UROP 1100 Report - Reinforcement Learning for Financial Trading

Nathaniel Wihardjo (20315011), M. Hanif Dean Nadhif (20314548), Kenneth Lee (20312693)

Department of Information Systems and Operations Management

Supervised by: Prof. Sai Ho (James) Kwok

I. Abstract

Reinforcement Learning (RL) has been gaining popularity for its ability to self-learn a set of actions to maximize gains in different situations. This project implements a financial-model-free RL approach to train an agent to trade against minute-data on the Hong Kong Stock Exchange (HKEX) for portfolio management purposes using Deep Direct RL. Model-free refers to an approach of not learning the model of the environment itself, whereas Deep Direct RL refers to using a variant of an RNN cell as the output layer for the agent's decision and using a deep neural network to learn the features. Two different methods were assessed to expand the features of the data: using technical analysis and closing price momentum. Moreover, the trading agent is trained for trading under one particular stock and multiple stocks while its performance is compared against the backtest. This report will discuss which different types of neural network works best for training RL agent and the different approaches to extract the information out of historical financial market data.

II. Introduction

Numerous researchers have explored various methods to perform better and accumulate more profit in the stock market, including technical analysis, which quantifies market trends into metrics and fundamental analysis, which is based on the intrinsic value of the companies itself. Recently, Deep Learning has emerged as one of the popular alternatives to those methods, however, the latest trend in stock market algorithms is Reinforcement Learning which offers substantial benefits compared to Deep Learning.

In this paper, we explored various Reinforcement Learning methods with the simple goal of maximizing profits. We evaluated a Deep Direct Reinforcement Learning model that is trained with minutes data from the HKEX. As we used a model-free fully machine learning approach, our results should be translatable to various market conditions, not only limited to HKEX. Minutes data is used because we believe that it mimics a real-world trading environment better. With a more frequent input of data, the agent can make trade decisions that are more up to date with real-time conditions. In addition, our model also expanded upon the features by incorporating technical analysis standards, therefore making the model more robust.

III. Research Questions

Some research questions that are addressed and tackled in this report:

1. What type of neural network and algorithm are best for a Reinforcement Learning agent to perform trade based on minute-data?

2. How the features and the data can be expanded to create a robust framework for the RL agent to predict the market?

IV. Literature Review

Table 1. Literature Review Table

o	Authors (Year)	Input Data	Model
	J. Moody, M. Saffell (2001)	Half-hourly FOREX (USD/GBP), Daily S&P 500 Stock Index and Treasury Bills	Recurrent Reinforcement Learning and Q-Learning
	Y. Deng, F. Bao, Y. Kong, Z. Ren, and Q. Dai (2017)	Daily stock-index future and commodity futures (silver and sugar) in Shanghai Exchange, Daily S&P 500 Stock Index	Fuzzy Deep Direct Reinforcement Learning
	Z. Jiang, D. Xu, and J. Liang (2017)	Half-hourly Cryptocurrency market data on eleven sizes	Ensemble Identical Independent Evaluators (EIIE)-based LSTM, RNN, CNN
	N.heess, J.J. Hunt, T.P. Lillierap, and D.Silver (2015)	Morris water maze	Recurrent Deterministic Policy Gradient Neural Network
	X. Gao (2018)	Artificially generated market data	Q learning using GRU, LSTM, CNN and MLP cells
	C.Y. Huang (2018)	Tick-by-tick 12y FOREX data	Deep Recurrent Q- Network

V. Question I: Deep Direct Reinforcement Learning

In order to decide which trading actions to take, it is important that the agent is able to understand a good representation of the data that is fed into the network and take into account the complexity of the environment the agent is trading in. Based on the several research papers

highlighted above, there are different approaches towards learning an optimal policy, or in other words, a set of actions that can maximize the cumulative rewards. Discussion on the two of the most promising approaches by using Deep Direct RL and Deep Recurrent Q-Network is explained below. Given the promising results and relative simplicity compared to the other papers that were shown by the implementation of the Deep Direct RL approach, we have found that this may be most suitable.

