National Technical University Of Athens

Stock Trading
A deep Q-learning approach

Constantinos Tsaousis

Constantinos Costopoulos

Stylianos Tzelepis

17. mars 2022

1 Introduction

The role of the stock market across the overall financial market is indispensable. The way to acquire practical trading signals in the transaction process to maximize the benefits is a problem that has been studied for a long time. Profitable automated stock trading strategy is vital to investment companies and hedge funds. It is applied to optimize capital allocation and maximize investment performance, such as expected return. Return maximization can be based on the estimates of potential return and risk. However, it is challenging to design a profitable strategy in a complex and dynamic stock market. Every player wants a winning strategy. Needless to say, a profitable strategy in such a complex and dynamic stock market is not easy to design. For the purpose of this project, we are making use of a deep reinforcement learning approach, more specifically Q learning, that automatically learns the optimal trading strategy by maximizing investment return.

2 Q-Learning and Q-Learning in Trading

Q-learning is a model-free reinforcement learning algorithm. It is used for finding the best policy $\pi(s, a)$ such that $\pi(s, a) = argmax_a Q(s, a)$, where Q(s,a) is called the Q-function, which quantifies the joint quality of taking the action $\boldsymbol{a}$ given a current state $\boldsymbol{s}$. Q-learning is an off-policy learner; it learns the value of the optimal policy independently of the agent's actions. The core of the algorithm is a Bellman equation as a simple value iteration update utilizing Temporal Differences (TD) to estimate the value of the optimal Q(s,a).

Stock trading contains short-term and long-term acknowledgements concerning the trader's actions (buy a stock, sell a stock, maintain state) as rewards. As a result, Reinforcement Learning seems a perfect fit for solving optimization problems in trading since the goal of an agent is to maximize it's rewards.

2.1 Basic Characteristics.

There are some characteristics we need to notice during designing methodology in the field of finance.

- The agent interacts with the financial market at discrete time steps even though the time steps may be extremely close, say, in the high frequency trading, trading decisions can be made in the matter of milliseconds.
- There is a set of legal actions an agent can apply to the market from naively submitting market orders with a fixed position size to submitting a fully specified limit order. And the trading size and historical position directly impact the cost of a single trading action. The financial market produces new information available to the agent at each time step enables the agent to make trading decisions. However, the agent has no idea about the future trends of the market price.
- The agent has the potential to alter, although without full control over, the financial market if it is powerful enough. Hence it is not entirely realistic to consider the market to be fully exogenous to the agent.

2.2 Approach.

In the real world, our ultimate goal would be to create an agent that can successfully make use of a large pool of stocks in order to maximise the trading profits. Such problem would require a lot of effort to begin with. Thus we opt to start with a sub-problem of the above, one that the agent has a small number of stocks (1, 2 and 3) to capitalise in order to maximise the profit. Additionally, we will ignore the transaction cost when buying and selling stocks.

Our approach to solve the trading optimization problem consists of three parts;

A) A Markov decision process (MDP) model is proposed for general signal-based financial trading task solvable by state-of-the-art deep reinforcement learning algorithm with publicly accessible data only.

B) Modification of the deep recurrent Q-network algorithm (DRQN). This involves using a substantially smaller replay memory and sampling a longer sequence for training, discover workable hyperparameters for the DRQN algorithm able to solve the financial trading MDP through random search and development a novel action augmentation technique to mitigate the need for random exploration in the financial trading environment.

C) Experimentation with different currency pairs to work on real financial data.

