MF 803 Finanl Project

Stock Portfolio Optimization with Deep Q-Learning

Jiachun Dai U64243821

Niyu Jia U97595078

Ziqi Nie U13274049

Yang Tao U40255851

Xuwei Wang U97177900

Introduction

Portfolio optimization is the process of selecting the best portfolio, out of the set of all portfolios being considered, according to some objective. The objective typically maximizes factors such as expected return, and minimizes costs like financial risk.

Common active methods for portfolio optimization contains Mean Variance Optimization, Minimize Conditional VaR, Minimize Maximum Drawdown and etc.

Generally, the active portfolio management always incorporates multiple constraints or objectives, which sometimes lead to a difficulty in finding the minima. (Such as minimizing VaR). Meanwhile, typical active assent management also requires datasets with multiple variables such as multi-factor model.

For the perspective of flexibility, if you have different objectives, you may need several separate models and different datasets to achieve the portfolio optimization. But with Q-learning, the process can become flexible.

Method Overview

For simplicity, our portfolio only contains two stocks, one is of high beta and another is of low beta. We consider that this kinds of stocks can actually represent a portfolio.

Assuming that our initial portfolio value is 100, and we will assign weights to two stocks and adjust the weights in the training periods. Please note that to do this algorithm, what we need is only the price data of two stocks which means there's no dimension problems.

Using deep Q learning, we let the algorithm choose every day weights automatically and compared the results with Do-Nothing portfolio. We want to figure out whether our algorithm performs at least better than passive investment with less constraints and data requirements.

Model Construction

The model is consist of three important parts:

Reinforcement Environment: how we construct the environment, the state, action, and reward. It also contains the possible dynamic change of the environment.

Deep Q-Learning Network: the training network to deal with the continues sates. Our network is based on tensorflow framework

Model Comparison: compare our model to the classic models, such as do-nothing portfolio, mean-variance portfolio and best sharpe ratio portfolio.

Data Set Information and Overview of The Strategy

We are using high beta stock and low beta stock price data. Time is from 1999.1.1 to 2019.10.1. Previous 70% of data is used for training and other is for testing.

We only put two stocks into our portfolio for simplification and we assume that low beta stock and high beta stock can actually represent a portfolio which has the same beta. Our strategy is to adjust portfolio weights in both stocks to earn higher return.

Environment Settings

We now consider about the necessary elements in Q learning.

Action space: We are using discrete action space which is given by:

$$action=[-0.15,-0.1,-0.05,0,+0.05,+0.1,+0.15,]$$

for each action, it means we are adjusting the previous low beta stock weights in the portfolio. For example, if the initial weights are $w_0=[0.5,0.5]$, after we take action $a=0.05$, the weights will become $w_1=[0.55,0.45]$

State: States are the input variables for our network. It is given by:

$$state_t=(w_{low},w_{high},Value,r_{low,1},r_{low,2},r_{high,1},r_{high,2})$$

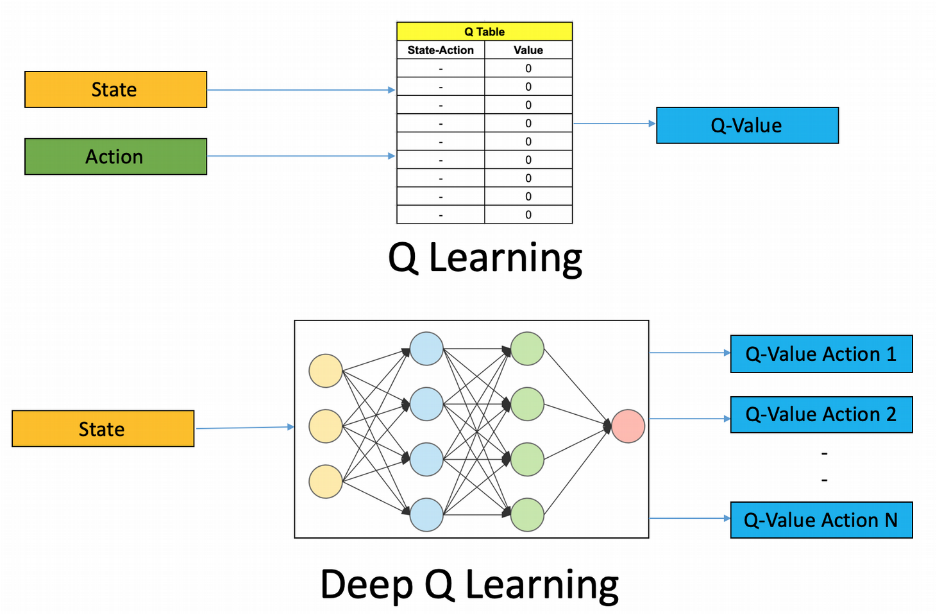
where w stands for the weights, Value is the total value of the portfolio, $r_{low,1}$ should be the return of low beta stock 1-day prior to present time t.

Reward: Reward defines how we let the network learns. There can be several reward type, such as portfolio value gain, sharp ratio, and portfolio return etc. Because it involves position change, transaction cost punishment is incorporated for any action that not equals to 0.

