Double DQN Method in Stock Trading Problem

Shiqin Sun[*], Kun Han, Qi Duan, Kun Wang, Xuanxian Du

October 15, 2018

Abstract

In this paper, we implement Double DQN model for stock trading problems. Different from supervised learning models, Double DQN model is a reinforcment learning model, whose outputs are trading signals of stock. This makes our model better accord with the concept of artificial intelligence. We choose eight stocks from S&P500 for the backtest and improve the strategy with respect to leverage. The backtest analysis for two version of model both show promising results.

1 Introduction

The problem focuses on using machine learning method in selecting a basket of stable-alpha-stocks, which has been a necessary and challenging research area in financial trading decision-making. The traditional models of solving this kind of problems are highly contingent upon reliability of prediction of future performance of stocks and appropriate portfolio construction, including several machine learning methodologies such as fuzzy systems, artificial neural networks (ANNs), evolutionary algorithms (EAs), support vector machine (SVMs) and etc. However, our model will mainly focused on reinforcement learning (RL), which is designed to improve actions and predict implicitly.

The goal of reinforcement learning (Sutton and Barto 1998) is to train software agents ought to take actions in an environment so as to maximize some notion of cumulative reward. The advantage of RL is that it does not depend on supervisory learning network. Thus, our model's outputs are actions but not estimation, which avoid risk of false prediction. Because of this nature of RL, outputs of our agent satisfy the concept of artificial intelligence. Among models of RL, Q learning (Watkins 1989) does not require a model of the environment and can handle problems with stochastic transitions and rewards, without requiring adaptations. Under a given policy π, the true value of an action a_t in a state s_t is

$$Q^{\pi}(s_t, a_t) = E[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots | s_t, a_t], \tag{1}$$

[*]Problem 01 Team JHUFM

where $\gamma \in [0, 1]$ is a discount factor that trades off the importance of immediate and later rewards.

Although Q learning is a popular reinforcement learning algorithm, it has some problems on overestimations, which means action values are tended to be overestimated. In order to improve the situation of imprecise value estimates, deep Q-learning (DQN) algorithm has been proposed. The updated action value function is

$$Q^*(s_t, a_t) = Q^\pi(s_t, a_t) + \alpha(r_{t+1} + \gamma \max_{a_{t+1}} Q^\pi(s_{t+1}, a_{t+1}) - Q^\pi(s_t, a_t)), \tag{2}$$

where α is the learning ratio, and the loss function of the neural network is

$$L(\theta_t) = E[(TargetQ - Q(s_t, a_t; \theta_t))^2], \tag{3}$$

$$TargetQ = r_{t+1} + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}; \theta_t). \tag{4}$$

DQN combines Q-learning with a flexible deep neural network and was tested on a varied and large set of games, reaching human-level performance on many games. The deep neural network provides flexible function approximation with the potential for a low asymptotic approximation error. Thus, DQN algorithm, to an extent, has become a good setting for Q-learning.

Nevertheless, it is not surprising to illustrate that even in this comparatively favorable setting DQN sometimes substantially does overestimation on action values. Then, Double DQN methodology has been proposed. The idea of Double Q-learning is to reduce overestimations by decomposing the max operation in the target into actionselection and action evaluation. Although not fully decoupled, the target network in the DQN architecture provides a natural candidate for the second value function, without having to introduce additional network. (Van Hasselt, Guez, & Silver, 2016) The two different functions about the target Q are

$$TargetQ = r_{t+1} + \gamma Q(s_{t+1}, \arg\max_a Q(s_{t+1}, a; \theta_t); \theta_t), \tag{5}$$

$$TargetQ^{'} = r_{t+1} + \gamma Q(s_{t+1}, \arg\max_a Q(s_{t+1}, a; \theta_t); \theta_t^{'}). \tag{6}$$

The two different function approximators are trained on different samples, one for selecting the best action and other for calculating the value of this action, since the two functions approximators seen different samples, it is unlikely that they overestimate the same action.

