Deep Reinforcement Learning for Portfolio Management*

Chi Zhang, Limian Zhang, and Corey Chen

Department of Computer Science

November 27, 2017

1 Problem Definition

1.1 Notations

In this project, we would like to manage portfolio by distributing our investment into a number of stocks based on the market. We define our environment similar to this paper [1]. Concretely, we define N to be the number of stocks we would like to invest. Without losing generality, at initial timestamp, we assume our total investment volume is 1 dollar. We define *close/open relative price vector* as:

$$y_t = [1, \frac{v_{1,t,close}}{v_{1,t,open}}, \frac{v_{2,t,close}}{v_{2,t,open}}, \cdots, \frac{v_{N,t,close}}{v_{N,t,open}}] \tag{1}$$

where $\frac{v_{i,t,close}}{v_{i,t,open}}$ is the *relative price* of stock i at timestamp t. Note $y[0]$ represents the relative price of cash, which is always 1. We define *portfolio weight vector* as:

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

where $w_{i,t}$ represents the fraction of investment on stock i at timestamp t. Note that $w_{0,t}$ represents the fraction of cash that we maintain. Then the profit after timestamp T is:

$$p_T = \prod_{t=1}^{T} y_t \cdot w_{t-1} \tag{3}$$

where $w_0 = [1, 0, \cdots, 0]$. If we consider a trading cost factor μ, then the trading cost of each timestamp is:

$$\mu_t = \mu \sum |\frac{y_t \odot w_{t-1}}{y_t \cdot w_{t-1}} - w_t| \tag{4}$$

where $\odot$ is element-wise product. Then equation 3 becomes:

$$p_T = \prod_{t=1}^{T} (1 - \mu_t) y_t \cdot w_{t-1} \tag{5}$$

1.2 Key Assumptions and Goal

To model real world market trades, we make several assumptions to simplify the problems:

- We can get any information about the stocks before timestamp t for stock i. e.g. The previous stock price, the news and tweets online.

*Instructor: Joseph J. Lim

- Our investment will not change how the market behaves.
- The way we calculate profit in equation 3 can be interpreted as: At timestamp t, we buy stocks according to the *portfolio weight vector* w_{t-1} computed by history data at **open** price and sell all the stocks at **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.

The **goal** of portfolio management is to maximum p_T by choosing portfolio weight vector w at each timestamp t based on history stock information.

1.3 MDP formulation

1.3.1 State and Action

We define state s_t as o_t, where o_t is the obseration of timestamp t. As the time goes by, the impact of history data decreases. Thus, we only consider the history price and news in a fixed window length W. Hence,

$$o_t = [\vec{v_{1,t}}, \vec{v_{2,t}}, \cdots, \vec{v_{N,t}}] \tag{6}$$

where

$$v_{i,t} = \begin{bmatrix} v_{i,t-W} \\ v_{i,t-W+1} \\ \vdots \\ v_{i,t-1} \end{bmatrix} \tag{7}$$

and N is the number of stocks. The action a_t is just *portfolio weight vector* w_t. Note that this is a continuous state and action space problem. We try to directly solve it in continuous space instead of using discretization in previous work [2, 3]. Essentially, we want to train a policy network $\pi_\theta(a_t|o_t)$.

1.3.2 State Transition

The underlining state evolution is determined by the market, which we don't have any control. What we can get is the observation state, which is the price. Since we will collect history price of various stocks, o_t is given by the dataset instead of o_{t-1}.

1.3.3 Reward

Instead of having reward 0 at each timestamp and p_T at the end, we take logarithm of equation 5:

$$\log p_T = \log \prod_{t=1}^{T} \mu_t y_t \cdot w_{t-1} = \sum_{t=1}^{T} \log(\mu_t y_t \cdot w_{t-1}) \tag{8}$$

Thus, we have $\log(\mu_t y_t \cdot w_{t-1})$ reward each timestamp, which avoids the sparsity of reward problem.

1.4 Datasets

Stocks used: We use 16 target stocks from NASDAQ100 that we feel are representative of different sectors in the index fund. They include "AAPL", "ATVI", "CMCSA", "COST", "CSX", "DISH", "EA", "EBAY", "FB", "GOOGL", "HAS", "ILMN", "INTC", "MAR", "REGN" and "SBUX".
Price Data: We collected history price of the stocks from 2012-08-13 to 2017-08-11. The price on each day contains open, high, low and close. We use 2012-08-13 to 2015-08-12 as training data and 2015-08-13 to 2017-08-11 as testing data.

