.compile(loss='mse',
              optimizer=keras.optimizers.Adam(lr=learning_rate))
self.model = model
self.model_name = self.qmodel + str(conv_n_hidden) +
                  str(RNN_n_hidden) + str(dense_units)

class QModelConvLSTM(QModelConvRNN):
    def init(self):
        self.qmodel = 'ConvLSTM'
    def build_model(self, conv_n_hidden, RNN_n_hidden, dense_units,
                   learning_rate,
                   conv_kernel_size=3, use_pool=False, activation='relu'):
        Layer = keras.layers.LSTM
        self._build_model(Layer, conv_n_hidden, RNN_n_hidden,
                          dense_units, learning_rate,
                          conv_kernel_size, use_pool, activation)

class QModelConvGRU(QModelConvRNN):
    def init(self):
        self.qmodel = 'ConvGRU'
    def build_model(self, conv_n_hidden, RNN_n_hidden, dense_units,

```

---

```
    learning_rate,  
    conv_kernel_size=3, use_pool=False, activation='relu'):  
    Layer = keras.layers.GRU  
    self._build_model(Layer, conv_n_hidden, RNN_n_hidden,  
                      dense_units, learning_rate,  
                      conv_kernel_size, use_pool, activation)
```

```
def load_model(fld, learning_rate):  
    s = open(os.path.join(fld, 'QModel.txt'), 'r').read().strip()  
    qmodels = {  
        'Conv': QModelConv,  
        'DenseOnly': QModelMLP,  
        'MLP': QModelMLP,  
        'LSTM': QModelLSTM,  
        'GRU': QModelGRU,  
    }  
    qmodel = qmodels[s](None, None)  
    qmodel.load(fld, learning_rate)  
    return qmodel
```

---

main.py

---

---

```
#!/usr/bin/env python2

from lib import *
from sampler import *

from agents import *
from emulator import *
from simulators import *
from emulator2 import RandomMovingMarket
from visualizer import *
from company_data_sampler import CompanyDataSampler
from properties import *

# import cProfile, pstats, io

def get_model(model_type, env, learning_rate, fld_load):

    print_t = False
    exploration_init = 1.

    if model_type == 'MLP':
        m = 16
        layers = 5
        hidden_size = [m]*layers
```

---

```

model = QModelMLP(env.state_shape, env.n_action)
model.build_model(hidden_size, learning_rate=learning_rate,
                  activation='tanh')

elif model_type == 'conv':

    m = 16
    layers = 2
    filter_num = [m]*layers
    filter_size = [3] * len(filter_num)
    #use_pool = [False, True, False, True]
    #use_pool = [False, False, True, False, False, True]
    use_pool = None
    #dilation = [1,2,4,8]
    dilation = None
    dense_units = [48,24]
    model = QModelConv(env.state_shape, env.n_action)
    model.build_model(filter_num, filter_size, dense_units,
                    learning_rate,
                    dilation=dilation, use_pool=use_pool)

elif model_type == 'RNN-LSTM':

    m = 32
    layers = 3
    hidden_size = [m]*layers
    dense_units = [m,m]

```

---

```
model = QModelLSTM(env.state_shape, env.n_action)
model.build_model(hidden_size, dense_units,
                  learning_rate=learning_rate)
print_t = True

elif model_type == 'RNN-GRU':
    m = 32
    layers = 3
    hidden_size = [m]*layers
    dense_units = [m,m]
    model = QModelGRU(env.state_shape, env.n_action)
    model.build_model(hidden_size, dense_units,
                    learning_rate=learning_rate)
    print_t = True

elif model_type == 'ConvRNN':

    m = 8
    conv_n_hidden = [m,m]
    RNN_n_hidden = [m,m]
    dense_units = [m,m]
    model = QModelConvGRU(env.state_shape, env.n_action)
    model.build_model(conv_n_hidden, RNN_n_hidden, dense_units,
                    learning_rate=learning_rate)
    print_t = True
```

---

```
elif model_type == 'pretrained':
    agent.model = load_model(fld_load, learning_rate)

else:
    raise ValueError

return model, print_t

def main():

    """
    it is recommended to generate database usng sampler.py before run
    main
    """

    exploration_init = 1.; fld_load = None

    open_cost = 3.3
    #db_type = 'SinSamplerDB'; db = 'concat_half_base_'; Sampler =
        SinSampler
    db_type = 'PairSamplerDB'; db = 'randjump_100,1(10, 30)[_]
    batch_size = 8
    learning_rate = 1e-4
    discount_factor = 0.8
    exploration_decay = 0.99
```

---

```

exploration_min = 0.01
window_state = 14

fld = os.path.join '..', 'data', db_type, db+'A')
# sampler = Sampler('load', fld=fld)
sampler = CompanyDataSampler(company_code=company_code)
# env = RandomMovingMarket(sampler, window_state, open_cost,
    considering_t_max= sample_count)
env = Market(sampler, window_state, open_cost)
model, print_t = get_model(model_type, env, learning_rate, fld_load)
model.model.summary()

