Practical Deep Reinforcement Learning Approach for Stock Trading

Anonymous Author(s)
Affiliation
Address
`email`

Abstract

Portfolio allocation plays a crucial role to the success of investment companies, since it determines the return of an investment portfolio. Our key idea is to use our deep reinforcement algorithm to do portfolio allocation from a random underlying environment and finding the optimum policy which will give maximum return under certain risk. First, we dynamically select top 20% stocks using machine learning prediction model. Secondly, we trained deep reinforcement learning agent to do portfolio allocation among these stocks we have previously select. Thirdly, we test our agent's performance compared with benchmark algorithms such as mean-variance and minimum variance. Our results show that the proposed system outperforms the original strategy on the S&P 500 index in terms of Sharpe ratio and cumulative returns.

1 Introduction

Allocation optimization is vital for portfolio managers. They use this to optimize allocation of capital and thus maximize performance like expected returns. Return forecasts are based on analysts' estimation of companies profitability and risk. However, it is very difficult for analysts to combine all the elements together in complex stock market.

Traditional method performed in two steps as described in Markowitz [4]. First, the expected returns of stocks is necessary to be computed, and the covariance matrix of the returns of the stocks as well. Then, solve this problem by either maximize the return for a fixed risk or minimize the risk of the portfolio for a range of return. Typical methods are Mean-Variance and minimum-variance, which perform diversification by constraining mean, volatility and correlation inputs to reduce sampling error[5]. But this problem can be extremely complicated if the manager wants to revise every decision he made at each time step and take transaction cost into consideration.

Reinforcement learning can also be used to solve this problem. By formalizing our problem as a Markov Decision Process (MDP), we can then solving this problem of finding optimum policy by solving the dynamic programming problem. Several algorithms have been used to solved this problem, such as Q-Learning in Neueier [6] and enhancing Q-Learning Neueier [7]. In these works, neural networks are used as value function approximations. However, the stability and convergency of original Q-Learning algorithm are not good. The structure need to changed in order to obtain better performance.

In this paper, we propose a practical deep reinforcement learning approach for stock trading. First, we based on previous research of using machine learning to predict reward to select the 20 percents of stocks. Then, we implement our improved Deep Q Network to learn how to allocate asset among selected stocks. The improved algorithm has two key structures, experience replay and fixed Q-target. The first one break the relationship between samples by selecting sample randomly , the second one

Submitted to 32nd Conference on Neural Information Processing Systems (NIPS 2018). Do not distribute.

Table 1: 20 Indicators

Part		
Name	Description	Size (μm)
Dendrite	Input terminal	$\sim$100
Axon	Output terminal	$\sim$10
Soma	Cell body	up to 10^6

use a slower updated Q network to offer Q-target value so that the stability and convergency are increased. Finally, we compare the P&L performance of our method with S&P 500 index, equal, Mean Variance and Minimum Variance portfolio allocation method. Our strategy got higher return than all other allocation methods and market, proving the competency of our strategy.

This paper proceeds as follows. Section 2 describes the data, preprocessing method and machine learning model to do the selection. Section 3 contains main reinforcement models for portofilio allocation and two problem statements. Section 4 states solution in this paper to address these problems. Section 5 presents and analyze the performance. Section 6 concludes the paper.

2 Data Preprocessing and Stock Selection

2.1 Data-set and Features

The data for this paper is from Compustat database accessed through Wharton Research Data Services (WRDS) [8], ranging from 06/01/1990 to 06/01/2017. First of all, rolling windows are employed in this paper to devide sequential data. Training data-set are from 16-quarters(4 years) to 40-quarters (10 years). Test data -set are from 1 quarter to 4 quarters. To build the model out of high dimensional space, top 20 most popular indicator is selected as Table 1 shows. Then, some of the stocks is deleted in order to reduce the ratio of missing data.