In a nutshell, the methodology we proposed here is to use the Double DQN to train a software agent. We will report the trading signals given by our well-trained agent on eight stocks selected from technology sector of S&P500.

2 DQN Model Implementation

In this section, we will first give a description on the structure of the deep neural network we use in this model, which includes number of layers, size of each layer and activation functions.

2.1 Q Network Structure

Number of Layers We use three-layer neural network model, containing two hidden layers and one output layer. Although fancier networks can be implemented in our model, considering the degree of complication, we decide to choose a comparatively simple setting to present the advantage of Double DQN without redundant computational complexity.

Sizes of Layers The size of input layer is 91, consisting of 90-day previous close prices and current position. The highlighting point here is the application of current position. Without current position as an input, the deep network model is actually just supervised learning model for prediction. The use of current position in the inputs emphasizes the nature of reinforcement learning.

For other layers, we set hidden layers to contain 100 nodes, while the output layer having three nodes. The size of hidden layers is set arbitrarily. The larger hidden layers are, the more complex the model will be. As our actions only contain "buy", "sell" and "hold", the outputs are automatically three-dimensional vectors.

Activation Functions In terms of activation functions, we use Rectified Linear Unit functions (ReLU) for the first two layers and softmax function for the output layer, which is a regular setting for deep neural network.

2.2 Parameters

The parameters here fall into two parts. The first part is related to Q learning while second part focusing on traning.

Parameters of Q Learning This part of parameters are inline parameters of Q learning, including discount factor γ, train frequancy, upadate frequncy as well as memory size. We set these parameters as follows,

$$\gamma = 0.97, \tag{7}$$
$$\text{train\_freq} = 10, \tag{8}$$
$$\text{update\_freq} = 20, \tag{9}$$
$$\text{memory\_size} = 500. \tag{10}$$

The reason that we choose 500 as memory size here is that 500 approximates two years of trading dates. We consider that time of two years is a proper length of the natural economical cycle. Therefore, 500 is a reasonable memory size for trading.

Parameters of Training This part of parameters serve for traning process, including batch size, epoch number and etc.

$$\text{batch\_size} = 50, \tag{11}$$
$$\text{epoc\_num} = 40. \tag{12}$$

In terms of learning algorithms, we choose Adam (Kingma & Ba, 2014), which is a commonly used algorithm, with default settings.

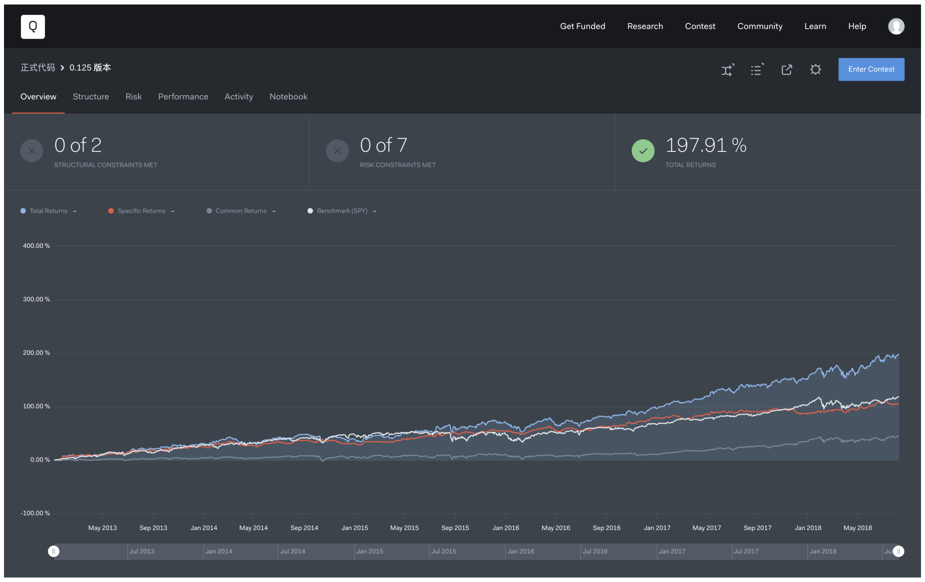
The main function is attached in the zip-file along with this report.