#return

agent = Agent(model, discount_factor=discount_factor,
    batch_size=batch_size)
visualizer = Visualizer(env.action_labels)

fld_save = os.path.join(OUTPUT_FLD, sampler.title, model.model_name,
    str((env.window_state, sampler.window_episode, agent.batch_size,
        learning_rate,
        agent.discount_factor, exploration_decay, env.open_cost)))

print('='*20)
print(fld_save)
print('='*20)

```



---

```
simulator = Simulator(agent, env, visualizer=visualizer,
    fld_save=fld_save)
simulator.train(n_episode_training, save_per_episode=1,
    exploration_decay=exploration_decay,
    exploration_min=exploration_min, print_t=print_t,
    exploration_init=exploration_init)
#agent.model = load_model(os.path.join(fld_save,'model'),
    learning_rate)

#print('='*20+'\nin-sample testing\n'+'*20)
simulator.test(n_episode_testing, save_per_episode=1,
    subfld='in-sample testing')

"""
fld = os.path.join('data',db_type,db+'B')
sampler = SinSampler('load',fld=fld)
simulator.env.sampler = sampler
simulator.test(n_episode_testing, save_per_episode=1,
    subfld='out-of-sample testing')
"""

if __name__ == '__main__':
    main()
    # profile()
```

---

emulator.py

---

---

```
from lib import *
```

```
# by Xiang Gao, 2018
```

```
def find_ideal(p, just_once):
```

```
    if not just_once:
```

```
        diff = np.array(p[1:]) - np.array(p[:-1])
```

```
        return sum(np.maximum(np.zeros(diff.shape), diff))
```

```
    else:
```

```
        best = 0.
```

```
        i0_best = None
```

```
        for i in range(len(p)-1):
```

```
            best = max(best, max(p[i+1:]) - p[i])
```

```
        return best
```

```
class Market:
```

```
    """
```

```
    state          MA of prices, normalized using values at t
```

```
                   ndarray of shape (window_state, n_instruments * n_MA),
```

```
                   i.e., 2D
```

```
                   which is self.state_shape
```

---

```

action          three action
                0: empty, don't open/close.
                1: open a position
                2:  keep a position
"""

def reset(self, rand_price=True):
    self.empty = True
    if rand_price:
        prices, self.title = self.sampler.sample()
        price = np.reshape(prices[:,0], prices.shape[0])

        self.prices = prices.copy()
        self.price = price/price[0]*100
        self.t_max = len(self.price) - 1

    self.max_profit = find_ideal(self.price[self.t0:], False)
    self.t = self.t0

    return self.get_state(), self.get_valid_actions()

def get_state(self, t=None):
    if t is None:
        t = self.t

    state = self.prices[t - self.window_state + 1: t + 1, :].copy()

```

---

```
for i in range(self.sampler.n_var):
    norm = np.mean(state[:,i])
    if norm != 0:
        state[:,i] = (state[:,i]/norm - 1.)*100
return state

def get_valid_actions(self):
    if self.empty:
        return [0, 1] # wait, open
    else:
        return [2, 3] # close, keep

def get_noncash_reward(self, t=None, empty=None):
    if t is None:
        t = self.t
    if empty is None:
        empty = self.empty
    reward = self.direction * (self.price[t+1] - self.price[t])
    if empty:
        reward -= self.open_cost
    if reward < 0:
        reward *= (1. + self.risk_averse)
    return reward

def calc_reward(self,t=None):
```

---

```
    if t is None:
        t = self.t
    return (self.price[t+1] - self.price[t])

def calc_commission(self, t=None, commission = 0.00134):
    if t is None:
        t = self.t
    return (self.price[t] * commission)

def step(self, action):

    done = False

    if action == 0:    # wait-stay
        reward = 0.
        self.empty = True
    elif action == 1: # buy
        reward = self.calc_reward() - self.calc_commission()
        self.empty = False
    elif action == 2: # holding - stay
        reward = self.calc_reward()
        self.empty = False
    elif action == 3: # sell
        reward = -self.calc_commission()
        self.empty = True
    else:
        raise ValueError('no such action: '+str(action))
```

---

```
self.t += 1
return self.get_state(), reward, self.t == self.t_max,
       self.get_valid_actions()

def __init__(self,
            sampler, window_state, open_cost,
            direction=1., risk_averse=0.):

    self.sampler = sampler
    self.window_state = window_state
    self.open_cost = open_cost
    self.direction = direction
    self.risk_averse = risk_averse

    self.n_action = 4
    self.state_shape = (window_state, self.sampler.n_var)
    self.action_labels = ['wait-stay', 'buy', 'holding - stay', 'sell' ]
    self.t0 = window_state - 1

if __name__ == '__main__':
    test_env()
```

---

emulator2.py

---



---

```

from lib import *

# by Xiang Gao, 2018
from emulator import Market, find_ideal
import random

class RandomMovingMarket(Market):
    """
    state      MA of prices, normalized using values at t
               ndarray of shape (window_state, n_instruments *
               n_MA), i.e., 2D
               which is self.state_shape

    action     three action
               0: empty, don't open/close.
               1: open a position
               2: keep a position
    """

    def reset(self, rand_price=True):
        self.empty = True
        if rand_price:
            prices, self.title = self.sampler.sample()
            price = np.reshape(prices[:, 0], prices.shape[0])

```

---

