Stock Trading with Reinforcement Learning

Team Members (Group 23):
Israel Diego, (israeldi@umich.edu)
Huiyang Ding (huiyangd@umich.edu),
Shuhan Wei (wshuhan@umich.edu),
Zhouyuan Zhang(hunterzz@umich.edu)

March 2020

Abstract

In this paper, we use Q-learning, which is a reinforce learning algorithm to make trading decisions on the U.S stock market. We find a Q-function that is profitable on training and testing data. Moreover, our model traded sensibly during the corona virus market meltdown. Our work is a starting point for more sophisticated implementations of Q-learning algorithms.

1 Introduction

Machine learning has grown in popularity for predicting the stock market [1][4][5] . One interesting branch of machine learning is reinforcement learning, which attempts to learn optimal decision-making to achieve maximum reward in a complex dynamic environment. In this project, we leverage the application of reinforcement learning to trade in the U.S stock market. Specifically, we use Q-learning which chooses the optimal action at each time step, given the current state and environment. The Q represents quality, which means the usefulness of a given action/decision from gaining reward. We will train our algorithm to choose from a set of actions and execute buy/sell trades when the model decides it is optimal to do so. We will also train the model to make a decision on the size of a trade at each specific time. Q-learning finds a policy that maximizes the expected value of the total reward over any and all successive steps, starting from the current state. More specifically our algorithm will aim to maximize the following utility function (Equation 1):

$$E\left[U\left(w_T\right)\right] = E\left[w_T\right] - \frac{1}{2}\kappa V\left[w_T\right] \approx \sum_{t=1}^{T} R_t \tag{1}$$

where w_T is an investor's wealth at some future time T, κ is a constant that depends on the initial wealth w_0 and R_t is the reward at time time $t < T$. Maximizing the utility function above is equivalent to maximizing the mean-variance of an investor's wealth at some future date T and this is approximately equal to maximizing cumulative reward over time [1].

2 Data

The data is obtained from Yahoo Finance[1]. We will focus on the pool of stocks from S&P 500 and use daily adjusted closing price data from January-01-2015 to April-13-2020 in order to calculate returns. Particularly we will focus on stocks from various industries, such as Alphabet Inc.(GOOG), Apple(AAPL), Nvidia(NVDA), Markel Corporation(MKL) and General Motors Company(GM). In Figure 1, we notice that Google stock performance was increasing overall, but experienced large losses during the corona virus outbreak.

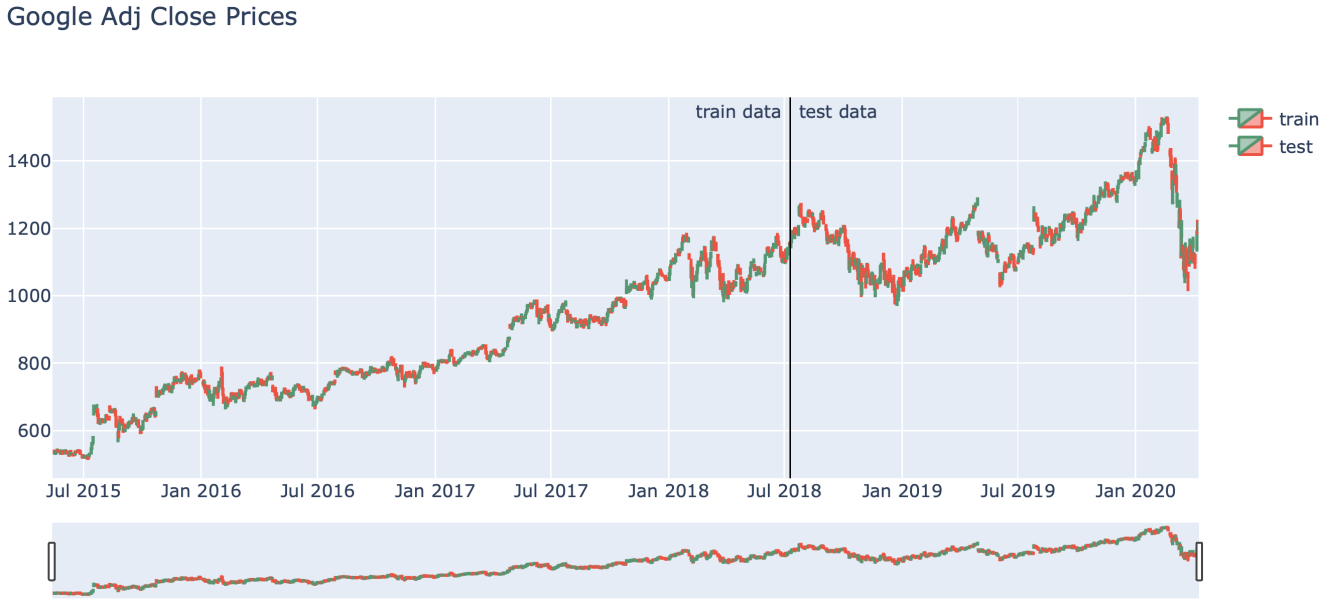

Figure 1: Google Stock Time Series from July 2015 to Jan 2020. Green is for increasing and red is for decreasing.

