Deep Q-Learning for Stock Trading using Transformers

Andrei Neagu

Abstract

The goal of this project was to see if it is possible to implement a DQN model using Transformers that can profitably trade a pair of stocks. Multiple different methods were applied to the DQN agent to stabilize it as vanilla Q-Learning is notoriously unstable due to it overestimating action values.

1 Introduction

An ideal stock trading strategy would be able to "outperform" the market, meaning it would get returns that exceed the general returns of the stock market while having the same risk. In this paper, we explore the idea of using Deep Q-Learning, with Transformers as the function approximator, to get a profitable Q-Learning agent. The idea of using reinforcement learning for stock trading is not new. However, most recent papers are using LSTMs as a function approximator [1]. Additionally, due to computational constraints, we only evaluate a portfolio consisting of two stocks: Apple and Microsoft, while other papers explore up to 30 stocks [2].

2 Background

2.1 Transformers

As mentioned previously, most papers on the topic of deep reinforcement learning for stock trading employ long short-term memory networks as their function approximators. Long short-term memory are a type of recurrent neural networks that are very good at processing sequences of information which makes them ideal for processing stock market data. However, they are known to perform weakly in processing long sequences of data. Transformers are another type of neural networks that can also process sequences of data. However, they use self-attention mechanisms that allow them to focus on important elements. Unlike recurrent neural networks, this allows them to capture long-range dependencies and outperform traditional RNNs like LSTMs. They are typically used in natural language processing applications and are the foundation of big language models such as ChatGPT [3].

2.2 Q-Learning

The Transformers are going to be used as a function approximator for the Q-learning agent, which is a reinforcement learning algorithm that learns the optimal action-values at each state by estimating the expected long-term reward of taking an action in a state. However, Q-learning requires some improvements due to its issues of variability and also because of the maximization bias due to picking the action which has the highest value at each state as the target [1].
Some of the improvements we have seen in class such as double Q-learning and using replay buffers. Double Q-learning splits the function approximation into two networks, network 1 is used to compute the target for network 2 and vice-versa. Replay buffers store previously seen transitions of state, action, reward and next state and are reused in a batch to reduce variability and that way we have

independent and identically distributed data which is necessary for SGD to work [1][4].
Two new methods we have not seen in class are fixed Q-targets and dueling DQN. fixed Q-targets freeze the target function approximators for a certain number of update steps to allow the unfrozen Q-functions to converge to the targets [1][4]. The next idea is dueling DQNs, which consists of splitting the Q-value into state values and the advantage of each action [1][5]. This allows the agent to learn the state value function and the value of picking a action in a state over all over actions. This has been shown to stabilize training [1].

3 Methodology

3.1 Environment

The environment is a custom stock trading environment that I coded in Python. Its purpose is to allow the user to trade a pair of stocks over a certain predetermined period on historical data of two stocks and two indices.

3.2 State Space

The two stocks chosen in our case are Apple and Microsoft because they are both traded on the NASDAQ stock exchange and since they are both technology stocks, we can assume that they are highly correlated. This way we have additional information about them. The two indices are the NASDAQ index, which tracks the performance of all stocks listed on the NASDAQ, and the VIX index, which is an index that tracks the volatility of the market. The observations consist of each day from January 1st 1990 to January 1st 2023. Each day has a open price, high price, low price, close price, adjusted close and volume. The adjusted close will be used as the price of the stock to calculate gains because it corrects for irrelevant market adjustments that might otherwise skew the results.
Additionally, the holdings at each state are used and are represented by the price of stock when bought or shorted. the price is positive for a long position (buy) and negative for a short position (sell). Shorting consists of betting that the stock will go down.

3.3 Action Space

The action space consists of buying (1), selling (-1) or holding the stock (0). For simplicity, we can only hold one stock at a time so that if we hold a stock, we cannot buy another one. So the action space consists of $3^2 = 9$ actions since we have 2 stocks and 3 possible actions. The action is indexed from 0-8 and an invalid action mask is also applied based on the current holdings.

3.4 Data PreProcessing

The data is then split into a 70-30 train-test split. The test set is stored in the data set called val-test and then it is further split into a 70-30 test-validation split. The data set is then normalized between 0.1 and 1.1 and not [0, 1] because the first observations consisting of all 0s might cause problems with the neural networks. The data sets are then transformed into sequences of length 64.