```

        self.prices = prices.copy()
        self.price = price / price[0] * 100
        self.t_max = self.considering_t_max
        id_range = [*range(self.t0, len(self.price) - 1, 1)]
        self.working_sample = random.sample(id_range, self.t_max+1)

self.max_profit = find_ideal(self.price[self.t0:], False)
self.t = 0

return self.get_state(), self.get_valid_actions()

def get_state(self, t=None):
    if t is None:
        t = self.t
    t_ = self.working_sample[t]
    state = self.prices[t_ - self.window_state + 1: t_ + 1,
        :].copy()
    for i in range(self.sampler.n_var):
        norm = np.mean(state[:, i])
        if norm != 0:
            state[:, i] = (state[:, i] / norm - 1.) * 100
    return state

def __init__(self,

```



---

```
        sampler, window_state, open_cost,
        direction=1., risk_averse=0., considering_t_max=800):
self.sampler = sampler
self.window_state = window_state
self.open_cost = open_cost
self.direction = direction
self.risk_averse = risk_averse

self.n_action = 4
self.state_shape = (window_state, self.sampler.n_var)
self.action_labels = ['wait-stay', 'buy', 'holding - stay',
                      'sell']
self.t0 = window_state - 1
self.considering_t_max = considering_t_max

if __name__ == '__main__':
    test_env()
```

---

lib.py

---

```
import random, os, datetime, pickle, json, sys
from tensorflow import keras
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
```

---

```
OUTPUT_FLD = os.path.join '..', 'results')
PRICE_FLD = '/Users/xianggao/Dropbox/distributed/code_db/price
            coinbase/vm-w7r-db'

def makedirs(fld):
    if not os.path.exists(fld):
        os.makedirs(fld)
```

---

sampler.py

---

```
from lib import *

def read_data(date, instrument, time_step):
    path = os.path.join(PRICE_FLD, date, instrument+'.csv')
    if not os.path.exists(path):
        print('no such file: '+path)
        return None

    df_raw = pd.read_csv(path, parse_dates=['time'], index_col='time')
    df = df_raw.resample(time_step, how='last').fillna(method='ffill')
    return df['spot'].values

class Sampler:

    def load_db(self, fld):
```

---

```

self.db = pickle.load(open(os.path.join(fld, 'db.pickle'),'rb'))
param = json.load(open(os.path.join(fld, 'param.json'),'rb'))
self.i_db = 0
self.n_db = param['n_episodes']
self.sample = self.__sample_db
for attr in param:
    if hasattr(self, attr):
        setattr(self, attr, param[attr])
self.title = 'DB_'+param['title']

def build_db(self, n_episodes, fld):
    db = []
    for i in range(n_episodes):
        prices, title = self.sample()
        db.append((prices, "[%i]_%i+title))
    os.makedirs(fld) # don't overwrite existing fld
    pickle.dump(db, open(os.path.join(fld, 'db.pickle'),'wb'))
    param = {'n_episodes':n_episodes}
    for k in self.attrs:
        param[k] = getattr(self, k)
    json.dump(param, open(os.path.join(fld, 'param.json'),'w'))

def __sample_db(self):

```

---

```
prices, title = self.db[self.i_db]
self.i_db += 1
if self.i_db == self.n_db:
    self.i_db = 0
return prices, title
```

```
class PairSampler(Sampler):
```

```
    def __init__(self, game,
                 window_episode=None, forecast_horizon_range=None,
                 max_change_perc=10., noise_level=10., n_section=1,
                 fld=None, windows_transform=[]):

        self.window_episode = window_episode
        self.forecast_horizon_range = forecast_horizon_range
        self.max_change_perc = max_change_perc
        self.noise_level = noise_level
        self.n_section = n_section
        self.windows_transform = windows_transform
        self.n_var = 2 + len(self.windows_transform) # price, signal

        self.attrs = ['title', 'window_episode', 'forecast_horizon_range',
                     'max_change_perc', 'noise_level', 'n_section', 'n_var']
        param_str = str((self.noise_level, self.forecast_horizon_range,
```

---

```

        self.n_section, self.windows_transform))

if game == 'load':
    self.load_db(fld)
elif game in ['randwalk', 'randjump']:
    self.__rand = getattr(self, '_PairSampler_'+game)
    self.sample = self.__sample
    self.title = game + param_str
else:
    raise ValueError

def __randwalk(self, l):
    change = (np.random.random(1 + self.forecast_horizon_range[1]) -
              0.5) * 2 * self.max_change_perc/100
    forecast_horizon =
        random.randrange(self.forecast_horizon_range[0],
                        self.forecast_horizon_range[1])
    return change[:l], change[forecast_horizon: forecast_horizon +
                                l], forecast_horizon

def __randjump(self, l):
    change = [0.] * (1 + self.forecast_horizon_range[1])
    n_jump = random.randrange(15,30)
    for i in range(n_jump):

```

---