We also perform some exploratory data analyses over all the 5 target stocks. The key finding is that we need to train our model on daily returns rather than prices. The daily return is calculated as,

$$R_t = \frac{S_t}{S_{t-1}} - 1 \tag{2}$$

where t denotes t^{th} day in a market, S_t is the adjusted close price, and R_t is the return.

Returns are preferable over stock prices, because are typically weakly stationary property. Figure 2 shows that stock prices tend to have different scales and frequencies of prices. On the other hand, stock returns for all stocks are centered around zero and are more comparable across the five stocks. From the bottom-right subplot from Figure 2 that returns have low auto correlations, whereas prices are highly correlated. Working with returns, also allows us to generalize our trading model to trade any stock, not just the stock we trained our model on.

[1] `https://finance.yahoo.com/`

Figure 2: Plots for Stocks of Interests on Their Original Prices and Returns

3 Experiment

3.1 Environment Settings

We will train our model purely on Google (GOOG) stock data during 2015-01 to 2018-11, and test on the remaining data, which is approximately 70/30 train and test split. To check the robustness of our model, we trade the other stocks using the Q-function we learned from trading Google stock. We will assume a $1 million portfolio for our trading experiment. We give our model six possible actions,

1. Hold position
2. Buy 5% of portfolio
3. Buy 10% of portfolio
4. Sell 5% of portfolio
5. Sell 10% of portfolio
6. Sell 20% of portfolio

We purchase and sell as percentages of our portfolio, so that the share price of a particular stock does not affect the decision our model has to make. For example, our model simply chooses to buy \$50,000 worth or sell \$100,000 worth of our portfolio, and determines the number of shares necessary to meet this value. In order to encourage high profit-taking but discourage large losses, we came up with the following reward function,

$$\text{Reward }_t = \begin{cases} 3 & if r_t > 30\% \\ 2 & if 10\% < r_t \leq 30\% \\ 1 & if 0\% < r_t \leq 10\% \\ -1 & if -10\% \leq r_t < 0\% \\ -2 & if -30\% \leq r_t < 10\% \\ -3 & if r_t < -30\% \end{cases} \quad \text{where } r_t \text{ denotes the return rate.}$$

3.2 Parameter Selection

To initialize our model, we focus on two tuning parameters: hidden size and epoch number. For the two parameters, we set up each a candidate pool with four values, given as follow,

1. hidden size = 10, 50, 100
2. epoch = 5, 70, 120

Hidden size assigns the size of linear layers of each layer of the Q-network, and epoch indicates the number of iterations. An increase in either of the two parameters results in higher model complexity. We trained the model and see to testing results for the best combination that gives best result while eliminates overfitting issues. The result is as follow,

	Test Period Profits ($)			Train Period Profits ($)		
Hidden Size/Epoch	**5**	**70**	**120**	**5**	**70**	**120**
10	57,489.79	242,759.2	243,789.9	112,006.2	651,586.7	471,679.9
50	328,939.1	154,586.7	223,046.1	852,119.3	387,565.8	415,642.7
100	248,775.1	254,428.7	353,789.7	810,394.3	424,773.9	696,878.3

The three highest profits are generated by the (Epoch, Hidden size) combinations (120, 100), (5, 50), and (70, 100). We choose the third combination in our following analysis in order to strike a balance between overfitting and underfitting.

3.3 Model Structure

Our Q-network is a three-layer neural network. Each layer applies a linear transformation which is 100 hidden layers deep, and the hidden activation functions are Relu. The output layer is a vector with size equal to the number of possible actions. We used memory size of 200 to make record of 200 actions and its corresponding observations. We also used batch size of 50, discount factor (to balance immediate and future reward) of 97% with learning rate of 0.001 to train on 70 epochs.

4 Results

In the results that follow, it will become apparent to the reader that our model is quite effective at trading, and almost every trade decision it makes will result in profit. However, our goal is not only to build and train a model that trades well, but we also aim to understand the trading logic that our model learned. We decided to color-code our plots in order to illustrate the decisions that our model made during the process. For each of the stocks that we traded using our model, we show four subplots. The first is simply the price series, the second shows the number of shares bought and sold over time, the portfolio plot shows the value of our portfolio which consists of cash plus number of shares, and the final profits plot shows the gains and losses from trading activity.