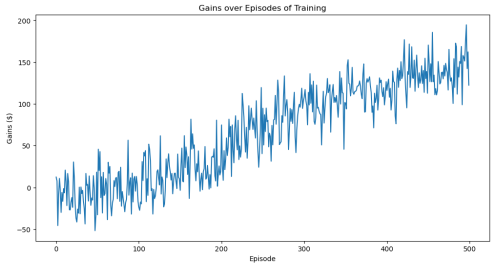
3.5 Transformer Network

The final Transformer architecture used, based on hyper-parameter tuning on the validation set, starts with a linear layer to which positional encodings are added and passed to two layers of a Transformer Encoder with 4 attention heads. The embeddings are of size 64 and a dropout value of 0.2 is used. That output is then passed to a final MLP layer where dueling DQN is implemented and outputs values for all 9 possible actions.

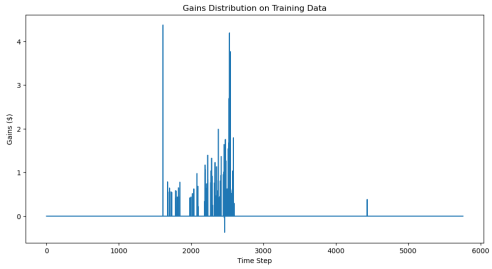
3.6 DQN Agent

Double Dueling DQN is implemented with a replay buffer of size 10,000 and a batch size of 128. The Target Networks are updated every 1000 steps. Exploration is ϵ-greedy where ϵ=0.1.

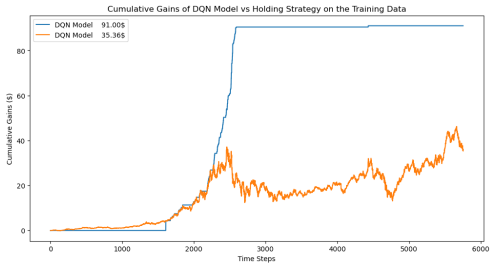
4 Results



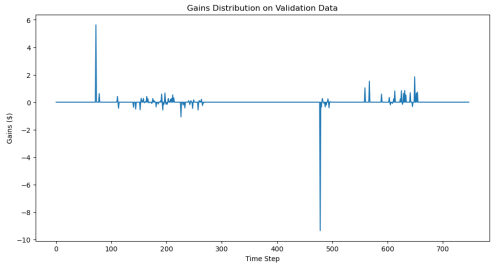
(a) Training Curve



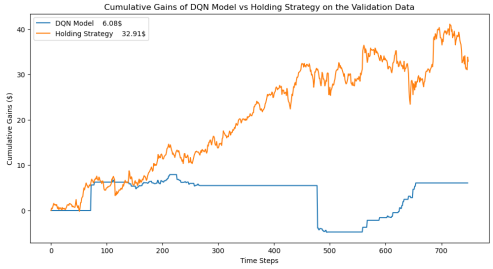
(b) Gains Distribution on Training Data



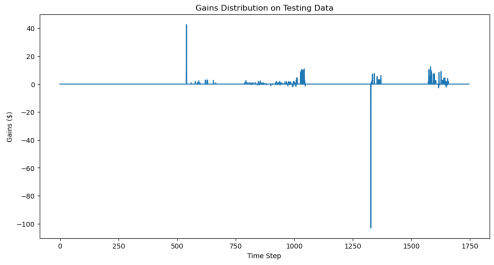
(c) Cumulative Gains Distribution on Training Data



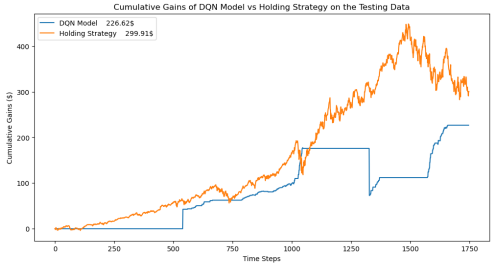
(d) Gains Distribution on Validation Data



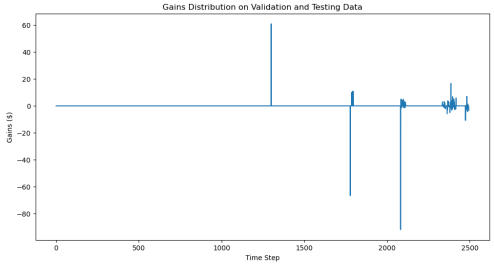
(e) Cumulative Gains Distribution on Validation Data