a. Deep Direct Reinforcement Learning

Deep Direct RL (DDRL) [2] is a framework that borrows its approach from [1] and can be divided into two different parts. The term 'Deep' refers to the use of a three-layer fully-connected network for feature learning. On the other hand, the 'Direct' part refers to the use of an RNN cell or any variant of it such as a GRU unit to produce the actionable output for the agent. The deep network structure exhibits the ability to learn complex representations of the input data. Meanwhile, the GRU unit enables the network to memorize historical actions and price changes to increase the capability of making better decisions. Using this unit as an output layer is important because of its ability to better capture time-series data. After every action from a set of {-1, 0, 1}, referring to buy, hold, and sell respectively, the agent receives a reward and the total cumulative reward is what we seek to maximize in this problem. However, the action space is not discretized and is instead continuous from (-1,1).

b. Deep Recurrent Q-Network

Due to the partially observable nature of the financial market and lack of baseline available, [6] seek to use a model-based approach instead by using this deep recurrent Q-network algorithm by framing this trading problem as a Markov Decision Process (MDP). The same action space is adopted as the first approach but is now discretized as the agent's current position is one-hot encoded using a 3-dimensional vector (i.e. if the position is buy, the vector is [1, 0, 0]). However, this approach uses portfolio log returns as the reward function, defined by the following:

$$r_t = \log\left(\frac{v_t}{v_{t-1}}\right)$$

Vt denotes the portfolio value at that particular time step t. In this way, the total cumulative reward function can be interpreted as future discounted log returns. For the network architecture itself, it uses a four-layered network to approximate the policy that it would like to learn with a linear layer as the first two, an LSTM layer in the third, and a linear layer as the output. Moreover, this approach uses a greedy action selection but given the poor results that are shown in [6] when trading FOREX data, we decided not to use this approach. Moreover, an online approach is not necessary and given that there are available codebases to adapt from for the first approach, the Deep Direct RL method is used for the purposes of this research.

VI. Question II: Knowledge Extraction from the Data

Analyzing the data and extracting the information out of the fed data is a crucial part in this project as the model will learn based on the data and as the trading agent will act mostly upon the data. The following are two of the most common and robust approaches to explore the knowledge from the data.

a. Pattern Matching

Y. Deng [1] on his RL-based financial market trading research uses integrated Supervised Learning (SL) to predict the market movement and trends. This is similar to the "Pattern Matching" categories [2] which essentially having an entire input of all assets' history prices and a neural network which is able to produce a predicted matrix of asset prices for the next period. The approach is relatively simple as it calculates the different types of closing price momentum and expands the data into 50 different features as the input [1]. Then, fuzzy clustering [3] is used to categorize the data into three types of market: bull, bear, and stagnant, for which the agent takes the information based on the information.

Although pattern matching is relatively less complicated and more straightforward to implement, the performance of the trading agent is highly coupled to the prediction accuracy, and the future market price is difficult to predict solely based on historical data and trends occurring over 50 different time points. An additional layer of logic should be added for the agent to learn the risk over the transaction cost and thus, is not fully extensible as Z. Jiang argues [2].

b. Technical Analysis

Compared to momentum changes, technical analysis provides a more robust framework as it calculates and produces numbers of different signals based on different indicators [4] not only on the closing price changes. The trading agent is fed with these different signals as inputs into Deep Neural Network (DNN) for the agent to learn on the extracted information. It is believed that all relevant information from the data is presented through different variables of technical analysis which reflects the sum total knowledge of all market participants [5]. As this

approach does not cluster the price change on three simple trends, it provides more analysis on the data which is more useful when the agent is made to manage different assets and portfolios [2].

VII. Approach and Implementation

a. Data Pre-processing

All of the data pre-processing is through the folder "data_wrangling/". Detailed documentation is provided in the folder, named "README.md".

Data wrangling and cleaning is required on top of the provided minute-data as the data contains some noises. A shell script is used to aggregate all of the data into one file for a particular stock because of the nature of the raw data that is aggregated based on time points. Bash scripting is used since it performs significantly faster in terms of file handling compared to using a Python script, particularly handling numerous files separated in each month. However, one drawback of the shell script usage is that it can only be performed in Linux OS.