We trained via deep Q-learning the agent to capitalise by buying, selling and holding from a **n** number of stocks where **n** =1,2,3 .Let's elaborate more on the agent that chooses from **2** stocks :

- The agent starts with initial capital of X.
- There are 2 profits that the agent can trade from, whose price is defined as P_A and P_B

3 Data Analysis

3.1 Dataset

The data that we are going to use is gathered from this dataset of Kaggle for more than 15 years :

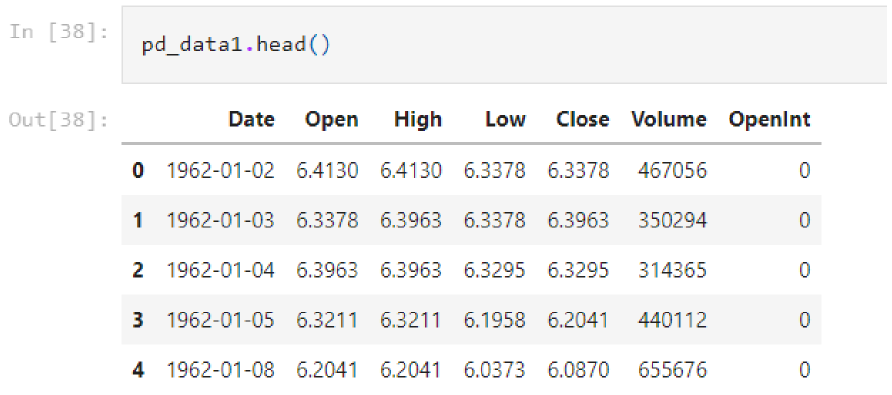
https://www.kaggle.com/borismarjanovic/price-volume-data-for-all-us-stocks-etfs

The CSV has the following format:

3.2 Pre-processing

First step in data pre-processing was to convert "Date" column to date format. Next step was to make sure that data for both stocks agree on the dates. It is possible that there are many days where either one of their data is not present. Where one's data is not present for a given day, we had 2 options:

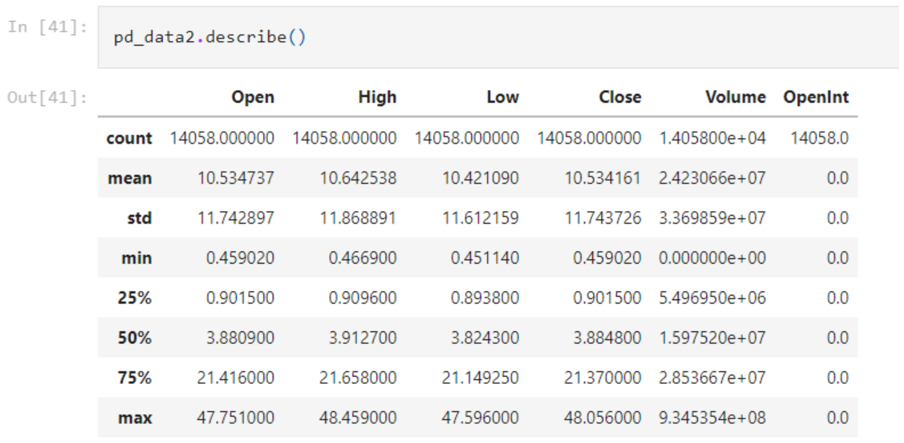
- Populate the data from the previous or the next day
- Delete the record from other stock.



```
In [38]: pd_data1.head()
```

Out[38]:

	Date	Open	High	Low	Close	Volume	OpenInt
0	1962-01-02	6.4130	6.4130	6.3378	6.3378	467056	0
1	1962-01-03	6.3378	6.3963	6.3378	6.3963	350294	0
2	1962-01-04	6.3963	6.3963	6.3295	6.3295	314365	0
3	1962-01-05	6.3211	6.3211	6.1958	6.2041	440112	0
4	1962-01-08	6.2041	6.2041	6.0373	6.0870	655676	0



```
In [41]: pd_data2.describe()
```

Out[41]:

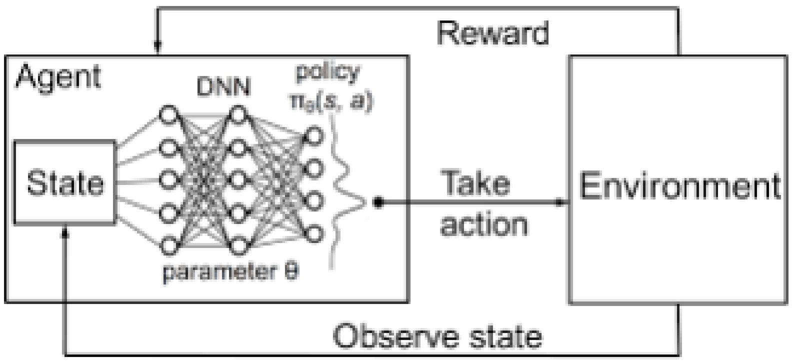
	Open	High	Low	Close	Volume	OpenInt
count	14058.000000	14058.000000	14058.000000	14058.000000	1.405800e+04	14058.0
mean	10.534737	10.642538	10.421090	10.534161	2.423066e+07	0.0
std	11.742897	11.868891	11.612159	11.743726	3.369859e+07	0.0
min	0.459020	0.466900	0.451140	0.459020	0.000000e+00	0.0
25%	0.901500	0.909600	0.893800	0.901500	5.496950e+06	0.0
50%	3.880900	3.912700	3.824300	3.884800	1.597520e+07	0.0
75%	21.416000	21.658000	21.149250	21.370000	2.853667e+07	0.0
max	47.751000	48.459000	47.596000	48.056000	9.345354e+08	0.0

We chose option 2, deleting the record from other stock for missing stock date. Another useful insight is that the Volume and Open Interest parameters were not providing any useful information and therefore not contributing to the training of the model. Therefore we will ignore them for the state.

4 Implementation

4.1 Deep Q Network

Neural networks are exceptional in coming up with good features for highly structured data. We could represent our Q-function with a neural network, that takes the state and action as input and outputs the corresponding Q-value. Alternatively, we could take only state as input and output the Q-value for each possible action. This approach has the advantage, that if we want to perform a Q-value update or pick the action with highest Q-value, we only have to do one forward pass through the network and have all Q-values for all actions immediately available.



4.2 State

Each state is described by the following parameters:

- Date
- Open
- High
- Low
- Close
- Open-Cash, available for agent to buy stocks.
- Portfolio, value at any point of time.

The Volume and Open-Interest variables were not useful for learning.

4.3 Actions

The agent can perform the following actions:

- Do not buy or sell from any stock
- Buy from stock A
- Buy from stock B
- Sell from stock A
- Sell from stock B

4.4 Experience replay

Our problem is that we have sequential samples from interactions of the environment with our neural network. Therefore, it forgets previous experiences and swaps them with new ones. For instance, if we are on Day 400, where stock is going up continuously for last 30 days, it will forget the experience from day 250 to day 300 where stock had a very low valuation because of poor financial performance. The Correlation Problem arises: If our agent is learning only from experiences of the past few days, it will evaluate the situations from a short term perspective, thus immediately selling as soon as a stock price goes down. The Optimal Action to take at any state can be found by utilizing deep Q-network.

4.5 Benchmark

Benchmark model is a model where there is no agent, no reinforcement learning. The idea is that we would like to compare the agent with a simple benchmark where there is no intelligence, buying the stocks at the start of period and sell it periodically over the whole period, with 10% sell every time. It is *hard coded*.

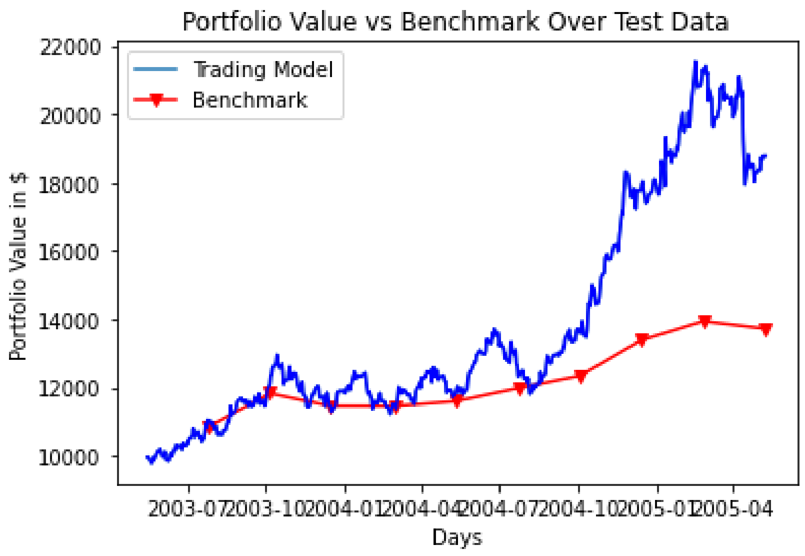
Thus, benchmark is a model where:

- At the start of each period: We buy stocks, with half amount invested in each stock. Open Cash remains the same through-out the period.
- Sell 10% of stocks in 10 intervals, thus increasing open cash.
- At the end of each period calculate portfolio value as follows:
  Portfolio Value = (Quantity of Stock1 * Price of Stock1) + (Quantity of Stock2 * Price of Stock2) + Open Cash

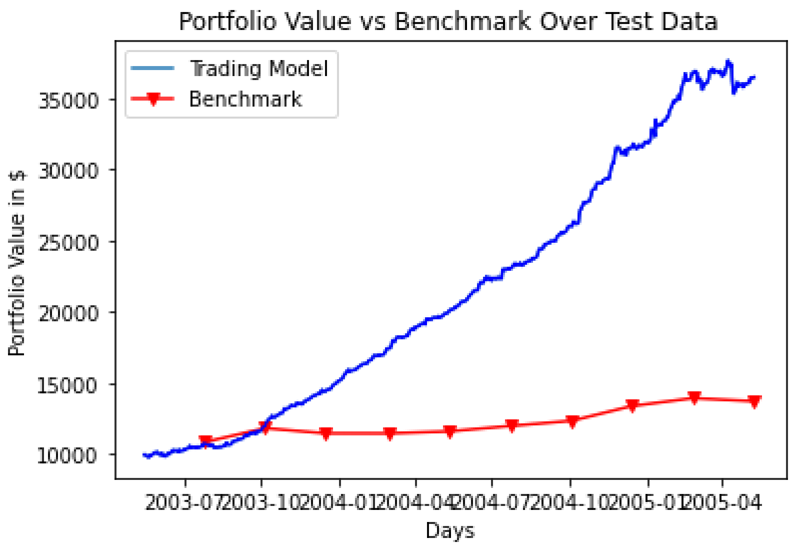
5 Results

Comparing the performance of the agent with the benchmark portfolio we observe that the agent gained much more profit than the benchmark model, both on the train and the test run. The model was trained on a part of the Apple-Amazon stock dataset and tested on the rest of it with and without the use of experience replay. As expected the use of experience replay significantly improved the convergence and the time needed to reach it.

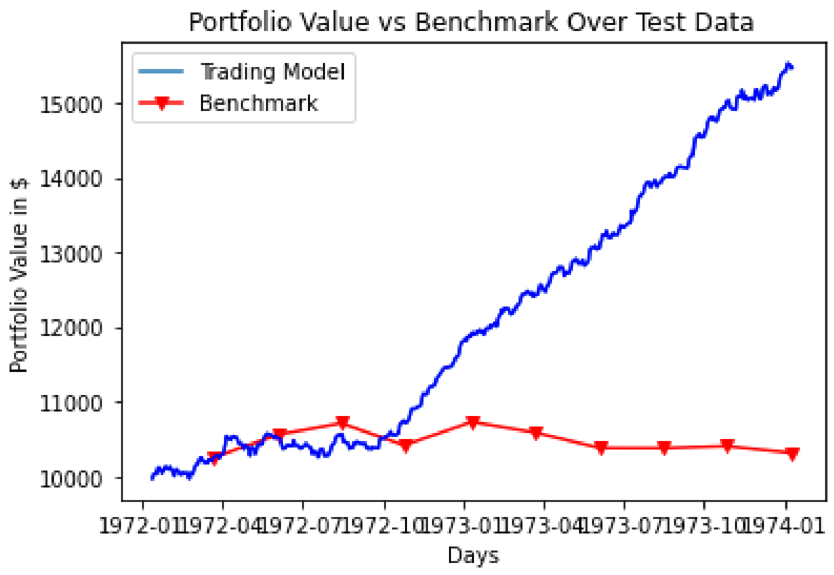
- Apple - Amazon **Without experience replay**



