Deep Reinforcement Learning in Portfolio Management

Mincai Lai*
laimc@shanghaitech.edu.cn

Jie Wang*
wangjie1@shanghaitech.edu.cn

Haoyu Tu*
tuhy@shanghaitech.edu.cn

ABSTRACT

Portfolio management combines the amount of monetary fund as an investment product and allocates the budgets dynamically based on their potential benefits. This study applies the deep reinforcement learning (DRL) methods DQN and DDPG to train the trading strategy in which an agent is trained to identify the optimal trading action. Our DQN model defines the actions as real trading actions (buy, sell, hold), and the DDPG model defines the actions as the distribution weights of assets. The models are implemented and applied to SSE(Shanghai Stock Exchange) dataset. To validate the ability of DRL methods in portfolio management, we conduct historical market backtests and generalization performance testings, which demonstrate the superiority of our deep reinforcement learning strategies over the baseline strategies.

KEYWORDS

algorithmic trading, portfolio management, reinforcement learning, deep learning

1 INTRODUCTION

Portfolio management is the process of combining an amount of fund into financial investment products based on their potential benefits. A successful portfolio management means low risk and high return. The process usually performed in two steps [5]. Firstly, find the best portfolio allocation. After computing the expected returns of the stocks and the covariance matrix of the stock prices, the best portfolio allocation can be found by either maximizing the return for a fixed risk or minimizing the risk for a fixed return. Secondly, conduct the best trading strategy following the best portfolio allocation. There are four categories of traditional portfolio management methods, "Following-the-winner", " Following-the-loser", "Pattern-Matching", and "Meta-Learning" [4]. Prior-constructed financial models are required in the first two categories, while some machine learning techniques may also be used since they contain the parameter determination part. “Pattern-Matching" methods optimize the portfolio based on the market distribution predicted by historical data and the sampled distribution. "Meta-Learning" methods are a combination of multiple strategies of other categories that can achieve stable performance.

There are existing attempts of applying deep learning technologies to financial market trading [3, 8]. The core idea of many of them is predicting the price movements based on historical prices. With the historical price of the fund, a supervised deep learning network can be trained. It is a regression problem straightforward to implement. However, the price of the fund is hard to predict since the market is complex and rapidly changing. Besides, the prediction of the price is not enough for portfolio management. Which action should be taken can not be directly decided by future price, there require additional information and logic. Therefore, deep learning is suitable for price prediction, while subsequent steps are required when applying into portfolio management.

Reinforcement learning (RL) methods are also used in trading problems. These methods try to output the trading signals instead of predicting future prices. They are on single-asset, while general portfolio management is on multiple assets.

Deep Reinforcement Learning (DRL) is a combination of deep learning and reinforcement learning. Recently, Deng et al. [2] have applied deep RL to single-asset trading and verified its performance in the fund market. Zhang et al. [1] achieved a high return by adopting the Deep Deterministic Policy Gradient (DDPG) algorithm with a strategy of investing the vast majority of money into the best fund.

There are few applications of deep RL methods in portfolio management in the Chinese fund trading market. This project will use deep RL methods like the DDPG algorithm to solve the portfolio management problem. We use 19 stocks as our target stocks from the SSE(Shanghai Stock Exchange) dataset. The dataset includes the history price of the stocks from 2014-01-02 to 2019-08-22. The price on each day contains open, high, low and close. We use 2014 to 2017 as training data and 2018 to 2019 as testing data.

We have implemented two deep reinforcement learning models DQN and DDPG in this project. Experiments are conducted to test their performance in portfolio management on China stock market. The following hypotheses part specifies the fundamental assumptions. The methods part introduces the pre-assets selection and the models we used. The experiments part shows the results and analysis of our implemented models. The conclusion part lists our contributions and the challenges we can explore in the future work.

2 PROBLEM STATEMENT

Portfolio management is the action of continuous reallocation of capital into several financial assets. For an automatic trading robot, these investment decisions and actions are made periodically.

