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 and $\sum_{i=0}^{N} w_{i,t} = 1$. 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.

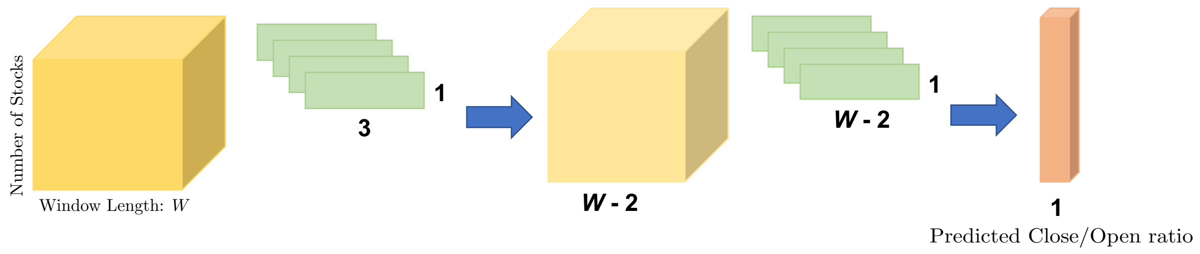

Figure 1: CNN predictor based on numerical history

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

2.1.1 Numerical Data

Instead of using the raw open, high, low and close price, 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. For missing data, we pad all open, high, low, close as the close price of previous day.

2.1.2 News Data

The steps to proprocess the news data are:

1. First, we filter each tweets by the most frequenct 2000 words.
2. We pad each word with end of sentence mark to make them equally long.
3. We use a sequence of end of sentence mark to fill the missing days.

2.2 Predictor Network

We show three predictors we train in Figure 1, Figure 2 and Figure 3. The LSTM predictor based on numerical history is very straight forward. Note that all the stocks share the same LSTM. The rationale behind the CNN predictor is adpated from [1]. For each timeseries of each stock, we use a 1×3 kernel to gather information in each window, then combine all the information to produce a single vector for each stock. Note that these convolutional windows doesn't cover information across any two stocks. For news based approach, we try to predict the $\frac{close-open}{open}$ ratio. We use pretrained 50d-GloVe to initialize the embedding layer followed by a LSTM and a fully-connected layer.

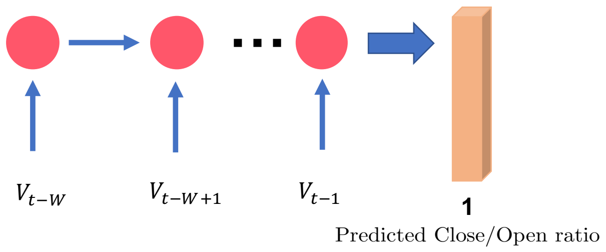

Figure 2: LSTM predictor based on numerical history

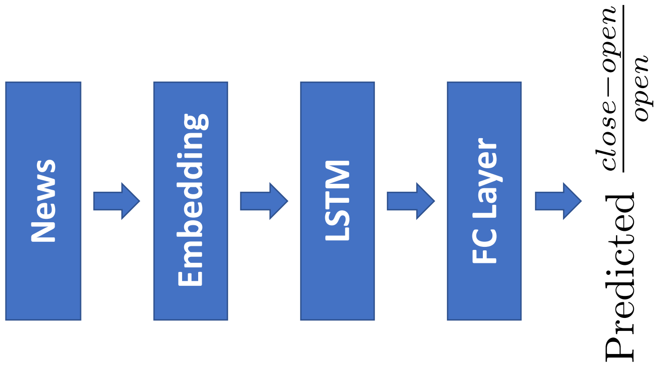

Figure 3: LSTM predictor based on history of news

2.3 Policy Network

We show the policy network topology in Figure 4. All the policy network follows the same topology with different predictors.

2.4 Optimal Action and Imitation Learning

Suppose we know the stock price of tomorrow, 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.