The shell script is also used to clean and make the format of the CSV files consistent. Some noise observed during pre-processing are the following: empty data points having no prices at the year of 2004, different format of the date and time, and inconsistent comma usage to separate the date and time on some files. These inconsistencies provide further complication in the data processing, for which the data is cleaned and its format made consistent on aggregation through bash script.

b. Reinforcement Learning Model

The project is adapted from several other research projects' codebase [6], [7] which are implementations based on the Deep Direct RL [1] and technical analysis features expansion [2].

i. Environment

The environment is where the trading agent is trained, and performed trades on. It manages the flow of the information from the input data, defines states of the trading agent, and holds the information of the reward, amount and values of assets in possession. A state is defined as a single time-point where the trading agent is located; thus, the state of our RL model is one-direction time-series. Since the complexity of a state makes the agent impossible to get all the information of a state, the environment passes on relevant data to the agent for each state. Before the trading agent is trained, input financial market data is re-processed and expanded to 26 features, each of which represents a signal from the technical analysis [6] utilizing python-implemented TA-Lib[1] library.

```
security['MOM'] = talib.MOM(close_price)
security['HT_DCPERIOD'] = talib.HT_DCPERIOD(close_price)
security['HT_DCPHASE'] = talib.HT_DCPHASE(close_price)
security['SINE'], security['LEADSINE'] = talib.HT_SINE(close_price)
security['INPHASE'], security['QUADRATURE'] = talib.HT_PHASOR(close_price)
security['ADXR'] = talib.ADXR(high_price, low_price, close_price)
security['APO'] = talib.APO(close_price)
security['AROON_UP'], _ = talib.AROON(high_price, low_price)
security['CCI'] = talib.CCI(high_price, low_price, close_price)
security['PLUS_DI'] = talib.PLUS_DI(high_price, low_price, close_price)
security['PPO'] = talib.PPO(close_price)
security['MACD'], security['MACD_SIG'], security['MACD_HIST'] = talib.MACD(close_price)
security['CMO'] = talib.CMO(close_price)
security['ROCP'] = talib.ROCP(close_price)
security['FASTK'], security['FASTD'] = talib.STOCHF(high_price, low_price, close_price)
security['TRIX'] = talib.TRIX(close_price)
security['ULTOSC'] = talib.ULTOSC(high_price, low_price, close_price)
security['WILLR'] = talib.WILLR(high_price, low_price, close_price)
security['NATR'] = talib.NATR(high_price, low_price, close_price)
security['RSI'] = talib.RSI(close_price)
security['EMA'] = talib.EMA(close_price)
security['SAREXT'] = talib.SAREXT(high_price, low_price)
security['RR'] = security[close_name] / security[close_name].shift(1).fillna(1)
security['LOG_RR'] = np.log(security['RR'])
if volume_name:
    security['MFI'] = talib.MFI(high_price, low_price, close_price, volume)
```

Figure 1: Technical Analysis Features Expansion

Each time the trading agent explores and executes a trade (in a particular state), the environment receives a matrix containing the desired position of each asset from the trading agent. Then, price and reward are calculated and normalized and passed to the trading agent on the next state. The reward is calculated by multiplying normalized portfolio value (amount of

[1] https://mrjbq7.github.io/ta-lib/

assets in possession multiplied by the closing price) with the log return rate (price at time t+1 divided by price at time t) [2]. The agent is trained to maximize this reward.

$$r_t = nPV_t * log(\frac{pr_{t+1}}{pr_t})$$

ii. Trading Agent

All details about the trading agent can be found in the folder '/model/' and in the files drl_agent.py. This code borrows its implementation from the Github RLQuant and is written using a machine learning framework PyTorch. The agent learns actions from the following description of the network. As aforementioned above, there are three layers of fully connected networks of output sizes 128, 64, and 1 respectively from an initial input size of 128. Each of these layers uses a ReLU activation function. It also uses a GRU cell in order to better learn the representations of time-series data. At the output, a *tanh* activation function is used because we want to have actions that are represented within the continuous range of (-1,1) to indicate the position that the agent should take on when trading. The input data given by the trading environment will be passed through this network when making the forward pass and the weights of the network will be adjusted in its backpropagation step.

c. Training and Result

All detailed results, comparison, indicators, different figures, their explanations, and documentation are provided in the Jupyter notebooks for each dataset which can be found under the name "DRL_AIA+HSE.ipynb", "DRL_BOC.ipynb", and "DRL_BOC+CNOOC.ipynb".