2.1 Trading Period

There are four important price points characterize the overall movement of a period, namely the opening, highest, lowest and closing prices. In our project, trading algorithms are time-driven, where time is divided into periods of equal lengths T. At the beginning of each period, the trading agent reallocates the fund among the assets. We can assume the period as t. where $w_{i,t}$ represents the fraction of investment on stock i at timestamp t, so we can get that

2.2 Mathematical Expression

As we know, the portfolio consists of m assets. The closing prices of all assets comprise the price vector for Period t, v_t. In addition to m stock information, the cash itself information is also very important. In continuous markets, elements of v_t are the opening prices for

*Both authors contributed equally to this research.

Period $t+1$ as well as the closing prices for Period t. So we can get

$$\boldsymbol{y}_t := \boldsymbol{v}_t \oslash \boldsymbol{v}_{t-1} = \left(1, \frac{v_{1,t}}{v_{1,t-1}}, \frac{v_{2,t}}{v_{2,t-1}}, \ldots, \frac{v_{m,t}}{v_{m,t-1}}\right)^\top \tag{1}$$

Note $y_t[0]$ represents the relative price of cash.Since the prices of all assets are quoted in cash, the first elements of $v_{0,t}^{(\mathrm{hi})} = v_{0,t}^{(\mathrm{lo})} = v_{0,t} = 1$, $y_t(1) = 1$. We define portfolio weight vector as

$$w_t = \left[w_{0,t}, w_{1,t}, \cdots, w_{N,t}\right] \tag{2}$$

where $w_{i,t}$ represents the fraction of investment on stock i at timestamp t, so we can get that

$$\sum_{i=0}^{N} w_{i,t} = 1 \tag{3}$$

We can get the profit after timestamp T as

$$p_T = \prod_{t=1}^{T} y_t \cdot w_{t-1} \tag{4}$$

When considering w_0, the initial portfolio weight vector w_0 is chosen to be the first basis vector in the Euclidean space:

$$\boldsymbol{w}_0 = (1, 0, \ldots, 0)^\top \tag{5}$$

2.3 Transaction Cost

Buying or selling assets for free is impossible in the real market. So we consider transaction cost. In order to make the simulation closer to the reality, we're going to set a trading cost factor u_t. Then the computation of profit changed:

$$p_T = \prod_{t=1}^{T} (1 - \mu_t)\, y_t \cdot w_{t-1} \tag{6}$$

But we will set this cost as zero firstly to simplify things.

2.4 Hypotheses

In establishing these formulas, we need to specify the following assumptions:

- The amount of trading cash is small enough that it does not affect the behavior of other investors in the market or the price of financial assets.
- At timestamp t, we buy stocks according to the portfolio weight vector w_t computed by history data at open price and sell all the stocks at the close price. This may not be true in practice because you will not always be able to buy/sell the stock at open/close price.
- Investors can purchase any number of shares according to the optimal capital allocation weight, including the purchase of shares can be non-integer.

3 METHODS

3.1 Data Preprocess

3.1.1 Data Source. We used the SSE(Shanghai Stock Exchange) dataset provided by Shanghai Stock Exchange Information Network Co., Ltd. The dataset contains the historical data of all stocks listed on the Shanghai Stock Exchange from 2014-01-02 to 2019-08-22. The history data contains open, close, high, low, volume and amount of every stock on each day.

3.1.2 Stock Selection. The dataset for our project is formed by the historical price data of 19 stocks chosen from SSE dataset. The stocks are chosen from 5 industries: media entertainment, home appliance industry, automobile manufacturing, biopharmaceuticals, electronic information. We chose stocks from different industries to make our model more applicable and to reduce the risk from the industry. Only five industries are chosen because we want to keep more than three stocks for each industry so that we can analyse whether our model correlates with a particular industry. Followed are various operations we took in this phase:

- Collect industry information. Collect the stock code list of all stocks from the five industries. This step is conducted by a Python package named 'tushare' which provided an API for stock industry information. The stock code list contains stock code, stock name and industry name for each stock.
- Find stocks listed in SSE: Filter the stock code list and kept the stocks whose stock code starts with '600'.
- Select stocks by industry. This step contains two sub-steps. Firstly, group the stocks by industry and pick four stocks from each group. Then, pick 19 stocks from the 20 stocks. Each step stocks are selected randomly.

3.1.3 Feature Selection. After stock selection, we got the stock code of 19 stocks. Then, we collect the close price of each stock from SSE dataset by stock code. Occurring to the proposal of our model, we only need the daily close price of every stock. Thus, the close price is the only feature we need. At this step, we process the original dataset and form our origin dataset. Our origin dataset is a table with 19 stocks' code as columns name and date as row index. The value of each cell is the close price of the stock on that day.