Given this fact, we can collect ground truth labels for each timestamp by choosing the optimal action. (e.g. $[0, 0, 1, \cdots, 0, 0]$, the stock with highest close/open to be 1 and all the others to be 0) Then, we

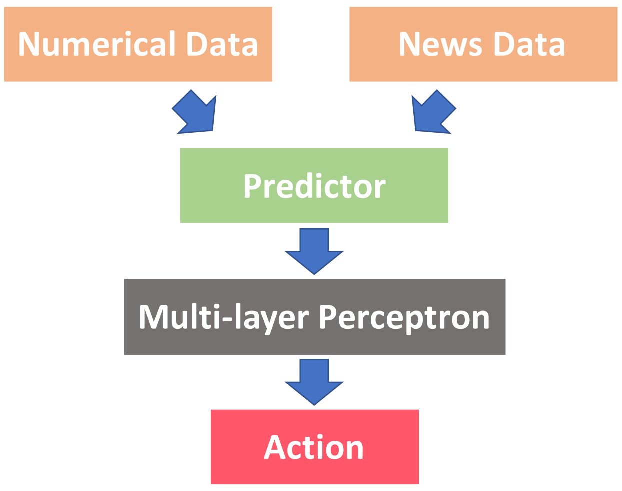

Figure 4: Policy Network

Algorithm 1 DDPG algorithm

```
Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights $\theta^Q$ and $\theta^\mu$.
Initialize target network $Q'$ and $\mu'$ with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer $R$
for episode = 1, M do
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state $s_1$
  for t = 1, T do
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action $a_t$ and observe reward $r_t$ and observe new state $s_{t+1}$
    Store transition $(s_t, a_t, r_t, s_{t+1})$ in $R$
    Sample a random minibatch of $N$ transitions $(s_i, a_i, r_i, s_{i+1})$ from $R$
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:
        $\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s_i}$
    Update the target networks:
        $\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$
        $\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$
  end for
end for
```

Figure 5: Deep Deterministic Policy Gradient Algorithm[4]

can train a policy network by imitating optimal action conditioned on the current state s_t. The policy network is a three layer MLP.

2.5 Deep Deterministic Policy Gradient (DDPG)

We try to directly learn a policy network $\pi_\theta(a_t|s_t)$ by continuous action space reinforcement learning algorithm [4]. We show the algorithm in Figure 5. The configurations are:

- Actor Network: predictor in Section 2.2 plus a three layer perceptron.
- Critic Network: a linear combination of actor network structure of state (observation) and action.
- Exploration noise: Ornstein-Uhlenbeck with zero mean, 0.3 sigma and 0.15 theta.
- For fairness, we train models with different settings in 500 episodes. Some of the models are not fully converged at that time though.

3 Results

3.1 Imitation learning

We show the result of trading on testing data using policy trained by imitation learning in Figure 6. The market value is obtained by equally distributing your investment to all the stocks. It turns out that smaller windows work better. Larger window means larger models and it tends to overfit very quickly since the training data is just around 1000.

3.2 DDPG

We show the result of trading on testing data using policy trained by DDPG in Figure 7. We find that LSTM predictor based models have better performance than the CNN predictor based models. It is not surprising that smaller windows perform best because of the temporaral relationship among price in a short window.