TQDM[2] is used to provide a simple interface to manage the epochs while training the RL agent. We trained the agent for 10 epochs and configured the batch training size to be 64. To shorten training time when any anomalies occurred, a special condition is added to stop the training epoch when the sum of the portfolio value at the end of the iteration is larger than a threshold. Following the previous adaptation [6], the agent was given initial capital to be 10^5 (unit follows the unit of the input data, in this case, HKD).

The agent was trained for two different categories, a single stock, and portfolio management. For single stock, we used Bank of China (BOC). For portfolio management, the agent was trained using two different datasets: AIA and Hangseng stock (HSE), BOC and China National Offshore Oil Corporation (CNOOC). The data used is ranged from 2015 to 2017. Stocks were chosen based on its high volatility, price movement, and trading volume. The high price movement is a desired property of the stocks as the agent would be able to exploit the price movement by selling or buying stocks more to gain more reward.

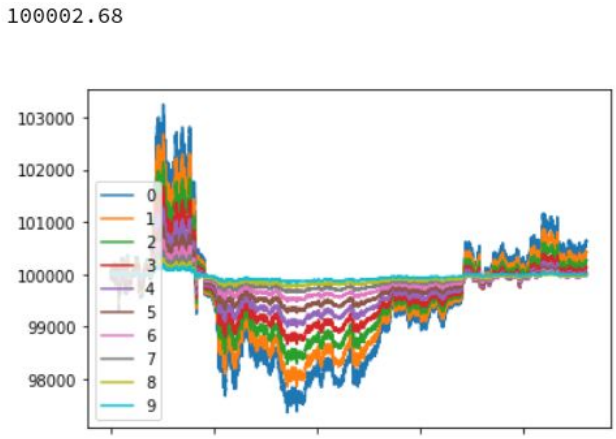

Figure 2: Portfolio Value of RL Agent on Single Stock (BOC)

[2] https://tqdm.github.io/

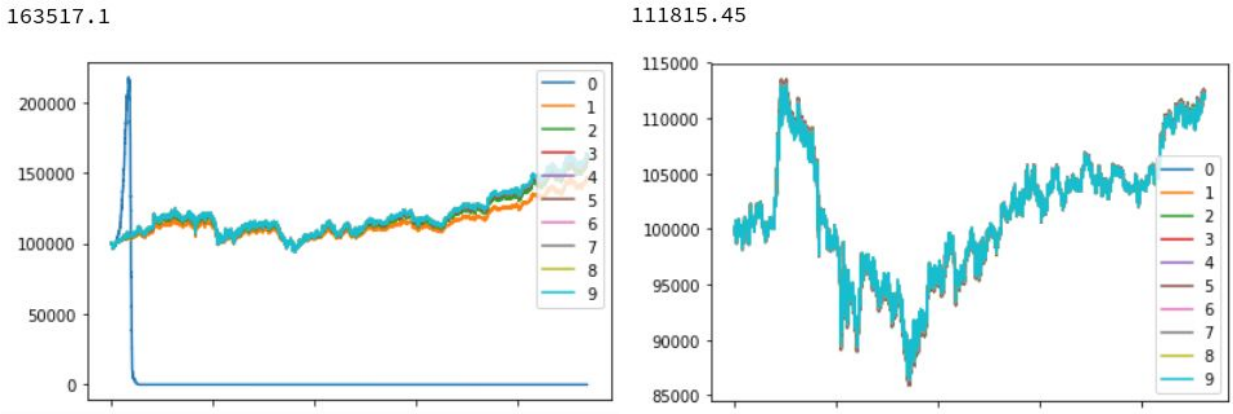

Figure 3: Portfolio Value of RL Agent on Multiple Stocks (left: AIA and HSE. right: BOC and CNOOC)

As observed from both Figure 2 and Figure 3 above, in all cases the RL agent is able to create some profit by the final portfolio value specified above each figure (the capital base is smaller than the final portfolio value). Each figure shows us the portfolio value over one full iteration in a time-series manner for 10 iterations. It can be observed that training the agent more than 10 epochs would not be necessary as the portfolio value over the last 2 or 3 iterations are about the same.