3.1.4 Filling missing data. Notice that the close price of a stock in origin dataset may be null value because of suspension in reality. As our proposal said, the price during the suspension period is set as the close of the last trading day before the suspension.

3.1.5 Split dataset. The dataset is spat by date. Training dataset contains data from 2014-01-02 to 2017-12-31. The testing dataset contains data from 2018-01-02 to 2019-08-22.

3.2 DRL Framework

We discuss the stock trading process in DRL framework like [1].

- `State(s)`: We define state s_t as the observation of timestamp t.

$$\boldsymbol{s}_t = (\boldsymbol{V}_t, \boldsymbol{w}_{t-1}) \tag{7}$$

  where V_t is the price tensor, w_{t-1} is the portfolio vector from the last timestamp.
- `Action(a)`: In our framework, action is the portfolio weight vector.

$$a_t = w_t \tag{8}$$

- `Reward(r)`: Reward is the change of the portfolio value when an action is taken at one state and arriving at a new state. The initial value of reward is 0 and the final value we defined as p_t. Here we use the Logarithm method to convert the quadrature form of the total portfolio into a more easily

solved summation form.

$$r_t = \log p_T = \log \prod_{t=1}^{T} \mu_t y_t \cdot w_{t-1} = \sum_{t=1}^{T} \log(\mu_t y_t \cdot w_{t-1}) \quad (9)$$

Based on the DRL solution framework, We explored two powerful DRL algorithms, namely Deep Q-network (DQN)[9] and Deep Deterministic Policy Gradient (DDPG)[6], find the best strategy in the complex stock market of China.

3.3 DQN

Q-learning is an algorithm that maximizes the reward by getting Q-table, Q-table is a mapping of environment states to agent actions. Here is the equation for Q, from Wikipedia:

$$Q^{\text{new}}(s_t, a_t) \leftarrow (1-\alpha) \cdot Q(s_t, a_t) + \alpha \cdot (r_t + \gamma \cdot \max Q(s_{t+1}, a)) \quad (10)$$

From the equation, we can see that Q-learning algorithm is performing according to the following steps:

- Initialize Table $Q(S, A)$ with random values.
- Take a action A with epsilon-greedy policy and move to next state.
- Update the Q value of a previous state by following the update equation.

The deep Q-network (DQN) algorithm is an upgrade of Q-learning, Q-table is replaced by Deep Neural Network in DQN. So the core procedure of DQN is to maximize the reward by solved action-value function Q. It is a powerful algorithm that can obtain strong performance in ATARI games, directly by learning from the pixels [7]. We want to see whether it can work well in portfolio management.

The definition of action space is a important part of DQN. Many previous work defined only three actions as (buy, sell, hold), which means the agent can only trade one stock for each timestep. This kind of definition is not suitable for this project because portfolio management has to reset multiple assets for each timestep. We define the actions as vectors so that one action can represent the trading of all stocks. Three trading types (buy, sell, hold) are encoded as (1, -1, 0). For example, the action of buy first stock and sell second stock at a timestep can be encoded as (1, -1). For each stock at each timestep, it has 3 trading types. As we have 19 stocks, our DQN action space has 3^{19} actions and each action is encoded as a 19-dimensional vector.

3.4 DDPG

As [6] mentioned, DQN can not be straightforwardly applied to continuous domains since it relies on finding the action that maximizes the action-value function. It can only handle discrete and low-dimensional action spaces. In our portfolio management problem, we have to simply discretize the action space. DDPG is an advanced DQN, which is a model-free, off-policy actor-critic algorithm using deep function approximators that can learn policies in high-dimensional, continuous action spaces.

The DDPG model consists of two separate networks for the actor and critic and follows the architecture from the paper[6]. The actor network aims to map states to actions. We consider time series is an important factor in the stock trading environment, so we used LSTM which is usually used to process and make predictions given sequences of data. The input is an observation vector, while the output is a vector with N values, one for each action. The output actions are transformed with hyperbolic tangent non-linearity to squeeze the values to the range (0,1).