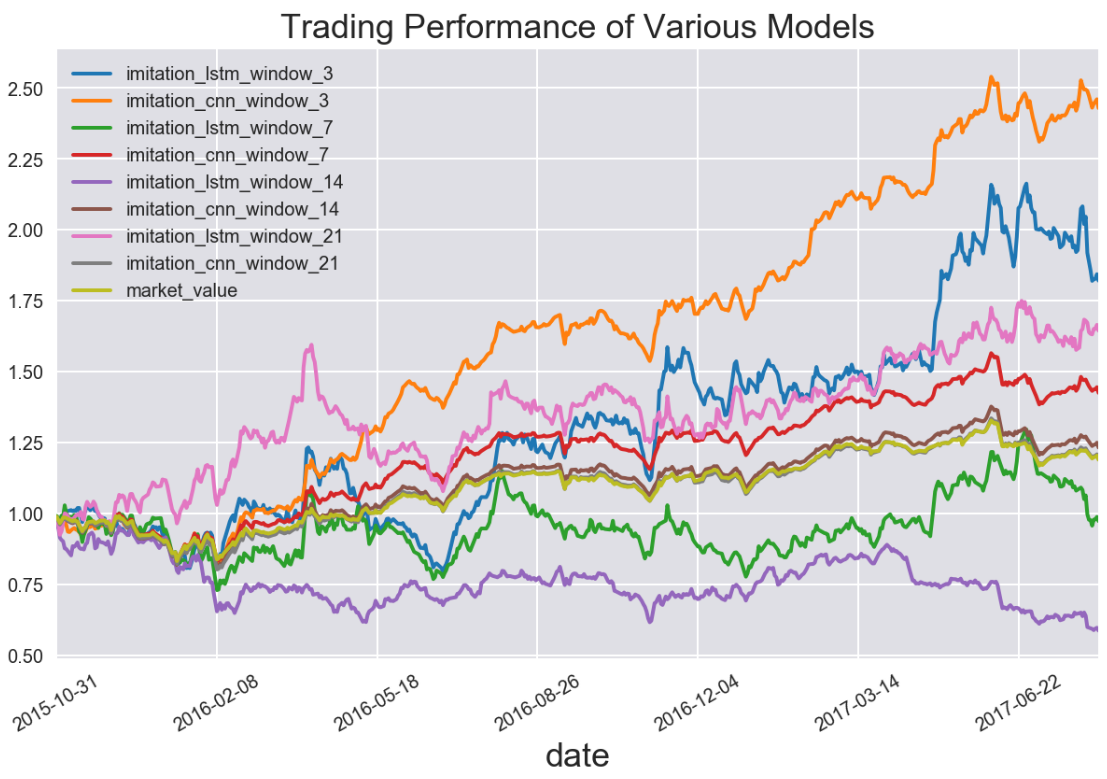

Figure 6: Results of trading on testing data using policy trained by imitation learning