4.1 Google Training

First, we discuss the results from our model training on Google stock. From Figure 3 we notice that our Q-learning algorithm has learned an effective policy that buys google stock when the price is relatively low and sell stock when the price is high which is a basic fundamental to trading. In addition, our reward system is effective since our algorithm has learned to sell a larger amount of shares in order to generate more profit. On the training set, every buy/sell trade that our algorithm made resulted in profit.

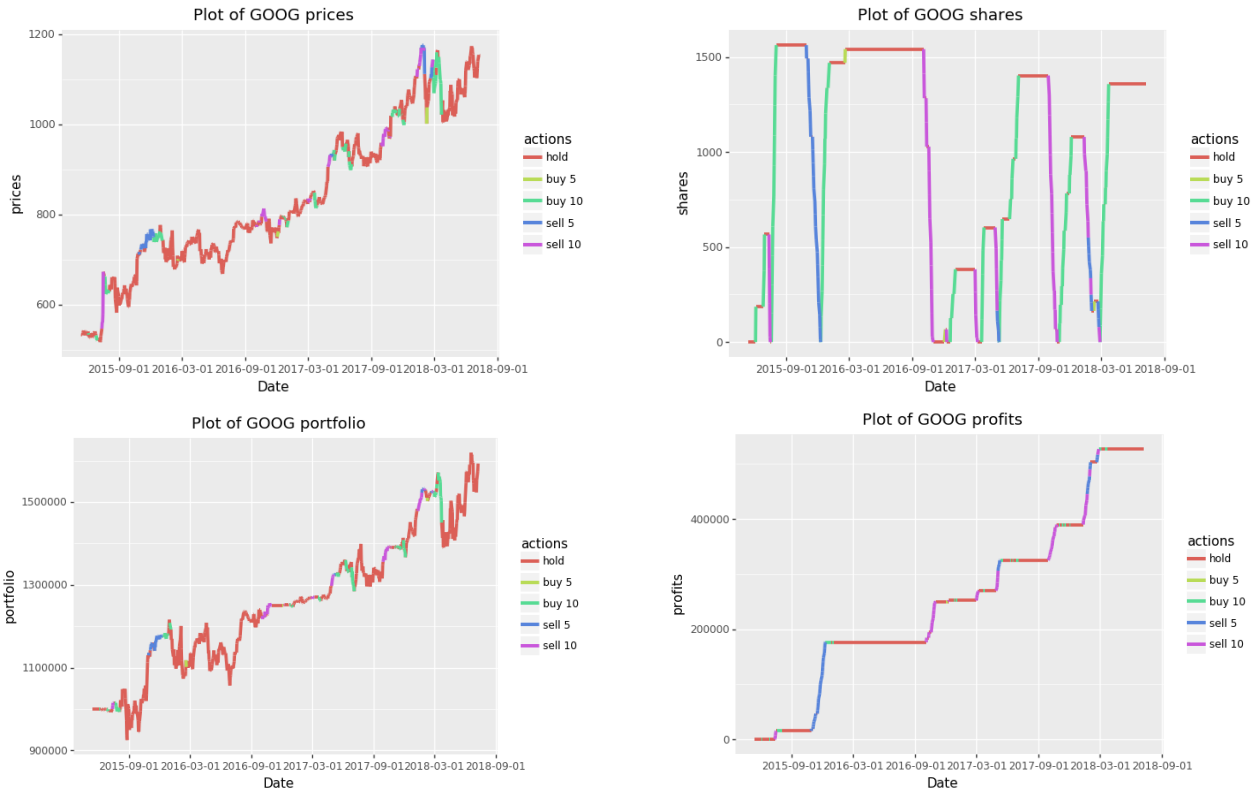

Figure 3: Google(GOOG) Training and Stock Prices

4.2 Google Testing Performances

Figure 4 shows that the Q-learning algorithm performed well on Google's testing set. The upper-right plot of Figure 4 shows that typically when the stock price is decreasing, our algorithm buys shares (From end of 2018 to March, 2019). Also, we see that selling of shares occurs before, during, and after the stock price reaches a local peak. After each selling period, the portfolio typically has sold all of its shares.