Termination: When and for what criteria we terminated the states will influence how the network performs. For simplification, we terminated the state every K days, standing for our investment cycle.

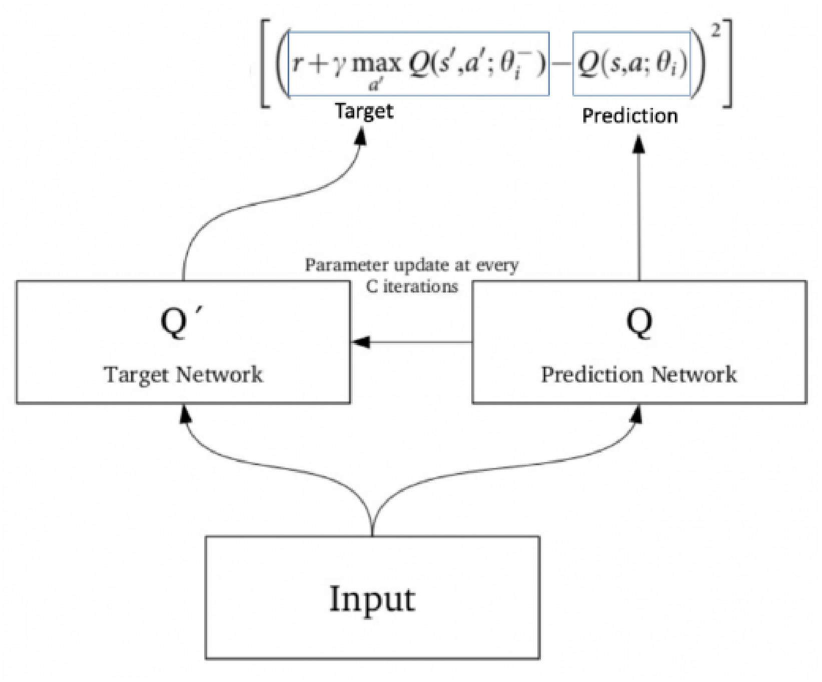
Deep Q-Learning

Since we have continues states, we are, of course, using deep Q learning network to fit our Q values by neural network. Deep Q network is developed with experience replay, target network and epsilon-greedy algorithm.



Algorithm training steps:

```
Initialize replay memory D to capacity N;Initialize Q network with random weights w;
Initialize target network with same weights w;
For e = 1 to total_episodes:
    Initialize state $s_1$
    For t = 1 to timesteps:
        With probability $\epsilon$ select a random action $a_t$ ,
        otherwise select $a_t = argmax_a Q(s_t, a, w)$
        Then execute $a_t$ , get reward $r_t$ and next state $s_{t+1}$
        Store tuple ( $s_t, a_t, r_t, s_{t+1}$ ) in memory D
        Sample a random minibatch form D, set:
        $y_j = \begin{cases} r_j & (if\ step\ j+1\ is\ \ termination\ \ state) \\ r_j + \gamma\, max_{a'} \hat{Q}(s_{j+1}, a_j, w) & (otherwise) \end{cases}$
        Then perform a gradient descent step on $(y_j - Q(s_j, a_j, w))^2$ with respect to w
        Update $\hat{Q} \leftarrow Q$ for every C steps
    End For
End For
```


$\left[\left(r+\gamma \max_{a'} Q(s',a';\theta_i^-) - Q(s,a;\theta_i)\right)^2\right]$
Target
Prediction
Parameter update at every
C iterations
Q′
Target Network
Q
Prediction Network
Input

Assuming that we are investing every 126 days (half year of trading days), then the environment should come with a termination every 126 days. (C=126) Besides, we are updating target Q network every 25 steps (about a month).

For the parameters, we choose $N=100, \gamma=0.9, \epsilon=0.1$. We replace the memory into a random index in the D in order to substitute some old memory therefore naturally give more weights to the recent experience. And we are using simple portfolio value increment as reward.

Deep Q Network Parameters

We use classic neural network to fit the Q values. The input tensor should have size of (1,7), the output size should be (7,1), the corresponding q-values of different actions(since we are having 7 actions).

The dropout rate: 0.5, which can prevent us from over-fitting

Dense layer units: 20 units for the first layer and 15 units for the second layer

Batch size: 8, the batch size for experience replay

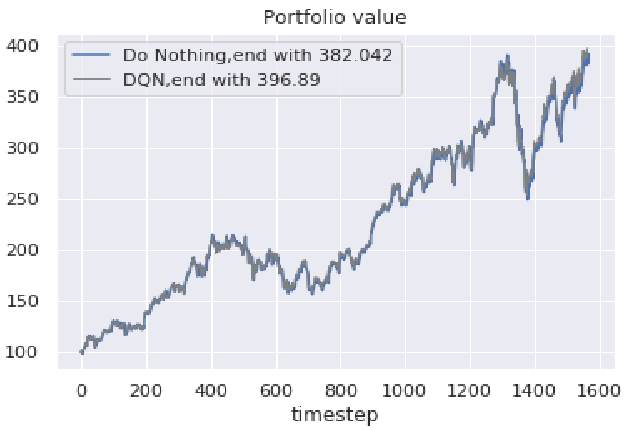
Learning rate: 0.01, the learning rate in back-propogation