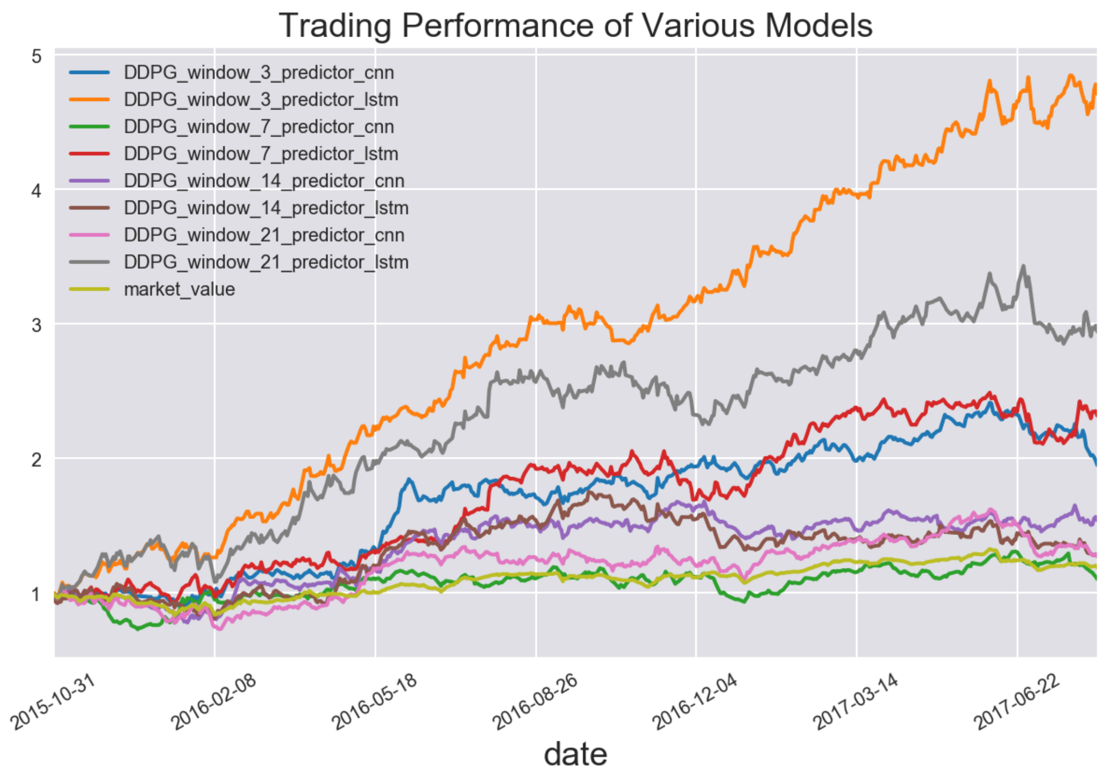

Figure 7: Results of trading on testing data using policy trained by DDPG

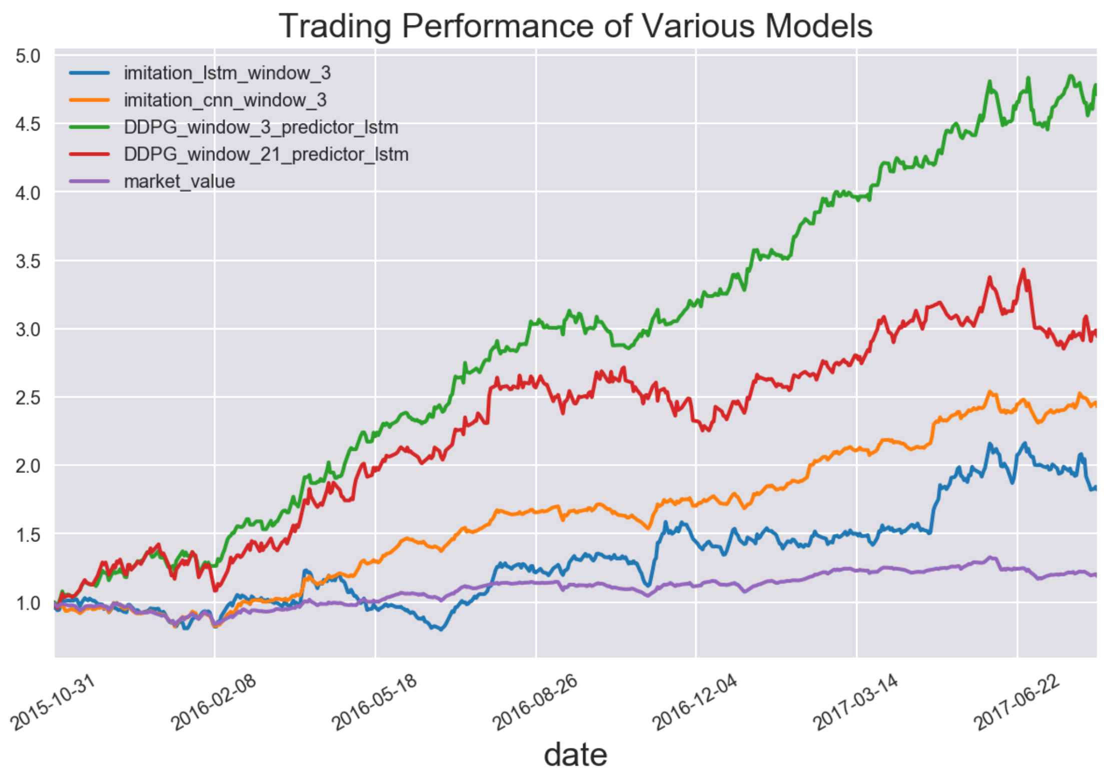

Figure 8: Results of trading on testing data using best two networks trained by imitation learning and DDPG

3.3 Imitation vs DDPG

Generally, models trained by DDPG is better than models trained by imitation learning as shown in figure 8. The problem of training good networks by imitation learning is the extreme prone to overfitting while DDPG doesn't suffer from this problem because each episode is sampled from the whole training period with different starting date and lengths.

3.4 Related Work

In [2, 3], the author proposes to use DQN to trade in 2 stocks market with discreted state and action space. Their settings are far simpler than ours. In [1], the author proposes to use Deterministic Policy Gradient (DPG) to trade in bitcoin market. It's very hard to compare with them because of different dataset. But DDPG is strictly better than DPG in terms of performance and convergence time.

4 Contribution

Chi Zhang:

- Collect and preprocess stock price.
- Set up environment (OpenAI gym).
- Train DDPG model.
- Write report.

Corey Chen:

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

Limian Zhang:

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

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.