```

        t = random.randrange(len(change))
        change[t] = (np.random.random() - 0.5) * 2 *
            self.max_change_perc/100
forecast_horizon =
    random.randrange(self.forecast_horizon_range[0],
        self.forecast_horizon_range[1])
return change[:l], change[forecast_horizon: forecast_horizon +
    l], forecast_horizon

def __sample(self):

    L = self.window_episode
    if bool(self.windows_transform):
        L += max(self.windows_transform)
    l0 = L/self.n_section
    l1 = L

    d_price = []
    d_signal = []
    forecast_horizon = []

    for i in range(self.n_section):
        if i == self.n_section - 1:
            l = l1

```

---

```

else:
    l = 10
    l1 -= 10

    d_price_i, d_signal_i, horizon_i = self.__rand(1)
    d_price = np.append(d_price, d_price_i)
    d_signal = np.append(d_signal, d_signal_i)
    forecast_horizon.append(horizon_i)

price = 100. * (1. + np.cumsum(d_price))
signal = 100. * (1. + np.cumsum(d_signal)) + \
    np.random.random(len(price)) * self.noise_level

price += (100 - min(price))
signal += (100 - min(signal))

inputs = [price[-self.window_episode:],
          signal[-self.window_episode:]]
for w in self.windows_transform:
    inputs.append(signal[-self.window_episode - w: -w])

return np.array(inputs).T,
       'forecast_horizon='+str(forecast_horizon)

```

---

```

class SinSampler(Sampler):

    def __init__(self, game,
                 window_episode=None, noise_amplitude_ratio=None,
                 period_range=None, amplitude_range=None,
                 fld=None):

        self.n_var = 1 # price only

        self.window_episode = window_episode
        self.noise_amplitude_ratio = noise_amplitude_ratio
        self.period_range = period_range
        self.amplitude_range = amplitude_range
        self.can_half_period = False

        self.attrs = ['title', 'window_episode', 'noise_amplitude_ratio',
                     'period_range', 'amplitude_range', 'can_half_period']

        param_str = str((
            self.noise_amplitude_ratio, self.period_range,
            self.amplitude_range
        ))

        if game == 'single':
            self.sample = self.__sample_single_sin
            self.title = 'SingleSin'+param_str
        elif game == 'concat':

```



---

```

        self.sample = self.__sample_concat_sin
        self.title = 'ConcatSin'+param_str
    elif game == 'concat_half':
        self.can_half_period = True
        self.sample = self.__sample_concat_sin
        self.title = 'ConcatHalfSin'+param_str
    elif game == 'concat_half_base':
        self.can_half_period = True
        self.sample = self.__sample_concat_sin_w_base
        self.title = 'ConcatHalfSin+Base'+param_str
        self.base_period_range = (int(2*self.period_range[1]),
            4*self.period_range[1])
        self.base_amplitude_range = (20,80)
    elif game == 'load':
        self.load_db(fld)
    else:
        raise ValueError

def __rand_sin(self,
    period_range=None, amplitude_range=None,
    noise_amplitude_ratio=None, full_episode=False):

    if period_range is None:
        period_range = self.period_range
    if amplitude_range is None:

```

---

```

    amplitude_range = self.amplitude_range
    if noise_amplitude_ratio is None:
        noise_amplitude_ratio = self.noise_amplitude_ratio

    period = random.randrange(period_range[0], period_range[1])
    amplitude = random.randrange(amplitude_range[0],
        amplitude_range[1])
    noise = noise_amplitude_ratio * amplitude

    if full_episode:
        length = self.window_episode
    else:
        if self.can_half_period:
            length = int(random.randrange(1,4) * 0.5 * period)
        else:
            length = period

    p = 100. + amplitude * np.sin(np.array(range(length)) * 2 *
        3.1416 / period)
    p += np.random.random(p.shape) * noise

    return p, '100+isin((2pi/i)t)+ie'%(amplitude, period, noise)

```

---

```

def __sample_concat_sin(self):
    prices = []
    p = []
    while True:
        p = np.append(p, self.__rand_sin(full_episode=False)[0])
        if len(p) > self.window_episode:
            break
    prices.append(p[:self.window_episode])
    return np.array(prices).T, 'concat sin'

def __sample_concat_sin_w_base(self):
    prices = []
    p = []
    while True:
        p = np.append(p, self.__rand_sin(full_episode=False)[0])
        if len(p) > self.window_episode:
            break
    base, base_title = self.__rand_sin(
        period_range=self.base_period_range,
        amplitude_range=self.base_amplitude_range,
        noise_amplitude_ratio=0.,
        full_episode=True)
    prices.append(p[:self.window_episode] + base)
    return np.array(prices).T, 'concat sin + base: '+base_title

def __sample_single_sin(self):

```

---

```
prices = []
funcs = []
p, func = self.__rand_sin(full_episode=True)
prices.append(p)
funcs.append(func)
return np.array(prices).T, str(funcs)
```