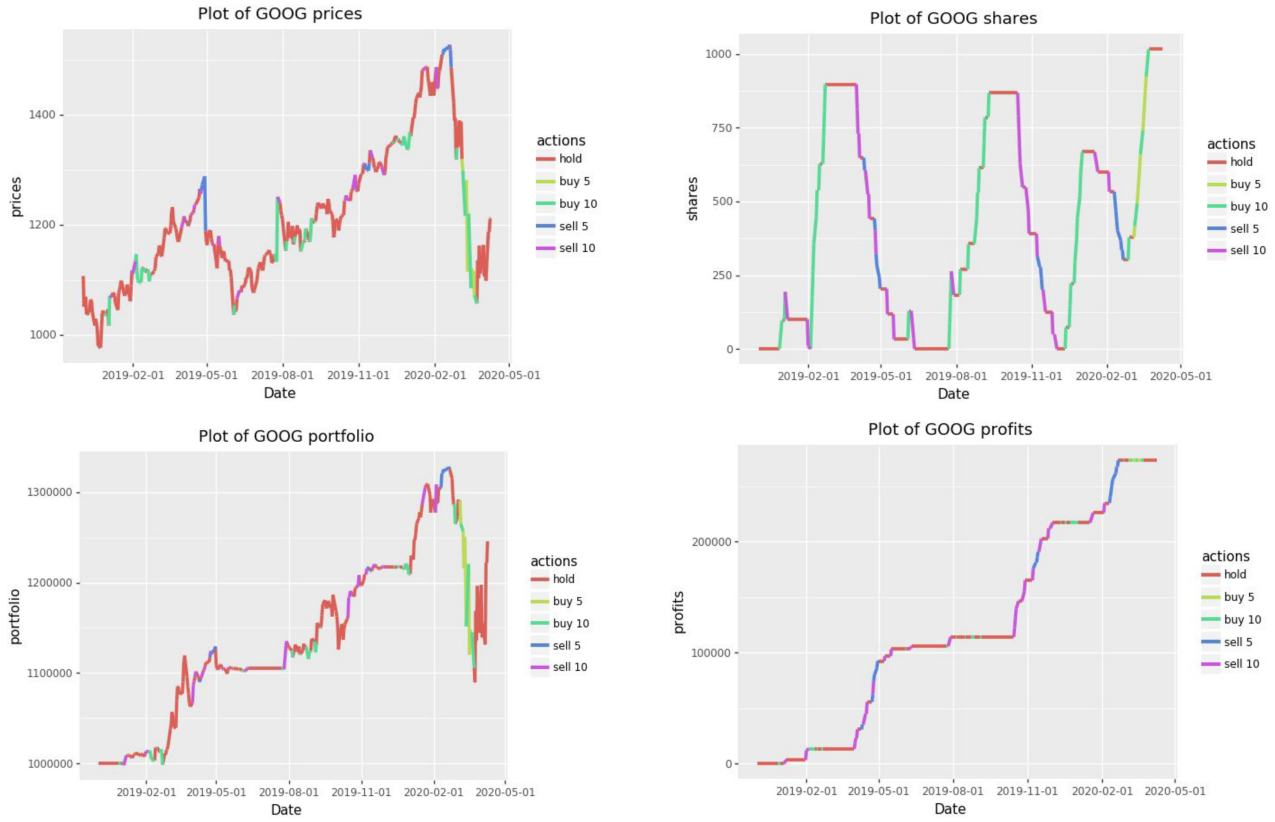

Figure 4: Google(GOOG) Test Performances and Stock Price

4.3 Testing Other Stocks

Next we test the performance of our model on the other stocks mentioned in the introduction. Figure 5 shows the performance of our model on trading Apple stock. We see that our model generated excellent profits from trading Apple stock, with more trade-sell phases compared to our model performance on the Google testing set. During the testing period which spans roughly two years, it gained 500,000 dollars in total profits. Figure 7 shows the annual returns from our trading, and we see that our model achieved 54% annual return over the test period on Apple stock. For trading results of the remaining stocks, please see the Appendix.