The trading agent was back-tested against an initialized value of weights which executes trade solely based on the price and not optimized to maximize the reward gained. The final comparison is made between the RL agent and backtests on the cumulative returns. Pyfolio[3] library which provides easy access to evaluate performance and risk analysis to do the comparison. In addition, different tearsheets were made to further compare the performance of the trading agent and the backtest based on different indicators. Based on Figure 5, the RL agent outperforms the backtest on both datasets and proves the RL agent's ability to maximize the rewards.

[3] https://quantopian.github.io/pyfolio/

```
In [66]: pf.create_simple_tear_sheet(returns=pr,benchmark_rets=bpr)
```

Start date	2015-01-02
End date	2017-12-29
Total months	11136
	Backtest
Annual return	0.1%
Cumulative returns	63.5%
Annual volatility	1.4%
Sharpe ratio	0.05
Calmar ratio	0.00
Stability	0.59
Max drawdown	-24.8%
Omega ratio	1.01
Sortino ratio	0.07
Skew	NaN
Kurtosis	NaN
Tail ratio	1.00
Daily value at risk	-0.2%
Alpha	0.00
Beta	0.99

Figure 4: Simple Tear Sheet from Pyfolio for AIA and HSE Dataset

Figure 5: RL Agent and Backtest Performance Comparison (left: AIA and HSE. right: BOC and CNOOC)

The next step of the experiment would be to further adjust and finetune the DNN architecture to better suit minute-data as the predecessor project that we have based our implementation upon is designed for daily-data. Different types of neural networks and different approaches to obtaining useful knowledge from the data for the agent to understand are some of the areas this project would like to be expanded. Given that this project assumes no commission fees for simplicity, the consideration of different values of commision fee as part of the risk would also be appropriate to be tested in future works as this would replicate real-life trading

more accurately. In summation, this project provides an understanding on building neural network architecture for training RL agent to trade on the financial market utilizing the knowledge from technical analysis based on historical data.

VIII. Conclusion

In conclusion, there are a variety of approaches that could be taken to build an RL agent that is able to trade on the financial market using a set of historical data consisting of prices and trade volume. Given that this field is particularly still at its early stages of development, we found that currently, the approach taken by using a Deep Direct RL is one that is most promising. However, in the future, we think that there are still some ways to better fine-tune the model and produce better performance for any kind of financial instrument. Moreover, the extension of trading with multiple stocks on a portfolio can be further improved given the added complexity of the environment. As of now, current research is also being done to simulate trading in a multi-agent environment as well.

IX. References

[1] Y. Deng, F. Bao, Y. Kong, Z. Ren, and Q. Dai, "Deep Direct Reinforcement Learning for Financial Signal Representation and Trading," in IEEE Transactions on Neural Networks and Learning Systems, vol.28, no. 3, pp. 653-664, March 2017.

[2] Z. Jiang, D. Xu, and J. Liang (2017). "A Deep Reinforcement Learning Framework for the FInancial Portfolio Management Problem" [online]. Available at: https://arxiv.org/abs/1706.10059

[3] C. -. Lin and C. S. G. Lee, "Neural-network-based fuzzy logic control and decision system," in IEEE Transactions on Computers, vol. 40, no. 12, pp. 1320-1336, Dec. 1991. doi: 10.1109/12.106218.

[4] A.W.Lo, H. Mamaysky, and J. Wang. “Foundations of Technical Analysis: Computational Algorithms, Statistical Inference, and Empirical Implementation”. The Journal of Finance, 55(4): 1705-1770, 2000.

[5] D. Charles, I. Kirkpatrick, and J.R. Dahlquist. “Technical Analysis: The Complete Resource for Financial Market Technician”. ISBN-12, pages 978-0137059447, 2006.

[6] https://github.com/yuriak/RLQuant

[7] https://github.com/ZhengyaoJiang/PGPortfolio