```
def test_SinSampler():
```

```
    window_episode = 180
    window_state = 40
    noise_amplitude_ratio = 0.5
    period_range = (10,40)
    amplitude_range = (5,80)
    game = 'concat_half_base'
    instruments = ['fake']

    sampler = SinSampler(game,
        window_episode, noise_amplitude_ratio, period_range,
        amplitude_range)
    n_episodes = 100
    """
```

---

```

for i in range(100):
    plt.plot(sampler.sample(instruments)[0])
    plt.show()
    """
fld = os.path.join('data', 'SinSamplerDB', game+'_B')
sampler.build_db(n_episodes, fld)

def test_PairSampler():
    fhr = (10,30)
    n_section = 1
    max_change_perc = 30.
    noise_level = 5
    game = 'randjump'
    windows_transform = []

    sampler = PairSampler(game, window_episode=180,
        forecast_horizon_range=fhr,
        n_section=n_section, noise_level=noise_level,
        max_change_perc=max_change_perc,
        windows_transform=windows_transform)

    #plt.plot(sampler.sample()[0]);plt.show()
    #"""
    n_episodes = 100

```

---

```
fld = os.path.join('data', 'PairSamplerDB',
    game+'_%i,%i'%(n_episodes,
        n_section)+str(fhr)+str(windows_transform)+'_B')
sampler.build_db(n_episodes, fld)
#"""
```

```
if __name__ == '__main__':
    #scan_match()
    test_SinSampler()
    #p = [1,2,3,2,1,2,3]
    #print find_ideal(p)
    test_PairSampler()
```

---

visualizer.py - drawing charts

---

```
import pandas as pd
```

```
from lib import *
```

```
def get_tick_labels(bins, ticks):
    ticklabels = []
    for i in ticks:
        if i < len(bins):
```

---

```

        ticklabels.append('%.2f'%(bins[int(i)]))
    else:
        ticklabels.append('%.2f'%(bins[-1])+'+')

    return ticklabels

class Visualizer:

    def __init__(self, action_labels):
        self.n_action = len(action_labels)
        self.action_labels = action_labels

    def plot_a_episode(self,
                      env, model,
                      explored_cum_rewards, explored_actions,
                      safe_cum_rewards, safe_actions,
                      fig_path):

        f, axs = plt.subplots(3, 1, sharex=True, figsize=(14, 14))
        ax_price, ax_action, ax_Q = axs

        ls = ['-', '--']

        for i in range(min(2, env.prices.shape[1])):
            p = env.prices[:, i] / env.prices[0, i] * 100 - 100

```

---

```

ax_price.plot(p, 'k' + ls[i], label='input%i - 100' % i)

ax_price.plot(explored_cum_rewards, 'b', label='explored P&L')
ax_price.plot(safe_cum_rewards, 'r', label='safe P&L')
ax_price.legend(loc='best', frameon=False)
ax_price.set_title(env.title + ', ideal: %.1f, safe: %.1f,
    explored: %.1f' % (
    env.max_profit, safe_cum_rewards[-1],
    explored_cum_rewards[-1]))

ax_action.plot(explored_actions, 'b', label='explored')
ax_action.plot(safe_actions, 'r', label='safe', linewidth=2)
ax_action.set_ylim(-0.4, self.n_action - 0.6)
ax_action.set_ylabel('action')
ax_action.set_yticks(range(self.n_action))
ax_action.legend(loc='best', frameon=False)

style = ['k', 'r', 'b', 'c']
qq = []
for t in range(env.t0):
    qq.append([np.nan] * self.n_action)
for t in range(env.t0, env.t_max):
    qq.append(model.predict(env.get_state(t)))
for i in range(self.n_action):
    ax_Q.plot([float(qq[t][i]) for t in range(len(qq))],
        style[i], label=self.action_labels[i])

```



---

```
ax_Q.set_ylabel('Q')
ax_Q.legend(loc='best', frameon=False)
ax_Q.set_xlabel('t')

plt.subplots_adjust(wspace=0.4)
plt.savefig(fig_path)
plt.close()

def plot_episodes(self,
                  explored_total_rewards, safe_total_rewards,
                  explorations,
                  fig_path, MA_window=100):

    f = plt.figure(figsize=(14, 10)) # width, height in inch (100
        pixel)
    if explored_total_rewards is None:
        f, ax_reward = plt.subplots()
    else:
        figshape = (3, 1)
        ax_reward = plt.subplot2grid(figshape, (0, 0), rowspan=2)
        ax_exploration = plt.subplot2grid(figshape, (2, 0),
            sharex=ax_reward)

    tt = range(len(safe_total_rewards))

