Reinforcement Learning on Stock Trading: Smooth Approach

st Andrew Huang
Master in Data Science, Indiana University, Bloomington
andhuan@iu.edu

***Abstract*—In this report, I aim to set a framework of reinforcement learning algorithm on stock trading. Compare to current models, I emphasize on two aspect: (1) the time-series property in stock (2) the delay effect in real trading. To explore the model, I applied Q-Learning on single ETF. While the algorithm performances are not ideal, it avoids the 2020 crash which was generally considered caused by the eruption of COVID-19.**

***Index Terms*—Reinforcement Learning, Q-Learning, stock, time-series**

I. Introduction

Implementing RL model on stock trading is not a blend new idea. There are enough incentives for researchers and hedge funds to pay attention on the approaches. If reinforcement learning algorithms can beat world chess champion, why not apply the models to beat other human investors and traders? Unfortunately (or fortunately), stock market and chess games are different essentially. In a word, it is possible there is no pattern in stock market, which fail the performance of basic algorithms. It is also hard to teach an algorithm the "common sense" in stock market; for instance, how do you tell machines the magic attractive in AAPL (Apple Inc.) but not in AAP (Advance Auto Parts, Inc.)? In the following paragraphs, I will show literature review and describe what I set to do. Then I will describe what I actually did, includes the framework setting, the trading rules, the evaluate parts. Finally is the outcome and experiments.

II. Literature Review

In Yang et al.(2020) [1], the authors used OpenAI to build the trading environment and focus on optimizing stock trading strategies through ensemble reinforcement learning methods. They show their algorithm performs better than A2C and DDPG. In Ma et al.(2021) [5], the authors propose a novel model named Parallel Multi-Module Deep Reinforcement Learning (PMMRL) algorithm, and they claim the proposed model can extract features from the whole environment by the above two modules simultaneously, taking the advantages of both LSTM and FC layers.

III. Project Plan

A. Set to Do

In my original proposal, I planned to use OpenAI to build the environment. Compare to Yang(2020) [1], I also planned to use more features in chip analysis, such as trading data of institutional investors, margin trading. But I found it is uneasy to obtain direct free API for them (there are some sources provide the data I need but they are not free, ex: Quandl). Therefore, I decided to skip this part.

Instead of onbtain maximum final return, the goal of the model is to generate "smooth" return. Also, I will do do experiments that training data source of individual stock data and get ensemble prediction to decide best action for ETF trading. And I would like to try more recent algorithms for the model.

B. Actual Progress

I change the environment from OpenAI to tensorflow [3]. Although the authors in Yang(2020) have provided their github works, it turns out they have different goal with me and it takes me a lot of time revising the environment. Quantitative investment performances are the results of prediction, which I want to focus on, and strategies, which Yang(2020) focused on. Also, due to the limitation of time, I am unable to finish the ensemble model.
Finally, my work is building the trading environment, developing reinforcement model (Q-Learning) and test on ETFs.

IV. Model Framework

A. Data

Data Source is `yfinance`, the API of *Yahoo! Finance*. I follow the period used in [1], that is, the range of train data is from 2009 to 2014, validation data is 2015 (in figure it shows validation period is 10/2015 t0 12/2015, however in my algorithm the periof is 01/2015 to 12/2015), and test data is from 2016 to May 2020.

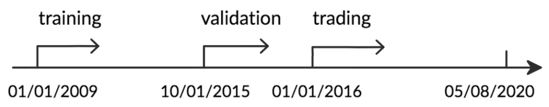

Fig. 1. Train, Validation, Test time (Copyright by AI4Finance-Foundation)

B. Features

The features I used are all technical analysis features which can be computed by public data. The feature engineering process are in *collect data.py*. MACD, RSI, ADX are features used in [1].

- **MACD** Moving Average Convergence Divergence (MACD) is calculated using close price. The only parameter is moving period, for example, if moving period is p, then

$$MACD(p) = (\sum_{i=0}^{p-1} Close(t-i))/p \tag{1}$$

time-series features are widely used in financial-related research, such as GARCH model [2]. Since MACD considers the close price from *t-p+1* to current data, it contains the information of *AR(p-1)* close prices. In the model I use *p=1,2,3,...,10,15,20,25,30,40,60* days.