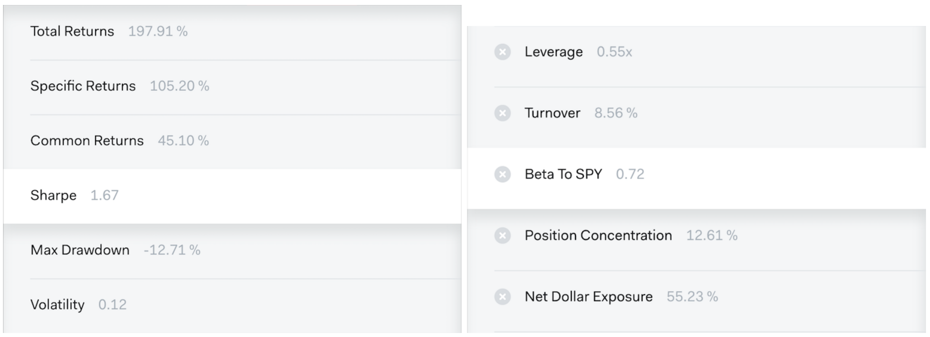
3 Backtest

Due to the fact that different stocks have various size of total sample, for all stocks, we split the data into two parts. Regardless of the different sizes, we take all data before 2013/1/4 as training data, and generate trading signals based on samples after 2013/1/4 via our well-trained Double DQN agent. The csv file of all trading signals starting from 2013/1/4 is attached in the zip-file.

Stocks in our portfolio comprise GOOG, AAPL, MSFT, AMZN, TSLA, NFLX, NVDA. We can only trade 1/8 of the total money to long one stock once we get a signal for this stock. This can be interpreted as the equal weight portfolio. As all the stocks we select are from S&P500, we choose S&P500 as benchmark portfolio for the backtest analysis. All the backtest results are generated from Quantopian.

The result is listed as follows:





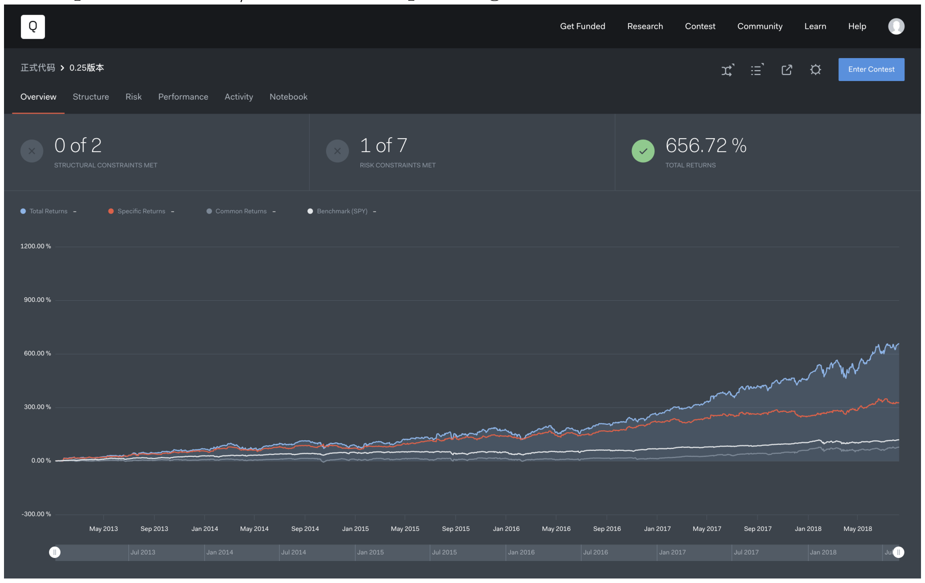
Total Returns	197.91 %
Specific Returns	105.20 %
Common Returns	45.10 %
Sharpe	1.67
Max Drawdown	-12.71 %
Volatility	0.12

Leverage	0.55x
Turnover	8.56 %
Beta To SPY	0.72
Position Concentration	12.61 %
Net Dollar Exposure	55.23 %

From the diagram, we can observe, that our method surpass the benchmark portfolio and get a 1.5 times final return.The white line denotes the benchmark portfolio. All the other diagrams of analysis feature are attached in the zip-file, named as ”0.125version”.