    if explored_total_rewards is not None:
```

---

```

pma = pd.Series(np.array(explored_total_rewards))
ma = pma.rolling(window=MA_window, min_periods=1).median()
std = pma.rolling(window=MA_window, min_periods=3).std()
ax_reward.plot(tt, explored_total_rewards, 'bv',
               fillstyle='none')
ax_reward.plot(tt, ma, 'b', label='explored ma', linewidth=2)
ax_reward.plot(tt, std, 'b--', label='explored std',
               linewidth=2)

series = pd.Series(np.array(safe_total_rewards))
ma = series.rolling(window=MA_window, min_periods=1).median()
std = series.rolling(window=MA_window, min_periods=1).std()
ax_reward.plot(tt, safe_total_rewards, 'ro', fillstyle='none')
ax_reward.plot(tt, ma, 'r', label='safe ma', linewidth=2)
ax_reward.plot(tt, std, 'r--', label='safe std', linewidth=2)

ax_reward.axhline(y=0, color='k', linestyle=':')
#ax_reward.axhline(y=60, color='k', linestyle=':')
ax_reward.set_ylabel('total reward')
ax_reward.legend(loc='best', frameon=False)
ax_reward.yaxis.tick_right()
ylim = ax_reward.get_ylim()
ax_reward.set_ylim((max(-100,ylim[0]), min(100,ylim[1])))

if explored_total_rewards is not None:
    ax_exploration.plot(tt, np.array(explorations)*100., 'k')

```

---

```
ax_exploration.set_ylabel('exploration')
ax_exploration.set_xlabel('episode')

plt.savefig(fig_path)
plt.close()

def test_visualizer():

    f = plt.figure(figsize=(5,8))
    axs_action = []
    ncol = 3
    nrow = 2

    clim = (0,1)

    ax = plt.subplot2grid((nrow, ncol), (0,ncol-1))
    ax.matshow(np.random.random((2,2)), cmap='RdYlBu_r', clim=clim)

    for action in range(3):
        row = 1 + action/ncol
        col = action%ncol
        ax = plt.subplot2grid((nrow, ncol), (row,col))
        cax = ax.matshow(np.random.random((2,2)), cmap='RdYlBu_r',
```

---

```
        clim=clim)

ax = plt.subplot2grid((nrow, ncol), (0,0), colspan=ncol-1)
cbar = f.colorbar(cax, ax=ax)

plt.show()
```

```
class VisualizerSequential:

    def config(self):
        pass

    def __init__(self, model):
        self.model = model
        self.layers = []
        for layer in self.model.layers:
            self.layers.append(str(layer.name))

        self.inter_models = dict()
        model_input = self.model.input
        for layer in self.layers:
            self.inter_models[layer] = keras.models.Model(
                inputs=model_input,
```

---

```

        outputs=self.model.get_layer(layer).output)

self.config()

class VisualizerConv1D(VisualizerSequential):

    def config(self):

        self.n_channel = self.model.input.shape[2]
        n_col = self.n_channel
        for layer in self.layers:
            shape = self.inter_models[layer].output.shape
            if len(shape) == 3:
                n_col = max(n_col, shape[2])

        self.figshape = (len(self.layers)+1, int(n_col))

    def plot(self, x):

        f = plt.figure(figsize=(30, 30))

        for i in range(self.n_channel):
            ax = plt.subplot2grid(self.figshape, (0, i))
            ax.plot(x[0, :, i], '.-')
            ax.set_title('input, channel %i' % i)

```

---

```
for i_layer in range(len(self.layers)):
    layer = self.layers[i_layer]
    z = self.inter_models[layer].predict(x)
    print('plotting ' + layer)
    if len(z.shape) == 3:
        for i in range(z.shape[2]):
            ax = plt.subplot2grid(self.figshape, (i_layer+1, i))
            ax.plot(z[0,:,i], '-.')
            ax.set_title(layer+' filter %i'%i)
    else:
        ax = plt.subplot2grid(self.figshape, (i_layer+1, 0))
        ax.plot(z[0,:], '-.')
        ax.set_title(layer)

ax.set_ylim(-100,100)

def print_w(self):
    layer = self.layers[0]
    ww = self.inter_models[layer].get_weights()
    for w in ww:
        print(w.shape)
        print(w)
```

---

simulators.py

---

---

```
from lib import *

class Simulator:

    def play_one_episode(self, exploration, training=True,
                        rand_price=True, print_t=False):

        state, valid_actions = self.env.reset(rand_price=rand_price)
        done = False
        env_t = 0

        try:
            env_t = self.env.t
        except AttributeError:
            pass

        cum_rewards = [np.nan] * env_t
        actions = [np.nan] * env_t
        states = [None] * env_t
        prev_cum_rewards = 0.

        while not done:
            if print_t:
                print(self.env.t)
```

---