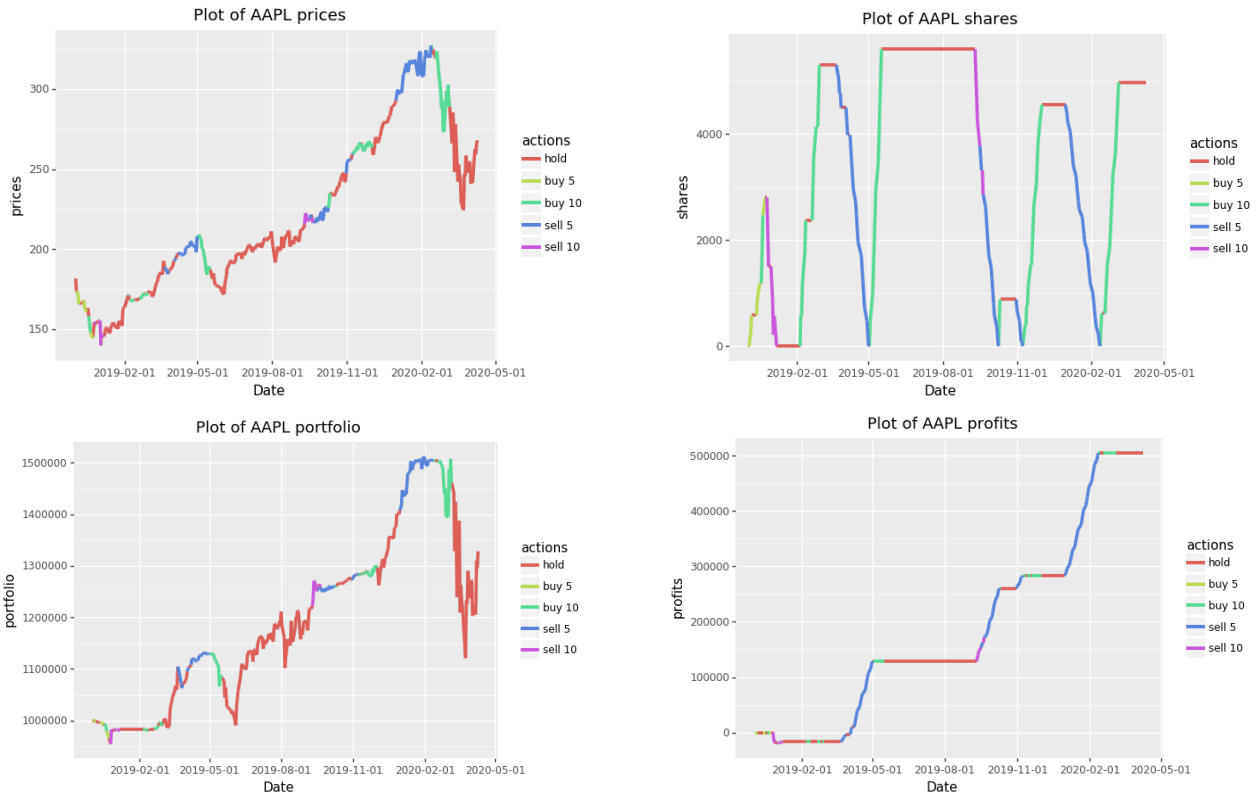

Figure 5: Apple(AAPL) Testing Performances and Stock Price

From Figure 6 we see that our algorithm was also able to trade the other stocks effectively, where the algorithm had the best trading performance for Apple and Nvidia. While the market suffered losses due to the corona virus, our algorithm outperformed the market and still had positive returns after the corona virus period in mid-February through mid-March. Figure 7 shows the returns from our trading process and also the Sharpe ratio, where we used the 10-year treasury bond as our risk-free asset. The Sharpe measures the average return earned in excess of the risk-free rate per unit of volatility. It is defined as,

$$SharpeRatio = \frac{R_p - R_f}{\sigma_p} \tag{3}$$

where R_p denotes the return of portfolio, R_f denotes the risk-free rate, and σ_p denotes the standard deviation of portfolio's excess return. Generally, sharpe ratios above two are considered good trading strategies.

We illustrate trading performance of all stocks over the training period and the testing period. These results correspond to data that our model has not seen before since we only trained on Google stock during the training period. Note that our model did not trade the S&P 500 index, but we include it in the table to use as a benchmark for performance (the Brown Line). We see that during the testing period, annual returns are above 22% for all stocks that were traded by our model, and the Sharpe ratios are also high from Figure 7. Our model was able to outperform the market benchmark performance in both the training period and the testing period.