The critic is a bit unusual, as it includes two separate paths for observation and the actions, and those paths are concatenated together to be transformed into the critic output of one number. The following is a diagram with the structures of both networks we implemented: Figure 1

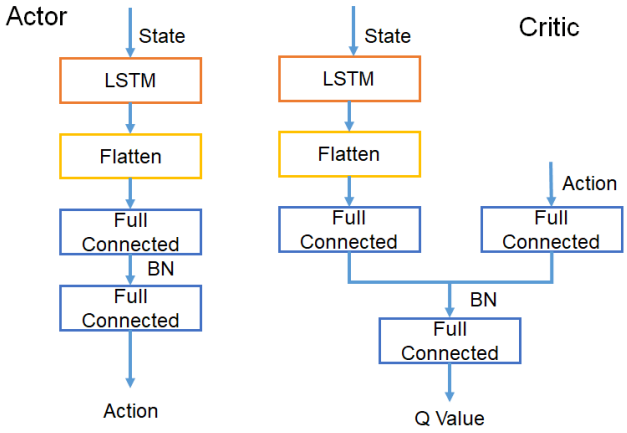

Figure 1: The structures of our model

4 EXPERIMENTS

4.1 Preprocess

4.1.1 Stock Table. Table 1 shows the information of 19 stocks we chose by industry.

4.1.2 Visualization. To obtain an overview of data, we visualised the price trending of 19 stocks in testing dataset. Figure 2 visualised the close price of stocks. Figure 2 visualised the percentage change of close price. From Figure 3, we can see it intuitively that there are some abnormal values since the one-day rise and fall of stocks should not exceed 10%. For example, the stock '600252' has a fall over 0.6 in Figure 2. We checked the raw data and there no mistake. The c lose price of stock '600252' on 2015-09-15 is 16.97 and on 2019-09-16 is 5.99. After search the historic announcement, we found the stock '600252' had a profit distribution and capitalisation of capital reserve into share capital event on 2015-09-15. That is the reason for the massive fall in the stock price. Since our model is not supposed to handle this kind of situation, we will treat this kind of data as usual bias.

4.1.3 Correlation Analysis. The correlation between stocks may affect our model. Thus, we do a pearson correlation analysis of the 19 stocks we chose. We computed the correlation and visualised it by heatmap (Figure 4) . From the heatmap we can assume that there is no apparent correlation between any of the 19 stocks.

4.2 DQN

4.2.1 Training performance. We trained the DQN model for 2000 episodes and 4000 episodes and the mean final returns are 61874.132

Table 1: 19 stocks

code	name	c_name
600757	长江传媒	传媒娱乐
600831	广电网络	传媒娱乐
600373	中文传媒	传媒娱乐
600551	时代出版	传媒娱乐
600854	春兰股份	家电行业
600870	厦华电子	家电行业
600983	惠而浦	家电行业
600336	澳柯玛	家电行业
600699	均胜电子	汽车制造
600303	曙光股份	汽车制造
600093	易见股份	汽车制造
600267	海正药业	生物制药
600252	中恒集团	生物制药
600196	复星医药	生物制药
600200	江苏吴中	生物制药
600198	大唐电信	电子信息
600288	大恒科技	电子信息
600485	信威集团	电子信息
600076	康欣新材	电子信息

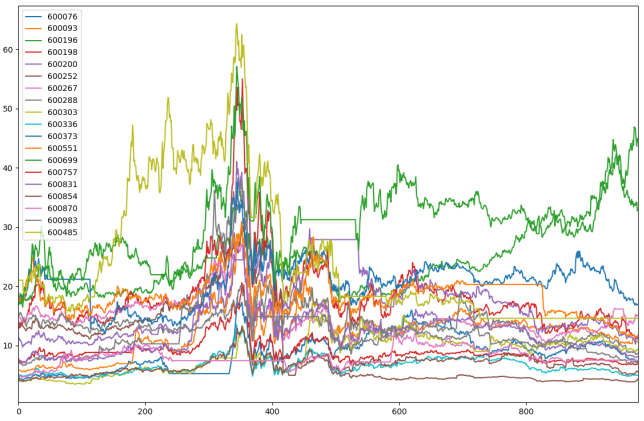

Figure 2: Close price of 19 stocks on the training dataset

and 116988.378. Figure 5 shows the change of final portfolio value during training.