```
    action = self.agent.act(state, exploration, valid_actions)
    next_state, reward, done, valid_actions = self.env.step(action)

    cum_rewards.append(prev_cum_rewards+reward)
    prev_cum_rewards = cum_rewards[-1]
    actions.append(action)
    states.append(next_state)

    if training:
        self.agent.remember(state, action, reward, next_state,
                            done, valid_actions)
        self.agent.replay()

    state = next_state

return cum_rewards, actions, states

def train(self, n_episode,
          save_per_episode=10, exploration_decay=0.995,
          exploration_min=0.01, print_t=False, exploration_init=1.):

    fld_model = os.path.join(self.fld_save, 'model')
    mkdirs(fld_model) # don't overwrite if already exists
    with open(os.path.join(fld_model, 'QModel.txt'), 'w') as f:
```



---

```

        f.write(self.agent.model.qmodel)

exploration = exploration_init
fld_save = os.path.join(self.fld_save, 'training')

mkdirs(fld_save)
MA_window = 100    # MA of performance
safe_total_rewards = []
explored_total_rewards = []
explorations = []
path_record = os.path.join(fld_save, 'record.csv')

with open(path_record, 'w') as f:
    f.write('episode,game,exploration,explored,safe,MA_explored,MA_safe\n')

for n in range(n_episode):

    print('\ntraining...')
    exploration = max(exploration_min, exploration *
        exploration_decay)
    explorations.append(exploration)
    explored_cum_rewards, explored_actions, _ =
        self.play_one_episode(exploration, print_t=print_t)
    explored_total_rewards.append(100.*explored_cum_rewards[-1]/self.env.max_profi
    safe_cum_rewards, safe_actions, _ = self.play_one_episode(0,
        training=False, rand_price=False, print_t=False)

```

---

```

safe_total_rewards.append(100.*safe_cum_rewards[-1]/self.env.max_profit)

MA_total_rewards =
    np.median(explored_total_rewards[-MA_window:])
MA_safe_total_rewards =
    np.median(safe_total_rewards[-MA_window:])

ss = [
    str(n), self.env.title.replace(',',';'),
        '%.1f'%(exploration*100.),
    '%.1f'%(explored_total_rewards[-1]),
        '%.1f'%(safe_total_rewards[-1]),
    '%.1f'%MA_total_rewards, '%.1f'%MA_safe_total_rewards,
    ]

with open(path_record,'a') as f:
    f.write(','.join(ss)+'\n')
    print('\t'.join(ss))

if n%save_per_episode == 0:
    print('saving results...')
    self.agent.save(fld_model)

self.visualizer.plot_a_episode(

```

---

```

        self.env, self.agent.model,
        explored_cum_rewards, explored_actions,
        safe_cum_rewards, safe_actions,
        os.path.join(fld_save, 'episode_%i.png'%(n)))

    self.visualizer.plot_episodes(
        explored_total_rewards, safe_total_rewards, explorations,
        os.path.join(fld_save, 'total_rewards.png'),
        MA_window)

def test(self, n_episode, save_per_episode=10, subfld='testing'):

    fld_save = os.path.join(self.fld_save, subfld)
    makedirs(fld_save)
    MA_window = 100    # MA of performance
    safe_total_rewards = []
    path_record = os.path.join(fld_save, 'record.csv')

    with open(path_record, 'w') as f:
        f.write('episode,game,pnl,rel,MA\n')

    for n in range(n_episode):
        print('\ntesting...')
```

---

```

safe_cum_rewards, safe_actions, _ = self.play_one_episode(0,
    training=False, rand_price=True)
safe_total_rewards.append(100.*safe_cum_rewards[-1]/self.env.max_profit)
MA_safe_total_rewards =
    np.median(safe_total_rewards[-MA_window:])
ss = [str(n), self.env.title.replace(',',';'),
    '%.1f'%(safe_cum_rewards[-1]),
    '%.1f'%(safe_total_rewards[-1]),
    '%.1f'%MA_safe_total_rewards]

with open(path_record,'a') as f:
    f.write(','.join(ss)+'\n')
    print('\t'.join(ss))

if n%save_per_episode == 0:
    print('saving results...')

self.visualizer.plot_a_episode(
    self.env, self.agent.model,
    [np.nan]*len(safe_cum_rewards),
    [np.nan]*len(safe_actions),
    safe_cum_rewards, safe_actions,
    os.path.join(fld_save, 'episode_%i.png'%(n)))

```

---

```
        self.visualizer.plot_episodes(
            None, safe_total_rewards, None,
            os.path.join(fld_save, 'total_rewards.png'),
            MA_window)

def __init__(self, agent, env,
            visualizer, fld_save):

    self.agent = agent
    self.env = env
    self.visualizer = visualizer
    self.fld_save = fld_save

if __name__ == '__main__':
    #print 'episode%i, init%i'%(1,2)
    a = [1,2,3]
    print(np.mean(a[-100:]))
```

---

company\_data\_sampler.py

---

---

```
from sampler import Sampler
import copy
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler

sample_size=-1

def get_data_file_path(company_code):
    return "../data/" + company_code + ".csv"

def load_data(file_path):
    data = pd.read_csv(file_path)
    data['Date'] = pd.to_datetime(data['Date'])