- **RSI** Relative Strength Index (RSI) is calculated using close price. IT consists of RS, average gain and average loss. The formula is

$$\begin{aligned} RSI &= 100 - \frac{100}{1+RS} \\ RS &= \frac{\sum_t |gain_t|/n}{\sum_t |loss_t|/n} \end{aligned} \tag{2}$$

If *n=14*, average gain = [(previous average gain) x 13 + current Gain] / 14. In model I use *n=3,5,10,15,20,25,30*

- **ADX** Average Directional Index (ADX) is calculated using high, low and close price. For some traders, the indicator can provide the information of the strength of a trend. The details of the equations are in [4]. In model I use period days *=7,9,11,13,15,...,19,21.*

In the algorithm, the features of $MACD$ and ADX will be computed as the ratio between two feature. Therefore, there are 30 features at one time ($MACD$(16) + RSI(7) + ADX(7).

C. Evaluate Method

It is a crucial problem to decide how to evaluate the model. The reason is the investment performance is the combination of model prediction and trading strategy. If a model has prediction accuracy as good as random guess, but the trading strategy is carefully designed, the final performance may not be too bad. I follow two principles for evaluation: First, the trading strategy should be simple, so we can reduce the effect of strategy. Second, the baseline should be another simple on the same target.

If we only have one target, say SPY (SPDR S&P 500 ETF Trust), how do we calculate the return rate? The easiest and straight-forward method is given an amount of initial money, so the algorithm cannot buy stock with unlimited volume. A question is the price of SPY can change a lot in training and test periods. At beginning of 2014, SPY price is about 183; while in January 2022, it goes to 466. If the initial money is too low, the algorithm may be unable to do any action because it cannot even afford one amount of stock. I use quadruple of the first close price in training data as the initial money. And I have to admit this may be inappropriate for some surging stock. In [1] authors use 30 stocks to build a portfolio, so somehow they avoid the problem. In my experiment, we can regard it is a portfolio with only two components: stock and cash. Consider the low saving interest rate, I set interest rate=0 to simplify the question.

1) Trading Rules:

- The actual trading date is one business day after the signal date. For example, if the algorithm gives action "buy" on 29 April, then the actual trading date and price is on 2 May. The algorithm makes decision base on the "close price" on 29 April– which indicates it is end of trading time. Therefore, we are unable to buy ticket on 29 May. Instead, we will buy the stock with the close price of 2 May.
- Loan is not allowed. As mentioned before, the initial money is four times teh first close price in prediction period. The algorithm can only trade one amount of stock at one time. If the algorithm buys stock four times without selling them, it is very likely it already uses all of its money, and it is not allowed to buy the fifth stock even if the algorithm gives out the action of "buy".

D. Model

1) Q-Learning: The Q-Learning update equation is as below:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha[R + \gamma * maxQ_a(s_{t+1}, a)Q(s_t, a)] \tag{3}$$

Also, *epsilon-greedy* function is applied. The initial value of $epsilon = 0.5$, and it will decay after each iteration.

2) Reward: How do we calculate reward? In my opinion, the reward should indicate the relative return of the algorithm and the baseline portfolio. The baseline is defined as below.

- **Baseline Portfolio** Given initial money, the baseline investor will buy one stock at the second day, and hold till the end of (evaluate) period. In any day T during the period, the value of baseline portfolio is Initial Money - Close Price (*t=1*) + Close Price (*t=T*).

To give algorithm the information of baseline portfolio, I set current reward *R* as

$$R_t = ((M_t + SV_t)/M_0)/PV_t - 1 \tag{4}$$

where M is cash in hand, SV is stock value hold (if no stock in hand then the value is 0), M_0 is initial money, and PV is the baseline portfolio value. Therefore, if R is negative, it means our algorithm perform worse than baseline. If the target stock is ETF (Exchange Traded Fund), it implies our algorithm does not "beat the market". So if our investment loss 10% money while the market crash for 30%, it is not too bad;m on the other hand, if the market is in bull market and we only keep cash, the algorithm will be punished by this equation.