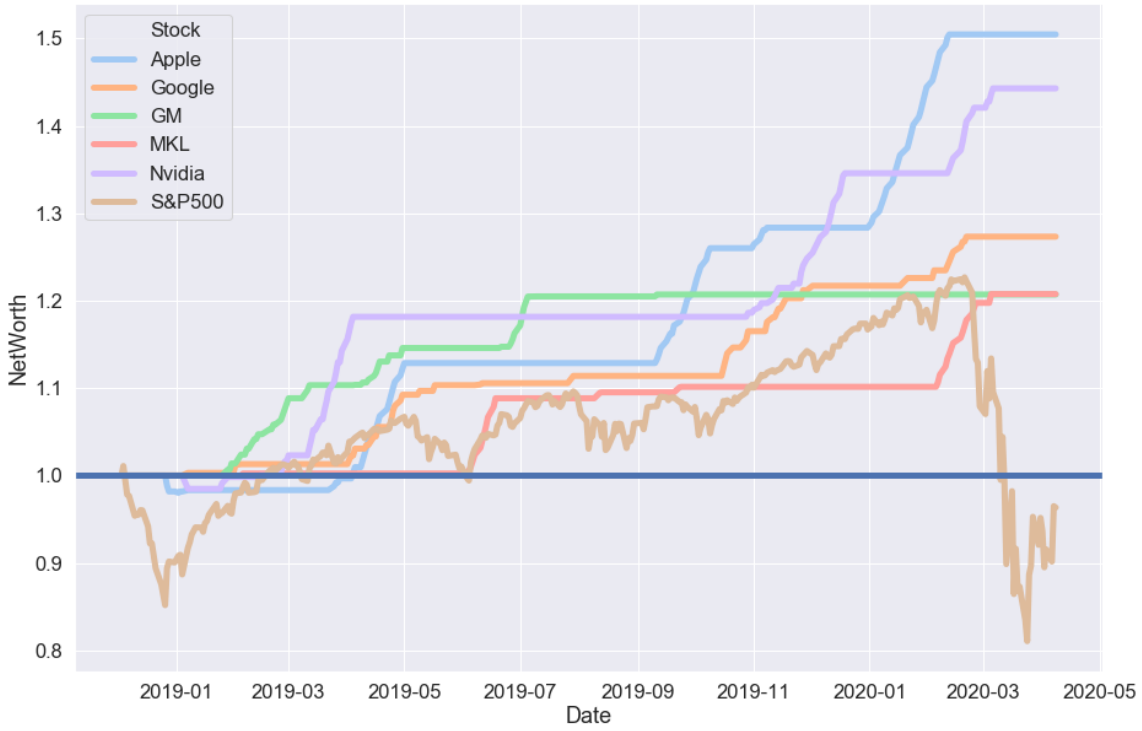

Figure 6: All Stock Performance on Their Net Worth from Model's Trading Results

	Ticker	Train anual return	Train sharpe ratio	Test anual return	Test sharpe ratio
0	S&P500	0.13	0.70	-0.00	-0.07
1	AAPL	0.21	5.77	0.54	10.70
2	GOOG	0.21	5.98	0.29	6.79
3	GM	0.15	3.85	0.22	5.68
4	MKL	0.09	2.85	0.22	5.57
5	NVDA	0.78	14.44	0.47	8.49

Figure 7: Result Table for the Stocks Tested.Note that SP 500 is the benchmark for comparison.

5 Conclusion and Discussion

5.1 Conclusion

We find that reinforcement learning can be an effective tool for developing trading strategies to make profiting trade decisions. By training our model on a small numbers of stocks' returns(like we showed in the experiment by only using Google stock returns), we were able to extend our model to trade in virtually any stock from a diverse industries. This illustrates that our model was able to learn a sensible buy low/sell high trading logic, without overfitting to training data.

5.2 Discussion

We consider other extensions and improvements we can make to the model that were not implemented in our report. So far we have only implemented a basic version of a Q-learning algorithm from Q-learning families. There are more advanced algorithm designs from Q-leaning families and from general Reinforcement learning families we could learn from to improve our model. For example, double Q-learning proposed by Hado Hasselt [2] is a possible next step to try out, because double Q-learning adopts a more complex value function to train in a mutually symmetric way, which may capture more information from the training data.

In addition, there is room to improve the deep learning structure within the Q-learning framework. By building upon our current model structure, we could expect better performance by experimenting with more sophisticated deep learning structures and fine-tuning techniques. For example, a discriminate fine-tuning process with unfreezing module for adjusting learning rate could be generalized in our structure for tuning the discount rate, which is one of the key parameter in our Q-learning structure. The idea is from Jeremy Howard and Sebastian Ruder [3].

Lastly, on the financial side, there are portfolio optimization techniques that can be incorporated into our model, such as periodically re-balancing our portfolio in order to achieve maximum possible Sharpe ratio.

References

[1] Nicole Bäuerle and Ulrich Rieder. *Markov decision processes with applications to finance*. Springer Science & Business Media, 2011.

[2] Hado V Hasselt. "Double Q-learning". In: *Advances in neural information processing systems*. 2010, pp. 2613–2621.

[3] Jeremy Howard and Sebastian Ruder. "Universal language model fine-tuning for text classification". In: *arXiv preprint arXiv:1801.06146* (2018).

[4] Raymond Rishel. "Optimal portfolio management with partial observations and power utility function". In: *Stochastic analysis, control, optimization and applications*. Springer, 1999, pp. 605–619.

[5] Hado Van Hasselt. "Reinforcement learning in continuous state and action spaces". In: *Reinforcement learning*. Springer, 2012, pp. 207–251.