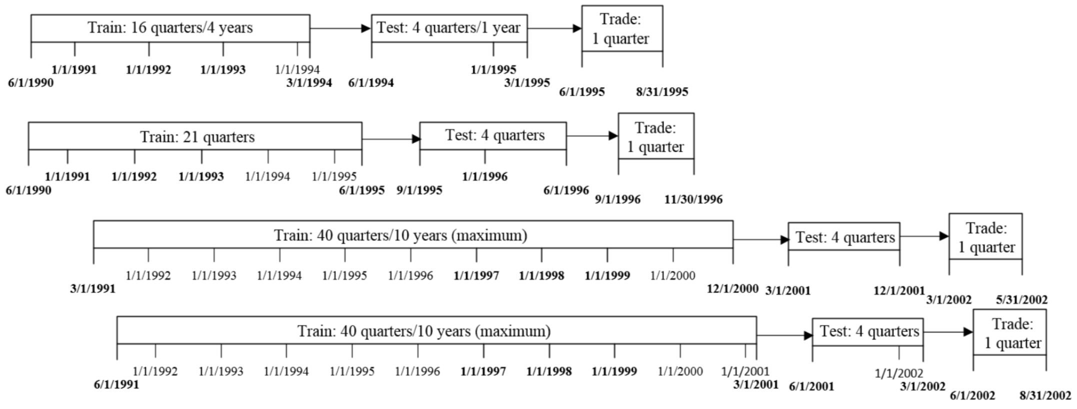

Figure 1: Rolling Window Based Data Seperation: Training rolling window is followed by a testing rolling window. There is a two-month delay after the end of the training rolling window.

2.2 Machine Learning for Stock Selection

In order to help later feature selection, five regression models are selected to approximate the reward given all these indicators. They are linear regression, stepwise regression under Akaike information criterion, ridge regression, random forest and boosted regression model. Standard R package are used to implement these models. Mean Square Error is used as metric to evaluate the model.

For the stock selection part, our goal is to select twenty percent stocks according to the predicted return of our model. In this paper, quarter log-return $r_(T + t, i)$ is used to evaluate the return of stock

i from time point T to time point $T+t$. Its relationship with stock price $S_{t,i}$, which identify the price of stock i at time point t, can be defined as:

$$r_(T+t,i) = ln(S_{T+t,i}/S_{T,i}), \tag{1}$$

3 Deep Reinforcement Learning for Stock Trading

3.1 Problem Formulation for Stock Trading

Portifolio allocation plays an important role in obtaining maximum return and balance investment risk at the same time. However, it can be done by solving Markov Decision Process (MDP). MDP provide a model for making decision in multi-stage stochastic environment. It is defined as a finite set of states $S = 1, ..., n$, a finite set of actions A_i for state $i \in S$, a transaction probability $p_{ij}^{a_i}$ and a return function $r(i, j, a_i)$ for $i, j \in S$ and $a_i \in A_i$. Furthermore, policy at each state is defined by $\pi(i)$, which deliver actions A_i for each state $i \in S$. Then, the given value function $V^\pi(i)$ of state $i \in S$ can be given as

$$V_\pi(i_0) = \mathbb{E}[\sum_{k=0}^{\infty} \gamma^k R(i_t, \pi(i_t))], \tag{2}$$

where γ is the discount factor with $0 \leq \gamma \leq 1$, and R is the expected return for $S = i_t$ and following $\pi(i_t)$. The goal is to find the best policy π^* that satisfies $V_i^* = max_\pi V_i^\pi$, which maximize V for every $i \in S$. In the scenario of this paper, the problem can also be simplified as Figure 2 shows.