E. Module Structure

The initial structure of the model follows the framework of Shah(2021) [3]. Most of the process in the raw module does not meet the principles I set. For example, the train period and test period are the same; and the trading dates are set as exactly the day of signal. $Tensorflow$ is used to build the environment. The environment building is in $RL_QL_test_v3.py$ In my original proposal, I plan to implement the environment from [1], which is built by gym. However, due to the different evaluate approach, the structure of [1] is not quite straightforward to my model. Therefore I have to abandon the idea for implementing their environmental module.

V. Experiments

I use SPY as target stock. $\epsilon = 0.5$, which will decay 0.999 in each iteration until it meets 0.01. The state size is 30, which is the number of the features at one time. Three actions are used: hold(0), buy(1) and sell(2).
The figure shows the performance on SPY. The train period is from 2009 to 2014, validation data is 2015, and test data is from 2016 to May 2020. Black line is the performance of algorithm, and the blue line is the performance of baseline portfolio. We can notice that, the algorithm avoid the crash at the end of 2018, and sells stock immediately during the crash in 2020.

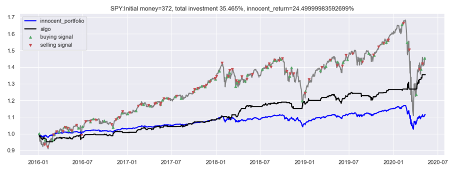

Fig. 2. Evaluation on SPY

If we see the performance in validation during training, we can find there's no amazing performance.Validation is on the period of 2015.

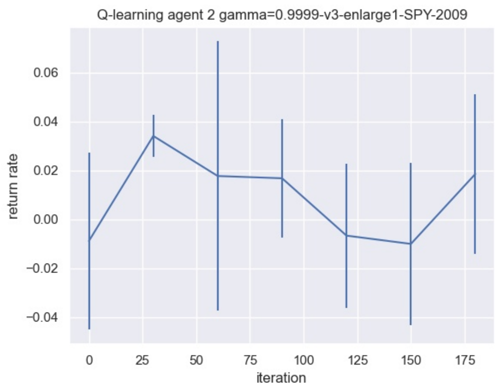

Fig. 3. Train Validation Evaluation on SPY

A. Punishment and Over-fitting

It is naturally that we would want the model can fit the data more. In order to emphasize the loss-averse or the reward, I tried two equations:

1 $R = R * \beta$ if $R < 0$

2 $R = R * \beta$

where $beta > 1$. In the first equation, the agent will be punished more if it is lose to baseline performance. However, the performance is not stable and it actually have worse performance. The figures shows when $\beta = 2$, one of the test performance and validation performance.

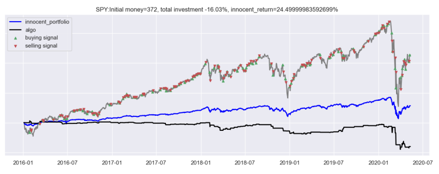

Fig. 4. Evaluation on SPY, $\beta = 2$

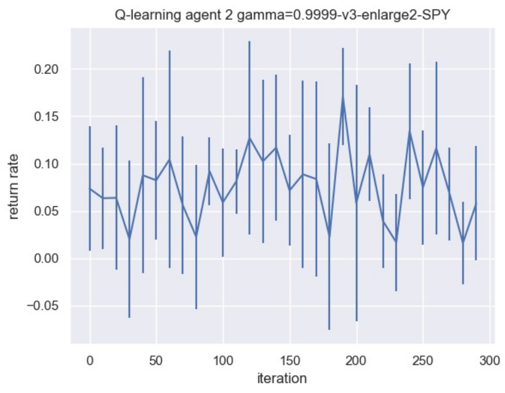

Fig. 5. Train Validation Evaluation on SPY, $\beta = 2$

VI. Conclusion

A. Results

In this report, I try to develop a reinforcement learning framework that can have smoother investment return and outperform baseline portfolio. In the case of SPY, the algorithm avoid main crashes and have better return rate than baseline. Notice that the algorithm use data only earlier than 2015, but the prediction period is from 2016 to 2020, which is amazing because it indicates that the model is still have some effect after four years. Compare to the performance in Yang et al.(2020), my model is not too bad on avoiding stuck in market crash.