    TechIndicator = copy.deepcopy(data)

    TechIndicator['Momentum_1D'] = (TechIndicator['Close'] -
        TechIndicator['Close'].shift(1)).fillna(0)
    TechIndicator['RSI_14D'] =
        TechIndicator['Momentum_1D'].rolling(center=False,
            window=14).apply(rsi).fillna(0)
```

---

```

TechIndicator['26_ema'] = TechIndicator['Close'].ewm(span=26,
            min_periods=0, adjust=True, ignore_na=False).mean()
TechIndicator['12_ema'] = TechIndicator['Close'].ewm(span=12,
            min_periods=0, adjust=True, ignore_na=False).mean()
TechIndicator['9_ema'] = TechIndicator['Close'].ewm(span=9,
            min_periods=0, adjust=True, ignore_na=False).mean()
TechIndicator['5_ema'] = TechIndicator['Close'].ewm(span=5,
            min_periods=0, adjust=True, ignore_na=False).mean()
TechIndicator['2_ema'] = TechIndicator['Close'].ewm(span=2,
            min_periods=0, adjust=True, ignore_na=False).mean()

TechIndicator['MACD_2_9'] = (np.tanh((TechIndicator['2_ema'] -
            TechIndicator['9_ema']) * 1000))
TechIndicator['MACD_5_12'] = (np.tanh((TechIndicator['5_ema'] -
            TechIndicator['12_ema']) * 1000))
TechIndicator['MACD_12_26'] = (np.tanh((TechIndicator['12_ema'] -
            TechIndicator['26_ema']) * 1000))

TechIndicator = TechIndicator.fillna(0)

columns2Drop = [
    'Open', 'Low', 'High', # 'Close',
    'Momentum_1D',
    '26_ema', '12_ema', '9_ema', '5_ema', '2_ema', # 'aupband',
    # 'adownband'
]

```

---

```

TechIndicator = TechIndicator.drop(labels=columns2Drop, axis=1)

data_columns = [
    # 'Open', 'High', 'Low', 'Volume',
    'Close',
    'MACD_2_9',
    'MACD_5_12',
    'MACD_12_26',
    'Volumn',
    'RSI_14D'
]

# X_traning, X_validation, y_training, y_validation =
    split_data(TechIndicator, data_columns)
# TechIndicator.insert(0, '-c-', TechIndicator['Close'])
np_arr = TechIndicator[data_columns].to_numpy()
return (np_arr)

# def convert_to_arr(data):

#
# def split_data(data, columns):
#     cols = columns.copy()
#     cols.append('Close')
#     input_features = data[cols]

```



---

```

#   input_data = input_features.values
#
#   prices = data['Close'].values
#
#   scaler = MinMaxScaler(feature_range=(0, 1))
#   input_data = scaler.fit_transform(input_data)
#
#   lookback = 14
#   total_size = len(data)
#   X = []
#   y = []
#   for i in range(0, total_size - lookback): # loop data set with
margin 50 as we use 50 days data for prediction
#       t = []
#       for j in range(0, lookback): # loop for 50 days
#           current_index = i + j
#           t.append(input_data[current_index, :]) # get data margin
from 50 days with margining i
#       X.append(t)
#       y.append(prices[lookback + i])
#
#   X, y = np.array(X), np.array(y)
#
#   X_training, X_validation, y_training, y_validation =
train_test_split(X, y, test_size=0.2, random_state=42)
#

```

---

```
# X_training = X_training.reshape(X_training.shape[0], lookback,
len(cols))
# X_validation = X_validation.reshape(X_validation.shape[0],
lookback, len(cols))
#
# return X_training, X_validation, y_training, y_validation
```

```
def rsi(values):
    up = values[values > 0].mean()
    down = -1 * values[values < 0].mean()
    return 100 * up / (up + down)
```

```
class CompanyDataSampler(Sampler):

    def __init__(self, company_code="AEL", window_episode=None):
        self.n_var = 6
        self.company_code = company_code
        self.sample = self.__load_my_data
        self.title = "CompanyData-" + company_code
        self.window_episode=window_episode

    def __load_my_data(self):
        file_path = get_data_file_path(self.company_code)
        return load_data(file_path), self.title
```

---

---

data\_loader.py - used to extract data from cse website

---

```
from company_list import companies
import requests

data = {
    "symbol": "AEL.N0000",
    "fromDate": "27-07-2020",
    "toDate": "27-07-2021",
    "period": 1,
    "token":
        "eyJhbGciOiJSUzI1NiIsImtpZCI6IjkwNEVEQONGMTVBOUFBNjI3NjIxMzAxNjI2MzVBOTg5NEUzQz"
}

def saveToFile(company, data):
    f = open("D:\\MSC Project\\ml-code\\row-data\\" + company +
            ".json", "w")
    f.write(data)
    f.close()

for company in companies:
    data['company'] = company
    resp = requests.post("https://www.cse.lk/api/charts", data)
```

---

```
saveToFile(company, resp.text)
```

---