Deep Reinforcement Learning for Portfolio Management[*]

Chi Zhang, Limian Zhang, and Corey Chen

Department of Computer Science

November 8, 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, a_{i-1}) pair, where o_t is the obseration of timestamp t. Since we can't feed variable length data into CNN/LSTM models, we only consider the history price in a window length W. Thus,

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

where

$$v_{i,t} = \begin{bmatrix} v_{i,t-W+1} \\ v_{i,t-W+2} \\ \vdots \\ v_{i,t} \end{bmatrix} \tag{7}$$

The action a_{i-1} is just *portfolio weight vector* w_{i-1}. In this problem, we would like to train a policy network $\pi_\theta(a_t|s_t) = \pi_\theta(a_t|o_t, a_{t-1})$. We will directly use $s_t = (o_t, a_{i-1})$ in our following discussion.

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.

1.3.4 Discount Factor

The discount factor γ is 1 because future rewards are as important as current rewards.

1.4 Datasets

Stocks used: We are currently using 16 stocks from NASDAQ100 that we feel are representative of different sectors in the index fund.
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, close, volume). 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 have gathered all tweets referencing the stocks from 2016-03-28 to 2016-06-15.

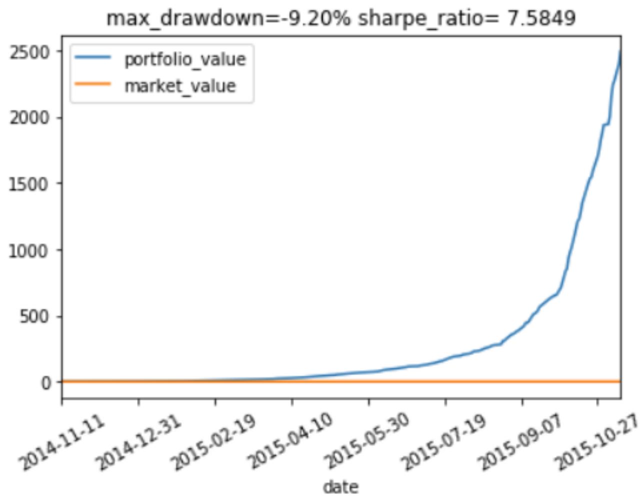
2 Methods

We consider both model-free and model-based approach. The main difference is that in model-based approach, we try to predict the open and close price of future timestamp before making actions. In model-free approach, we try to learn policy $\pi_\theta(a_t|s_t)$ directly from history data.

2.1 Model-free Approach

2.1.1 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.

2.1.2 Deep Deterministic Policy Gradient (DDPG)

We applied similar reinforcement learning algorithm for continuous action space in this paper [2]. 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 [3].

2.2 Model-based Approach

2.2.1 Price/Risk Prediction

Using pre-trained word embeddings from GloVe, we analyze sentiment from the tweets to determine a numerical effect of the tweets on the stock price. We combine this with some historical prices to build an LSTM to predict the next day's price and potential risk.

2.2.2 Planning

Given a predictions on the next day's price and risk (represented as covaraiances), we want to build a balanced portfolio that maximizes return while minimizing risk. Details on how to do this still need to be fleshed out, but in the end we are looking at maximizing the Sharpe ratio:

$$s = \frac{p_T - p_i}{\sigma} \tag{9}$$

where p_i represents risk-free inflation rate (1.02 per year) and σ is the variance of the entire portfolio.

2.2.3 End-to-End training

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] 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.

[3] 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.