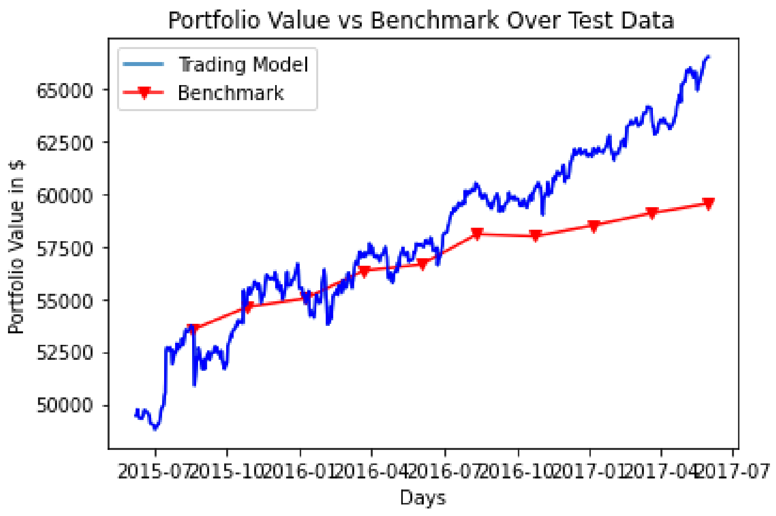
- Apple - Amazon **With experience replay**



- IBM - GE. More episodes and more days per episode



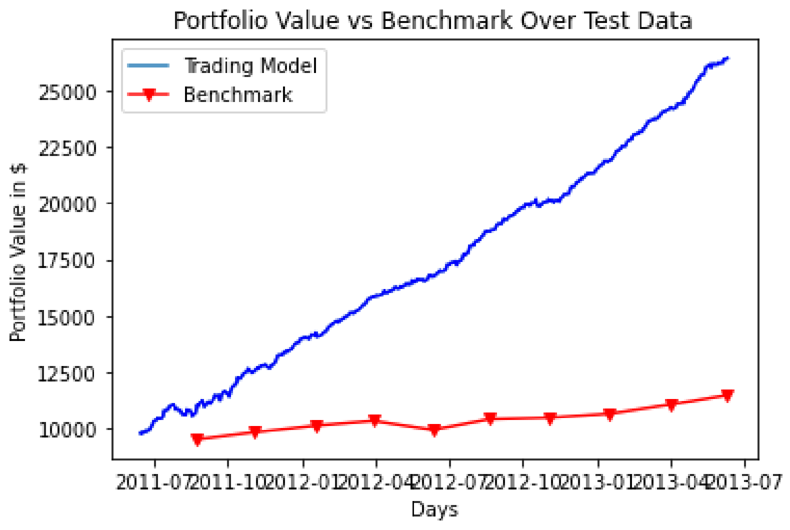
- Google - Walmart. Starts from 50.000



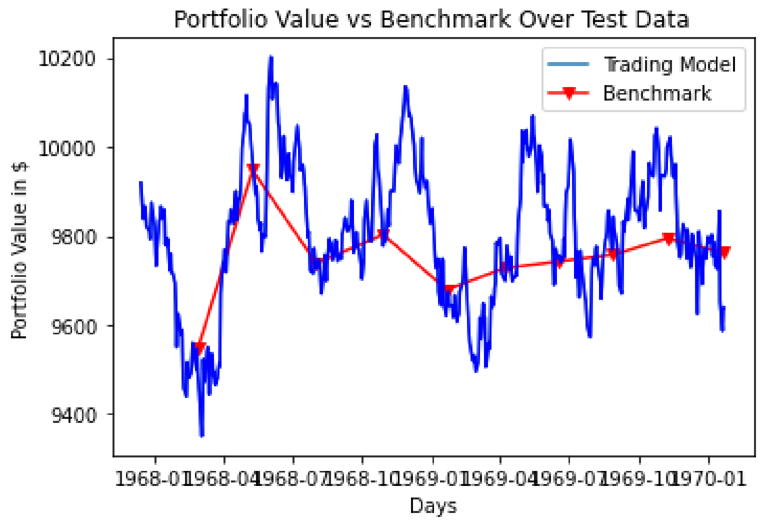
Additionally, as a final insight we tested our agent in unseen data of **different** stocks than the one that is trained on. The convergence of this, further proves the robustness of the agent, which is now surpassing the profit of the benchmark model in not only data but also patterns not seen before.