4.2.2 Baseline. We use the "buy and hold" policy as our baseline. This policy will split the initial investment equally to buy as much stocks as we can at first and hold them until the end.

4.2.3 Testing performance. We used the trained weights and test it on the 2018-2019 dataset for 500 episodes. The mean return of DQN model is -2296.728 and the baseline is -6277. Although we cannot make good profits, we still get a much better performance than the baseline. The Figure 6 shows the result of one epoch. As we can see, the DQN keeps a more smooth performance compared with the baseline. When the market dropped rapidly, the DQN agent can maintain a smaller loss.

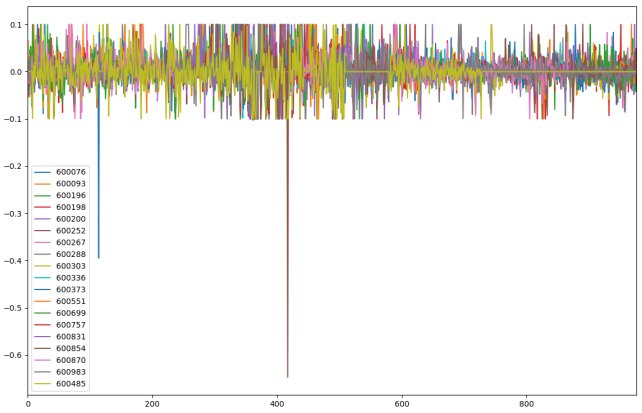

Figure 3: Percentage change of close price of 19 stocks on the training dataset

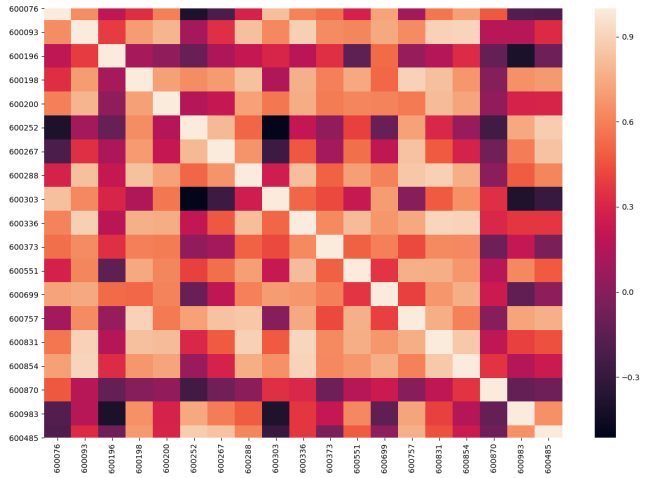

Figure 4: Heatmap of correlation

Figure 5: Training performance of DQN

4.2.4 Generalization performance. We also applied our DQN model on new stocks to test it generalization ability. Same as before, we tested for 500 episodes. The mean return of our agent is -2084.524 and the baseline is -3532.2.The Figure 7 shows the result of one

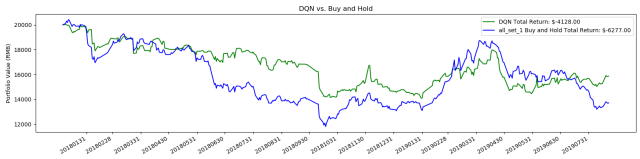

Figure 6: Testing performance of DQN

epoch. Same as shown in testing performance, the DQN model achieves a better performance than the baseline when the market value is dropping. The performance in new stocks is basically same as in the old stocks, which demonstrate that our model has a good generalization ability.

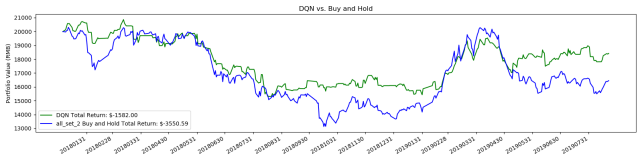

Figure 7: Generalization performance of DQN

4.3 DDPG

Training a reinforcement learning model is always difficult, we still do not get a satisfactory result so far.

4.3.1 Train Performance. Before analyze our result, we must know that:

- The window size is the number of the columns in each input price matrix.
- The reward value of our problem defined as follows the equation (3)
- Qmax value is the max q value output by critic network in each episode.