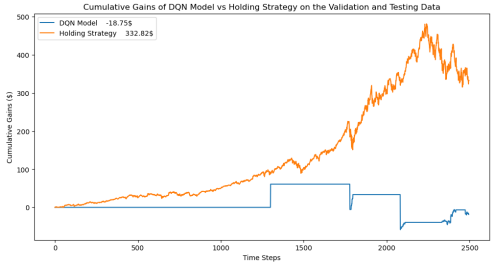
(f) Gains Distribution on Testing Data



(g) Cumulative Gains Distribution on Testing Data



(h) Gains Distribution on Validation + Testing Data



(i) Cumulative Gains Distribution on Validation + Testing Data

Figure 1: Experimental Results

We can see from the training curve in Figure 1 (a) that the DQN agent seems to be learning on the training set since the gains are increasing after each of the 500 episodes. We can also see from the gains distribution graph in (b) that it has way more gains than losses. In (c) we see that the agent performs very well when compared with the baseline strategy of just holding the pair of stocks. In fact it ended with a gain of 91$ versus 35.36$.
This was of course the best performing agent that I got based on the validation set performance. However, we can see that there is a big gap between the training set performance and the validation set performance. In (e), we can see that on the validation set, the model managed to get a dollar gain of only 6.08$ as opposed to 32.91$ obtained simply by holding the pair of stocks. In many cases while training, the model actually lost money trading on the validation set. This indicates that the Transformer network used is very prone to over fitting to the training data.
On the test set, the performance was decent and the gains of 226.62$ are close to the gains obtained of 299.91$ by holding the pair of stocks. In both the validation set and the training set, we can see that a few big losses wipe out many of the small incremental gains that the model achieves. This is apparent in graphs (d) and (f).
I have also evaluated the performance on a combination of the validation and the test set and we can see that the model actually lost money on that set. It lost about 18.75$ instead of getting a gain of 332.82$ just by holding the pair of stocks. This clearly shows that the model is over fitting to the data.
Over fitting was a big issue while training and while doing hyper-parameter tuning. This was the best result I got after decreasing the number of layers and decreasing the embedding size in the Transformer and yet it is still unstable. In the last few tests, I tried increasing the dropout rate, but even increasing it from 0.2 to 0.25 made the agent unable to learn and gave a flat learning curve. The solution to this over fitting problem isn't clear since there are so many hyper-parameters to tune. This is one of the drawback of using Transformers as the Q-function approximator. In addition to having the Q-learning hyper-parameters such as the epsilon of the ϵ-greedy exploration, the gamma, and the number of episodes to train on, we also have the hyper-parameters of the Transformer to optimize for such as the number of heads, the embedding size, the number of layers, the sequence length, the dropout rate, the learning rate, etc.
Additionally, the number of combination of hyper-parameters were limited by the computational cost of each training loop which would take between 6 and 12 hours to each run for 500 episodes on a 20GB GPU.

5 Conclusion

As can be seen from my experiments, the results were mixed and the model was not able to be consistently profitable when trading a pair of stocks. However, the DQN algorithm implemented shows that it is able to learn so with perhaps more time and computational power for hyper-parameter tuning I could find a model that could consistently be profitable or even outperform the baseline holding strategy. It could also be interesting to implement different Reinforcement Learning strategies. In many of the papers I have read, the authors use different algorithms to compare with each other such as Deep Deterministic Policy Gradient, which uses both Q-learning and policy gradient, and Proximal Policy Optimization, which is an actor-critic algorithm that aims to have updates that don't swing the new policy too far away from the previous policy [6]. Most papers also introduce commission fees into the reward function such that each trade costs a fraction of the price of the transacted stock [2]. Commission costs were added to the model but since the results of learning without these costs were mixed, I decided to run the experiments without them. However, the code to allow for commission fees is still implemented and ready to be used. It would be interesting to play with different commission percentages, after finding a model that is consistently profitable when commission costs are not taken into account, to see how they affect performance.

References

[1] Z. Zhang, S. Zohren, and S. Roberts, "Deep reinforcement learning for trading," arXiv.org, 22-Nov-2019. [Online]. Available: https://arxiv.org/abs/1911.10107. [Accessed: 23-Apr-2023].

[2] H. Yang, X.-Y. Liu, S. Zhong, and A. Walid, "Deep Reinforcement Learning for Automated Stock Trading: An ensemble strategy," SSRN Electronic Journal, 2020.

[3] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," arXiv.org, 06-Dec-2017. [Online]. Available: https://arxiv.org/abs/1706.03762. [Accessed: 23-Apr-2023].

[4] J. TORRES.AI, "Deep Q-Network (DQN)-II," Medium, 10-May-2021. [Online]. Available: https://towardsdatascience.com/deep-q-network-dqn-ii-b6bf911b6b2c. [Accessed: 23-Apr-2023].

[5] C. Yoon, "Dueling Deep Q Networks," Medium, 20-Oct-2019. [Online]. Available: https://towardsdatascience.com/dueling-deep-q-networks-81ffab672751. [Accessed: 23-Apr-2023].

[6] X.-Y. Liu, Z. Xiong, S. Zhong, H. Yang, and A. Walid, "Practical deep reinforcement learning approach for stock trading," arXiv.org, 30-Jul-2022. [Online]. Available: https://arxiv.org/abs/1811.07522. [Accessed: 23-Apr-2023].