Note: The code of the project can be found **here**.

- Google - Walmart. Converges almost immediately.



- IBM - GE. Does not converge with the 12 episodes on which it was trained. This is expected, since the IBM - GE pair was trained in over 20 epochs. Moreover, the starting budget on which we trained the agent is 10,000$, while the corresponding one of the IBM - GE pair is 50,000$.



6 Challenges

We identified seven major challenges to applying reinforcement learning to financial trading:

- Impossible for exploration policies to be used in trading, as opposed to the majority of deep reinforcement learning algorithms.

- Input selection. As we were trading with two stocks simultaneously, we had to make sure that the two respective companies are in the same industry, thus same factors determine their price changes. We initially trained our model with unrelated stocks, therefore the agent produced rather concerning results.

- Defining States: Since we are facing a Markov Decision Process (MDP) problem, the next state solely relies on the current one. However, stock prices change on a daily basis over small time intervals. As a result, past stock prices influence current and future ones and we have to include this conclusion in our State model. After some trial and error, a 5-day span was chosen for the price shifts.
- Defining Actions: Trading 2 stocks provides the agent with numerous options. We chose to provide it with the 5 actions mentioned before, although we tried other alternatives too. Buy Stock1 sell Stock2 or buy Stock2 sell Stock1 to name a few, but the agent's profit did not improve significantly.
- Defining Rewards: A lot of fine tuning took place at this point. We faced scenarios where the agent had less open cash than the half of his initial balance and it went bankrupt rather soon, as it tried to buy stocks when prices dropped. We then modified the rewards system to punish the agent if it buys stocks when low on money or when it sells when the stock balance is not high.
- Number of stocks. Trading with 1 stock was trivial so we skipped it, while training with 3 stocks turned out to require multiple days of training, which significantly slowed down the fine tuning process. Moreover, the convergence relied on many more factors, thus further holding down the accuracy of the model. The golden ratio was 2 stock simultaneous trading.
- Different datasets and tweaks needed. We gathered data from various companies and focused on the stocks of 6. Initially, we created 3 dyads, trained and tested our model in each one of them. We noticed that some converged when the initial balance was significantly higher, while others when we increased the number of episodes, as well as the days of the training. As a result, we had to narrow down the parameters most important to the training process and tweak all of them to an all-round optimal one. Eventually, we trained our agent in such a stock pair and tested its results in the other two pairs; the first one saw great success, while the second one - the one that needed a higher initial budget - did not provide any important profits.

7 Improvements

Some improvements that can be done in future work are :

- Use of double Deep Q-Network might lead to better performance
- As mentioned in [1] as well as other publications, a possible improvement would be to include an LSTM keeping the memory of a number of states before the current i.e the last 30 days. This could perhaps greatly improve the approximation of the Q value but increase the computational complexity

8 References

1. Financial Trading as a Game: A Deep Reinforcement Learning Approach: Huang, Chien-Yi https://arxiv.org/pdf/1807.02787.

2. Deep Reinforcement Learning in Quantitative Algorithmic Trading: A Review https://arxiv.org/abs/2106.00123

3. Application of Deep Reinforcement Learning in Stock Trading Strategies and Stock Forecasting by Yuming Li, Pin Ni and Victor Chang https://core.ac.uk/download/pdf/322327703.pdf

4. Deep Reinforcement Learning for Automated Stock Trading https://towardsdatascience.com/deep-reinforcement-learning-for-automated-stock-trading-f1dad0126a02

5. MACHINE LEARNING FOR TRADING: GORDON RITTER: https://cims.nyu.edu/ ritter/ritter2017machine.pdf