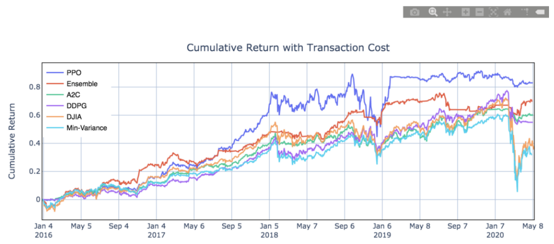

Fig. 6. Performance of Yang et al.(2020)(Copyright by AI4Finance-Foundation)

However, there are so many aspects in the algorithms can be improved. And the results of the experiments did not meet my expectation. In the following paragraph I am going to discuss the failures and possible improvements.

B. Discussion

1) Continuously Outperform: Stock trading is a time-series strategy. It can be a problem if the algorithm does not perform "equally", that is, the model has high accuracy in time t to $t+k$, but poor accuracy from $t+k+1$ to $t+k+d$. If k, d is large enough, the algorithm would be considered unstable. For example, in the experiment of enlarge punishment, I try different time period for train and test data. The figure shows when test period is 2018 to 2022 (train period is 2011 to 2016, validation year is 2017). Although it looks OK after 2020, it has some terrible performance at the end of 2018. And in real world, unfortunately, the loss in 2018 may cause the model been abandoned by investors (or, the investors would give up being an investors).

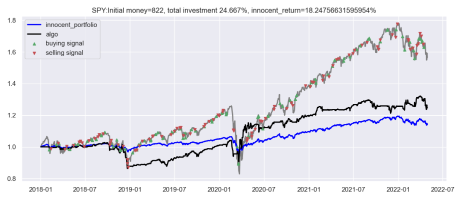

Fig. 7. SPY, $beta = 2$, test period:$2018 - 2022$

2) Over-fitting: I think this is the most changeable and interesting part in financial prediction. Stock market reflect the ensemble behaviors of investors, and may be affected by any information source. For example, if there is no epidemic disaster occur during the training period, how can the algorithm "know" how to react when it meets 2020 crash?
One method to solve this is to shorten the test period. Given the assumption that the algorithm are more accurate to the test periods that are closer to its train periods, we can use train data $t : t+k$ to predict $t+k+1$, and use $t : t+k+1$ to predict $t+k+2$ period.

C. Future Work

I plan to build ensemble approach for the algorithm. Since ETF consists of a bundle of tickers (stock companies), it may be reasonable to do reinforcement learning toward individual stock, and use the prediction for the stock as input to do reinforcement learning on ETF prices. It is a pity that due to time limitation, I cannot complete and perform the work in this report.
I am grateful having the chance that force me to build the model which I plan to do but never take actions.

References

[1] Hongyang Yang, Xiao-Yang Liu, Shan Zhong, and Anwar Walid. 2020. Deep Reinforcement Learning for Automated Stock Trading: An Ensemble Strategy. In ICAIF '20: ACM International Conference on AI in Finance, Oct. 15–16, 2020, Manhattan, NY. ACM, New York, NY, USA.

[2] Franses, P.H. and Van Dijk, D. (1996), Forecasting stock market volatility using (non-linear) Garch models. J. Forecast., 15: 229-235. https://doi.org/10.1002/(SICI)1099-131X(199604)15:3¡229::AID-FOR620¿3.0.CO;2-3

[3] Ekta Shah (2021), Bear run or bull run, Can Reinforcement Learning help in Automated trading, Data Science Blogathon, https://reurl.cc/9G5LYa

[4] https://www.investopedia.com/terms/a/adx.asp

[5] Cong Ma, Jiangshe Zhang, Junmin Liu, Lizhen Ji, Fei Gao,A parallel multi-module deep reinforcement learning algorithm for stock trading,Neurocomputing, Volume 449,2021, Pages 290-302, ISSN 0925-2312, https://doi.org/10.1016/j.neucom.2021.04.005.