News Data: We gather all tweets referencing the stocks from 2016-03-28 to 2016-06-15.
Additional Testing Data: As another form of backtesting, we randomly select 16 stocks from NASDAQ100 the network has never seen and test whether the network generalizes.

2 Methods

We mainly consider model-free approach in our project. First, we train a predictor given a fixed history window length W of price and news. With the predicted price, we can train a policy network $\pi_\theta(a_t|s_t)$ to maximize our portofolio value at the end of the trading period. Note that in practice, they are trained end-to-end instead of separately.

2.1 Data Preprocessing

Instead of using the raw price data, we normalize the history price as $(\frac{close}{open} - 1) \times scale$. There are two main advantages:

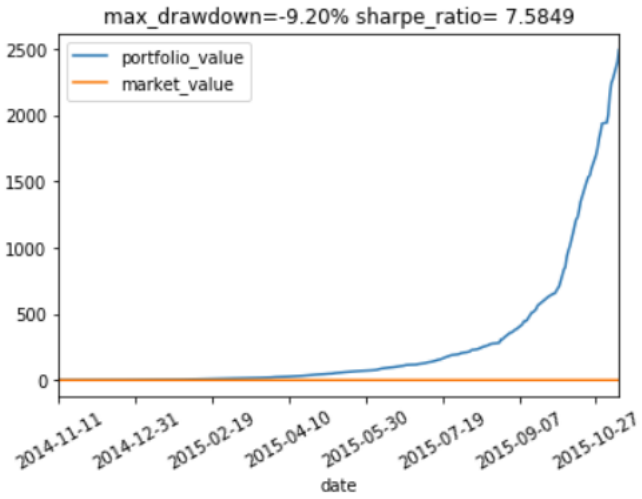
1. The history price are all in the same scale regardless of their actual price.
2. Since the final portfolio is determined by the close/open ratio of each timestamp, it serves as a better feature compared with raw price.

The *scale* factor is heuristically set to 100 in our experiments.

2.2 Predictor Network

2.3 Optimal Action and Imitation Learning

If we have a perfect predictor, then we can ignore risk and just pick the best stock for each day. We greedily choose the stock with the highest close/open ratio (taking into account trading cost of changing stocks), buying as much as possible on the open and selling all at the close. Here is one such possible monotonically increasing output:



Then, we can train a model by imitating optimal action: we have as input the previous state (o_t, a_{t-1}), and as output label we compute to our next day's action. We will be using a CNN and LSTM to do this and compare the results. I believe this approach will yield good outcomes if the model can generalize to unseen data.

2.4 Deep Deterministic Policy Gradient (DDPG)

We applied similar reinforcement learning algorithm for continuous action space in this paper [4]. However, we don't have any promising results yet. The main issue is that the model doesn't seem to be training because the action the actor network produces are all equally distributed weights. e.g. $[0.25, 0.25, 0.25, 0.25]$ if we have 3 stocks plus cash. I want to take some time to debug the network by seeing how the network is updated by the gradient at each step. Also, a key part is the exploration noise we added when sample actions. Currently, I just use uncorrelated Gaussian noise and the result is very bad. I would like to try parameter noise mentioned here [5].

2.4.1 Critic Network Topology

3 Results and Discussions

4 Contribution

Chi Zhang:

- Collect and preprocess stock price.
- Set up environment (OpenAI gym).
- Train DDPG model. (Doesn't work very well now, need to dig more)

Corey Chen:

- Compute optimal action
- Train CNN and LSTM policy network using imitation learning. (TODO)

Limian Zhang:

- Collect and preprocess tweets datasets.
- Predict future stock price based on history price and tweets data (TODO)

References

[1] Z. Jiang, D. Xu, and J. Liang, "A Deep Reinforcement Learning Framework for the Financial Portfolio Management Problem," *CoRR*, vol. abs/1706.10059, 2017. [Online]. Available: http://arxiv.org/abs/1706.10059

[2] O. Jin and H. El-Saawy, "Portfolio management using reinforcement learning."

[3] X. Du, J. Zhai, and K. Lv, "Algorithm trading using q-learning and recurrent reinforcement learning," http://cs229.stanford.edu/proj2009/LvDuZhai.pdf.

[4] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous Control with Deep Reinforcement Learning," *CoRR*, vol. abs/1509.02971, 2015.

[5] M. Plappert, R. Houthooft, P. Dhariwal, S. Sidor, R. Y. Chen, X. Chen, T. Asfour, P. Abbeel, and M. Andrychowicz, "Parameter Space Noise for Exploration," *CoRR*, vol. abs/1706.01905, 2017.