From the diagram, our method has the advantage of the high sharpe ratio and low turnover. However, we also have the disadvantage of high β. The reason for this high β is because when use 90 previous return as the training input, this leads to the fact that the generated trading signals are less sensitive to the price oscillation. As a result, output signals tend to be of low frequency, which causes the high β and low turnover. To cope with this, we can change the inputs to 30 previous prices and the current position. In addition, reduce γ would also help. However, though β decreases, turnover would increase correspondingly, which introduces a trade-off that needs further research.

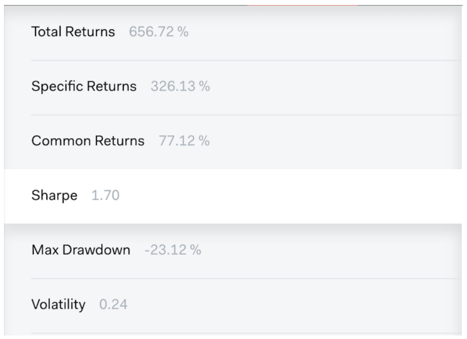
From this example, we also observe that the average leverage of the portfolio is 0.55. This means that half of the money is not used for stock trading. To make full use of our portfolio, under the same trading signal, we make maximum share potion to be 1/4. The corresponding result is listed as follow:



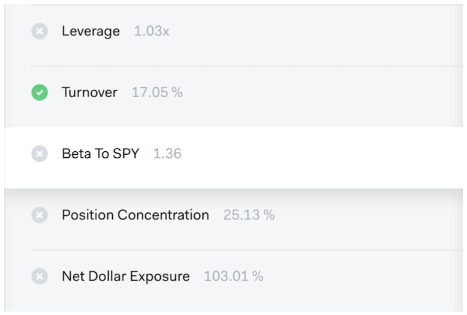
0 of 2 STRUCTURAL CONSTRAINTS MET

1 of 7 RISK CONSTRAINTS MET

656.72 % TOTAL RETURNS



Total Returns	656.72 %
Specific Returns	326.13 %
Common Returns	77.12 %
Sharpe	1.70
Max Drawdown	-23.12 %
Volatility	0.24



⊗	Leverage	1.03x
✓	Turnover	17.05 %
⊗	Beta To SPY	1.36
⊗	Position Concentration	25.13 %
⊗	Net Dollar Exposure	103.01 %

The new version shows a promising result with over 6 times return. As expected, the portfolio still possesses the problem of high β, but the satisfactory total return and sharpe ratio justify our improving points. All the other diagrams of analysis feature are attached in the zip-file, named as ”0.25version”.

4 Conclusion

In this research project, we implement Double DQN model to trade stocks. In contrast to normal supervised learning models, our model output trading signals instead of future return predictions. Therefore, our method suffice to accord with the concept of artificial intelligence. The backtest gives out a positive feedback, which justifies our innovative application.

We provide two tunable facts for future research on our model. Initially, more complicated training environment of Q-function can be introduced. In our attempt, 1/8 portion of the total asset is our only option if we long a stock. However, more flexible choices of share holdings shall be considered. In addition, robustness of the model needs to be tested. Different from supervised learning, reinforcment learning models have the disadvantage of less robustness. This shortcoming derives from the stochastic nature of the model construction. Therefore, the randomness of our model should be taken into account.

References

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Van Hasselt, H., Guez, A., & Silver, D. (2016). Deep reinforcement learning with double q-learning. In *Aaai* (Vol. 2, p. 5).