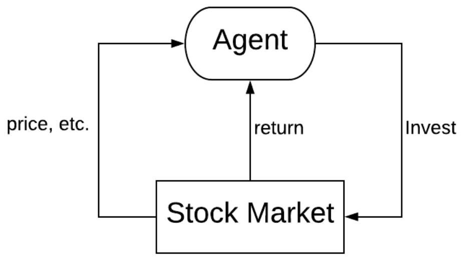

Figure 2: MDP model

In this graph, agent is the asset manager. The stock market is provide state information to agent, containing stocks price, rates and other indicators. Agent make decisions to allocate his asset by selling and buying stocks according to the state information. Then, stock market will return profit according to the invention agent made. According to the characteristics of MDP, stock market here will not be interfered by agent's decision. For a long term stock market, our problem can be represented as Figure 3 shows. Our sequence of states are consisted of stock markets at every time points locating along the time line. For example, three states can be select when time t equal to T, $T + t_1$ and $T + t_2$. Suppose an agent start from time T, all the data of market at that time is the state that agent is standing on. Then, the agent makes a invest action. For example, the agent can put 40% of total asset to a_1 and 60% to a_2. After making actions, agent waits until time $T + t_1$. At this time, reward R_{T+t_1} will be given to this agent according to the price of stocks. And agent need to make decision again. If the agent believe that slightly increasing the ratio of stock a_1 will give higher return, the ratio of each stock may change to 50% and 50%. Then, at time $T + t_2$, the environment will give another return according to the price of stocks hold by this agent. The agent stands at a now state and makes decisions based on new information of this state. After many time intervals, the agent get the final cumulative return.

3.2 Q-learning and optimization

In order to maximize the cumulative return, we need to maximize the return of taking each action at each state. We define the optimal action-value function $Q^*(s, a)$ as the maximum expected return achieved by acting action a at state s, $Q^*(s, a) = \max_\pi \mathbb{E}[R_t|s_t = s, a_t = a, \pi]$. We use Q learning algorithm to get the maximization, which obeys an important identity called *Bellman Equation*. The

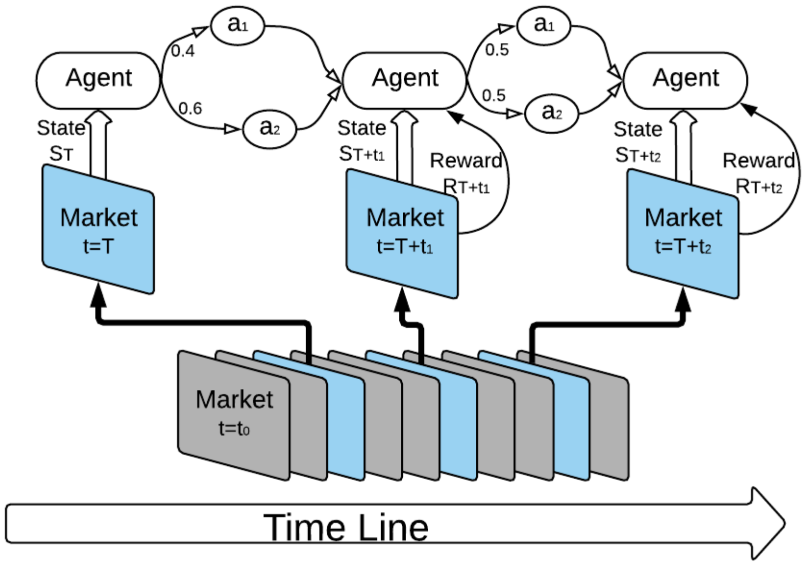

Figure 3: stock market as MDP

basic intuition is: if the optimum values $Q^*(s\prime, a\prime)$ of state $S\prime$ at next time-step are known for all possible actions $s\prime$ at this state, then the optimum value strategy is to select the action $a\prime$ that maximize the expected value of $r + \gamma Q^*(s\prime, a\prime)$, which comes from equation 2,

$$Q^*(s,a) = \mathbb{E}_{s\prime\ \epsilon}[r + \gamma \max_{a\prime} Q^*(s\prime, a\prime)|s,a], \tag{3}$$

Using bellman equation as iterative update, the action-value function at iteration $i+1$ can be represented as $Q^*(s,a) = \mathbb{E}[r + \gamma \max_{a\prime} Q_i(s\prime, a\prime)|s,a]$. This iteration can repeat for many times until it converges to optimum, which is $Q_i \to Q^*$ as $i \to Q$.