We choose reward and Qmax as evaluation metrics in training periods, our results are showed as Figure 8 and Figure 9

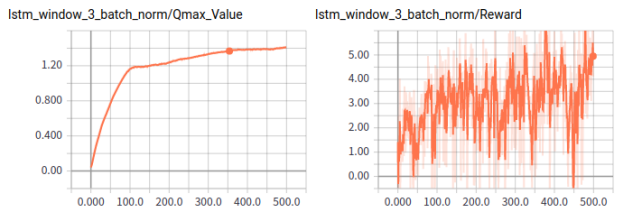

Figure 8: Training performance(windows size=3)

From these figures we can see that:

- 1. The reward value does not fully converge in epoch 500, but the overall trend is still rising.
- 2. If we set a small window size, we can get a better performance at last.

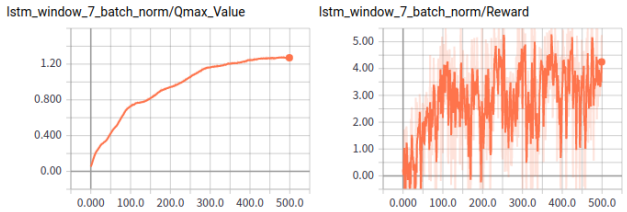

Figure 9: Training performance(windows size=7)

Table 2: The hyper parameters we used

parameter	value
episode	500
max step	960
batch size	64
actor learning rate	0.0001
critic learning rate	0.001
buffer size	100000
seed	1337

4.3.2 Baseline. We choose market value as baseline of our series experiments. The market value is obtained by equally distributing your investment to all the stocks. Before we train our model, we observe that Figure 10 the trend of market value is significantly different between the train data and the test data. This shows that

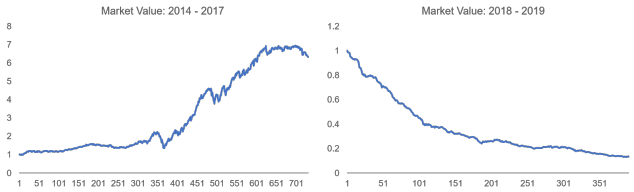

Figure 10: Baseline performance between train data and test data

the performance of the stocks we choose is particularly volatile, and our model will learn in the market which keep upward trend and test in the market which keep downward trend.

4.3.3 Test Performance. We set lots of hyper parameters in our experiments, it can be see in Table 2

We show the results of trading on testing data using our model in Figure 11 . We use sharp ratio and portfolio value to evaluate the performance of our model in test periods. The Sharpe Ratio is used to take risk into account. The ratio is a risk adjusted mean return, defined as the average of the risk-free return by its deviation.

As the result shows: Firstly, The portfolio value is better than market value in all experiments. It means that our model can make more profit than average distribute strategy. At least we prove that reinforcement learning can work in portfolio management problems. Secondly, it turns out that smaller window size works better. We think that maybe larger window size means larger models and it tends to overfit very quickly or maybe the temporal relationship

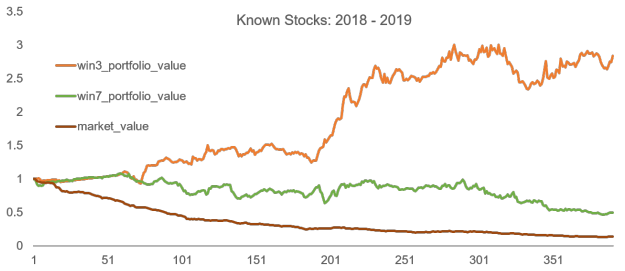

Figure 11: Testing performance. The sharpe ratio of win3 portfolio is 0.6813, win7 portfolio is -0.2622

among price in a short window can be captured better. Thirdly, unfortunately, the Sharp rate is still very low.

4.3.4 Generalization Test. In order to test the generalization ability of our model, we also run our agent on the stock which have not appear in our train periods. The results can be see in Figure 12.