Results

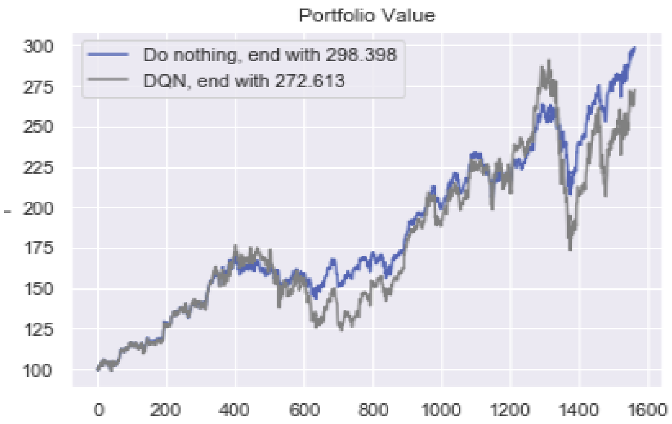
Comparison with Other Strategies

(a) For AEP and AAPL portfolio combination, using value as reward

(b) For AEP and AAPL portfolio combination, using sharpe ratio as reward:



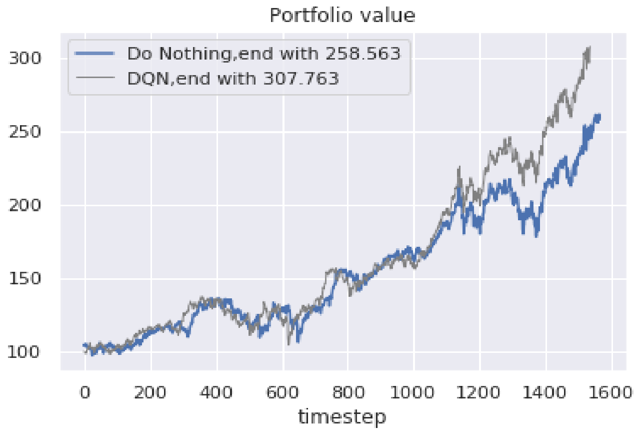
(a)



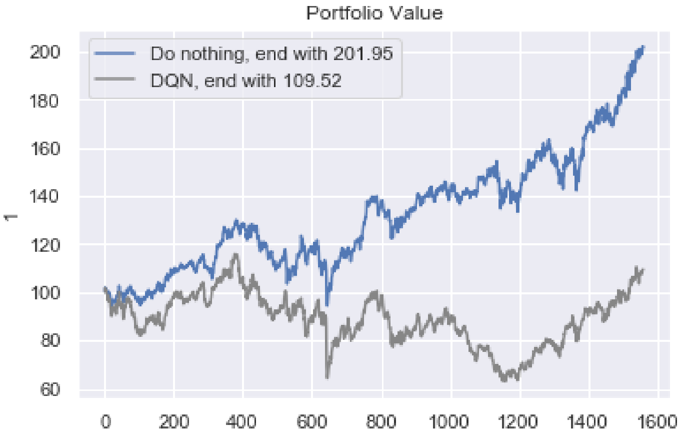
(b)

(c) For HCP and TXN portfolio combination, using value as reward

(d) For HCP and TXN portfolio combination, using sharpe ratio as reward



(c)



(d)

Further Consideration

Advantages and disadvantages

Advantages:

- It is an off-policy learning, which is easy to explain and understand
- Deep Q-Learning solves the problem of continues sates and avoids extreme large Q table
- With small batch size and experience replay, the model can be trained efficiently
- Different reward types can result in different dynamic strategies without re-construct model

Disadvantages:

- The results can be really unstable since there are too many tuning parameters. And the results would be sensitive to the natural features of stocks you choose.
- Demanding computation speed and capacity are needed.
- We need enough big datasets to explore enough strategies.

Methods for improvement

Sometimes this method would totally fail in the circumstance we mentioned above. And also the usage of Deep Q learning can lead to overestimation. Taking the maximum overestimated values as such is implicitly taking the estimate of maximum value. A maximization bias in learning will be triggered by this systematic overestimation. And since Q learning involves the bootstrapping, which is learning estimates from estimates, the overestimation will be problematic in the end.

In order to solve this, we can try the Double Q learning, involving two separate Q-value estimators, each of which is used to update each other. So we can unbiased Q estimation of the actions selected using the opposite estimator.

Conclusion

Deep Q-Learning can be used on portfolio maximization and the network give us a more precise profile for our outcome of position adjustment. Due to the time limit and computer combination capacity, we only chose several portfolio combination for detailed analysis.

In this project, we managed a two-stock portfolio with the goal of maximizing portfolio value while minimizing risk. By investigating various reward functions and hyperparameters, we successfully implemented an algorithm which performed on-par if not better than preset performance benchmarks, according to the different metrics.