3.3 The Deep Reinforcement Learning Approach

In practice, the basic Q learning approach is not practical. Because the each state and action sequence is separately used to estimate the action-value function, without any generalization of function. The common solution is to use a function estimator $Q(s,a;\theta)$ to estimate the action-state function, $Q(s,a;\theta) \approx Q^*(s,a)$. To construct the estimator, liner and non-linear function are used. In this paper, we choose to use non-linear function approximator neural network. We define loss function of each iteration as $L_i(\theta_i)$. The network can be trained by minimizing the loss function value at each iteration i.

$$L_i(\theta_i) = \mathbb{E}_{s,a\ \rho(\cdot)}[(y - Q(s,a;Q_i))^2], \tag{4}$$

where $y = Q(s,a;\theta)$ is the target value for iteration i and $\rho(s,a)$ is the probility distribution over state s and action a.

In this paper, a technique known as experience replay is used. We store agent's experience at each time step $e_t = (s_t, a_t, r_t, s_{t+1})$ in data-set $\mathcal{D} = e_1, ..., e_N$. During the inner loop of update, we apply our deep Q network to samples of experience randomly drawn from data-set $\mathcal{D}$. There are two major advantages of doing this. One is more efficient use of previous experiences. The other is better convergence behavior when training the function approximator. Because data is more identical independent distributed. The full algorithm is presented in Algorithm 3.3

Algorithm 1 Deep Q-learning with Experience Replay

```
Initialize replay memory $\mathcal{D}$ to capacity $N$
Initialize action-value function $Q$ with random weights
for episode $= 1, M$ do
    Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
    for $t = 1, T$ do
        With probability $\epsilon$ select a random action $a_t$
        otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
        Execute action $a_t$ in emulator and observe reward $r_t$ and image $x_{t+1}$
        Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
        Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
        Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
        Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
        Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
    end for
end for
```

4 Solutions

5 Performance Analysis

6 Performance Evaluations

6.1 Experiment Setting

In our experiment, we ran our deep reinforcement learning algorithm on GPU to offer parallel computation support. Then, we compare returns at each time point with benchmark S&P 500 index, equal weighted portfolio, mean-variance portfolio and minimum-variance portfolio. Equal weighted portfolio only separate the asset equally. Mean-variance portfolio and minimum-portfolio mainly base on a fixed risk value to maximize return.

Three metrics are used to evaluate our result, annualized return, annualized standard error and Sharpe Ratio. The first metric indicate the direct return of the asset. The second one shows the robustness of our model. Sharpe Ratio combine return and risk together to give a value.

6.2 Results

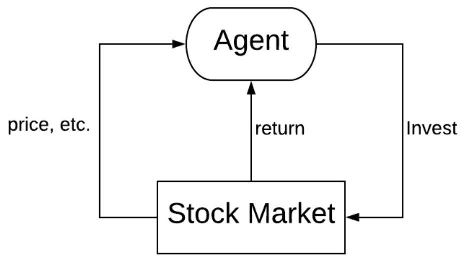

Figure 4: MDP model

[1] Alexander, J.A. & Mozer, M.C. (1995) Template-based algorithms for connectionist rule extraction. In G. Tesauro, D.S. Touretzky and T.K. Leen (eds.), *Advances in Neural Information Processing Systems 7*, pp. 609–616. Cambridge, MA: MIT Press.

[2] Bower, J.M. & Beeman, D. (1995) *The Book of GENESIS: Exploring Realistic Neural Models with the GEneral NEural SImulation System.* New York: TELOS/Springer–Verlag.

Table 2: In Sample result

Part		
Name	Description	Size (μm)
Dendrite	Input terminal	$\sim$100
Axon	Output terminal	$\sim$10
Soma	Cell body	up to 10^6

Table 3: Overall Sample result

Part		
Name	Description	Size (μm)
Dendrite	Input terminal	$\sim$100
Axon	Output terminal	$\sim$10
Soma	Cell body	up to 10^6

[3] Hasselmo, M.E., Schnell, E. & Barkai, E. (1995) Dynamics of learning and recall at excitatory recurrent synapses and cholinergic modulation in rat hippocampal region CA3. *Journal of Neuroscience* **15**(7):5249-5262.

[4] Markowitzz, H. (1952) Portfolio selection. *The journal of finance* 7(1):77-91.

[5] MathWorks, (2017) Matlab financial toolbox: Protfolio object,