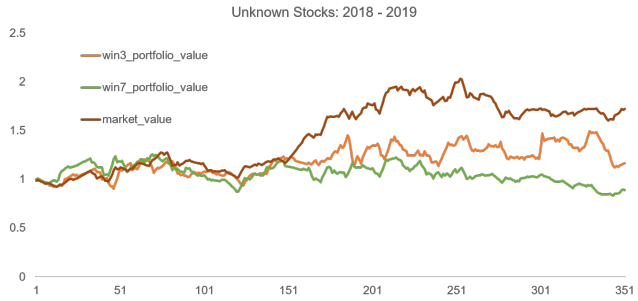

Figure 12: Generalization performance. The sharpe ratio of win3 portfolio is 0.1655, win7 portfolio is -0.0438

We can find that the generalization performance is quite bad. Our agent can not win the baseline. An interesting finding is that small window size are still better than large window size.

5 CONCLUSION

5.1 Contribution

We applied the deep RL methods in portfolio management in Chinese fund trading market by implementing DQN and DDPG models. Training with 2014-2017 dataset and testing in 2018-2019 dataset, both of DQN and DDPG can perform better than the market baseline.

In DQN model, we provided a vector kind of definition of action to encode the assets resetting in portfolio management. With three traing type (buy, sell, hold) encoded as (1, -1, 0), the trading action of 19 stocks at the same timestep can be encoded as a 19-dimensional vector. The portfolio value of baseline dropped heavily at some timesteps, while our DQN model can avoid these rapidly dropping down and achieve a smooth performance. Notice that training performance is much better than testing performance which may means with more training data we can get even better performance. Besides, as we can see in Figure 5, the final portfolio value kept growing and has not reached the stable state. Training the DQN model for more episodes may also improve the performance.

As an advanced deep reinforcement learning algorithm, DDPG can attain better performance than DQN, it can attain positive profit in test data while perform better than market value. We also develop a visualization tool by EchartJS to observe the dynamic change of the weight gave by our DDPG agent. We get the insight that most of the time step, our agent trend to choose only one or two, three stocks in hand. It is due to our reward function, the goal is to maximize the profit. So it keeps profit while take huge risk. Its the biggest disadvantage of our agent, we should consider the risk factor to our agent.

5.2 Challenge and future work

5.2.1 Model Training.

- The model is complex so that it is hard to debug.
- Our model is hard to coverage. It's the common problem in reinforcement learning model. We must add the episode.
- We must make our code structure more flexible so that we can perform more comparison experiments in the future.

5.2.2 Financial Events. As we demonstrated in Figure 3, some financial events can change the stock price greatly. Our model relies on stock price value heavily. Events like profit distribution and capitalisation of capital reserve into share capital are not our target problem.Thus, we treat these as common bias. However, this kind of events can effect the performance of model. Besides, in real world, it is necessary to take these financial events into account.To consider this kind of situation, text data like news should be included in future work.

5.2.3 Computation resources limitation. As mentioned before, the models may get better performance if training on a larger dataset. At present, the dataset is on day-scale. We had tried to train the model on minute-scale but failed due to the computation resouces limitation. With more computation resources or more time, it is worthy to train the model on minute-scale.

REFERENCES

[1] Limian Zhang Chi Zhang, Corey Chen. 2017. Deep Reinforcement Learning for Portfolio Management. http://www-scf.usc.edu/~zhan527/post/cs599/.

[2] Yue Deng, Feng Bao, Youyong Kong, Zhiquan Ren, and Qionghai Dai. 2016. Deep direct reinforcement learning for financial signal representation and trading. *IEEE transactions on neural networks and learning systems* 28, 3 (2016), 653–664.

[3] JB Heaton, NG Polson, and Jan Hendrik Witte. 2017. Deep learning for finance: deep portfolios. *Applied Stochastic Models in Business and Industry* 33, 1 (2017), 3–12.

[4] Zhengyao Jiang, Dixing Xu, and Jinjun Liang. 2017. A deep reinforcement learning framework for the financial portfolio management problem. *arXiv preprint arXiv:1706.10059* (2017).

[5] Zhipeng Liang, Kangkang Jiang, Hao Chen, Junhao Zhu, and Yanran Li. 2018. Deep reinforcement learning in portfolio management. *arXiv preprint arXiv:1808.09940* (2018).

[6] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2015. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971* (2015).

[7] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* (2013).

[8] Seyed Taghi Akhavan Niaki and Saeid Hoseinzade. 2013. Forecasting S&P 500 index using artificial neural networks and design of experiments. *Journal of Industrial Engineering International* 9, 1 (2013), 1.

[9] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.