6 Appendix

6.1 Other Stocks' Results

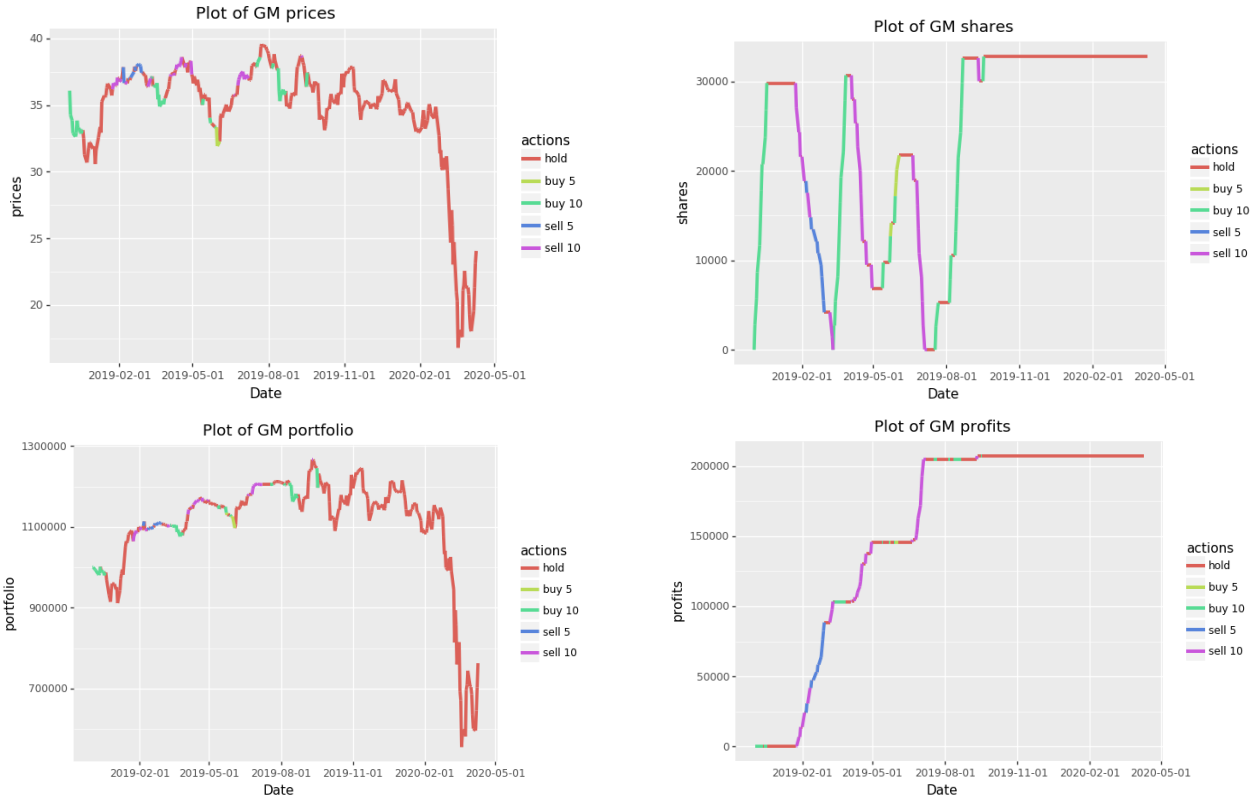

Figure 8: GM

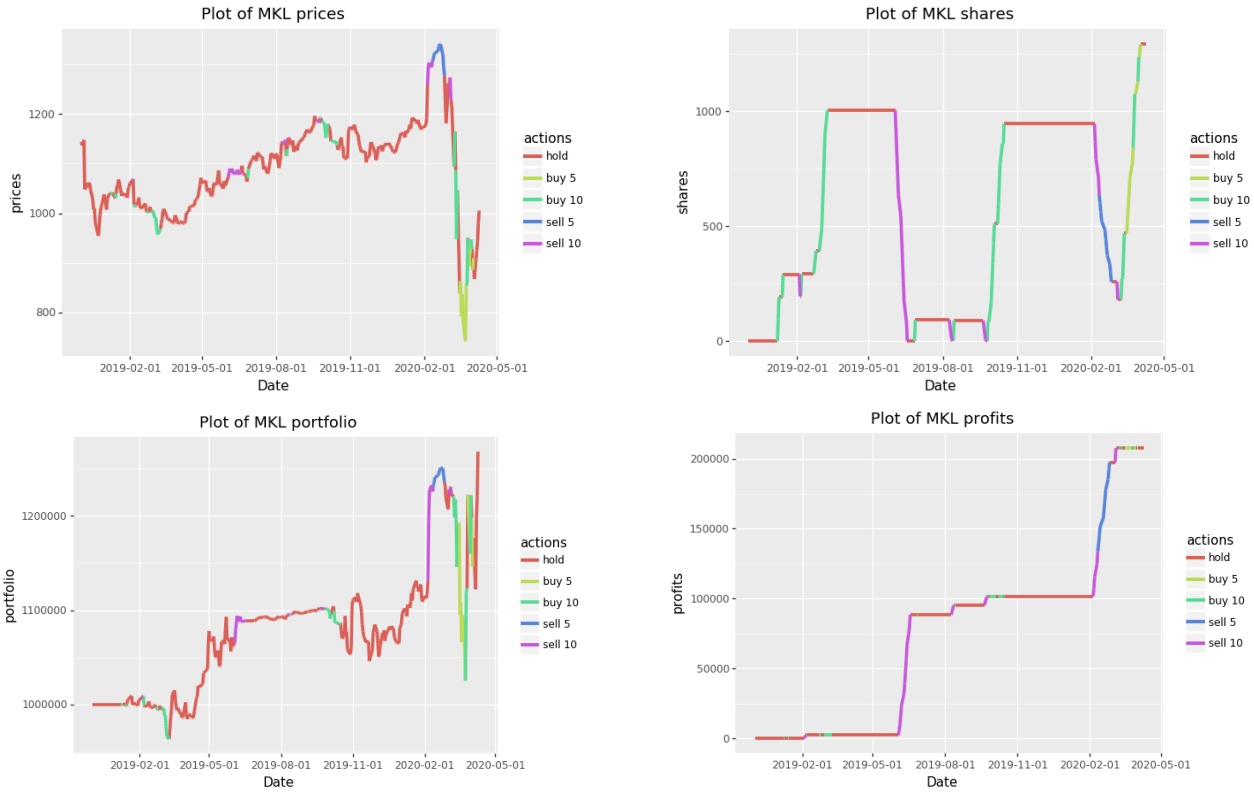

Figure 9: MKL

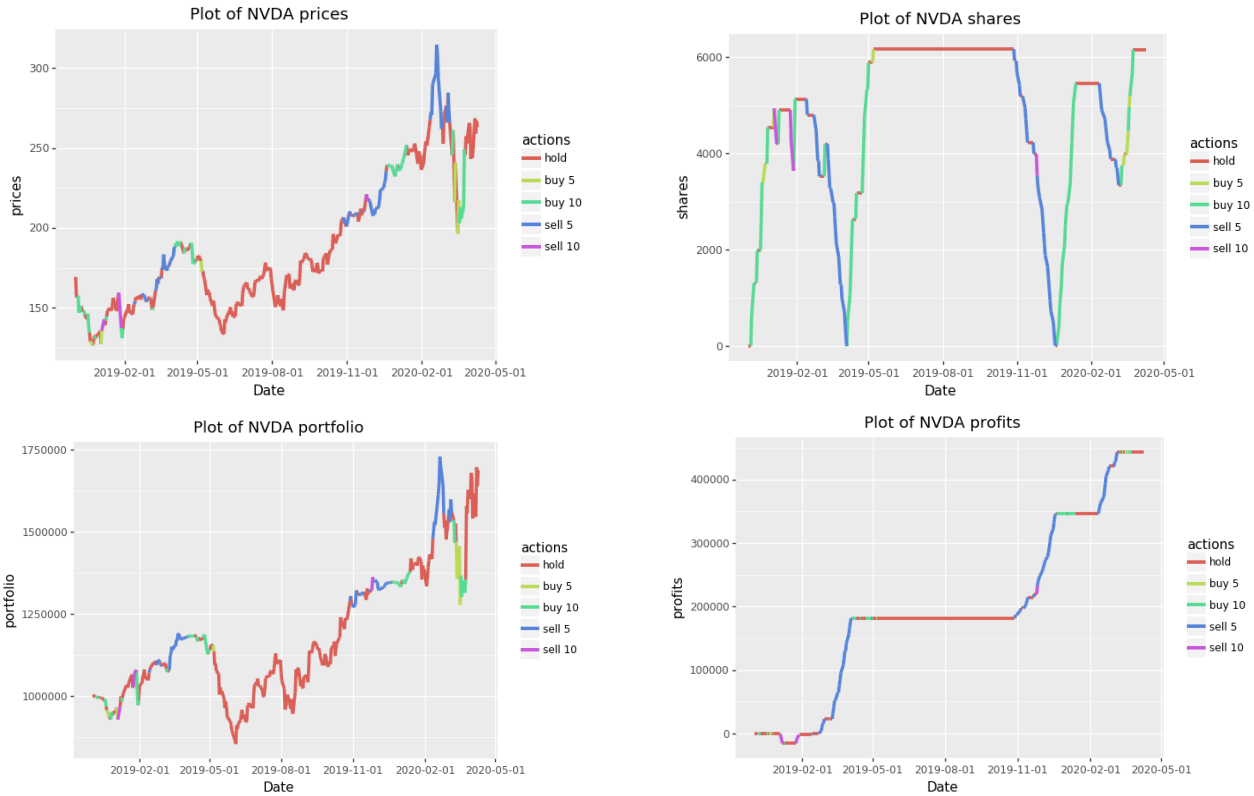

Figure 10: NVDA

6.2 Changing Discount Factors

We also changed the discount factor to 0.5 and 0.1 to see the difference in resulting testing and training profits. While discount factor 0.5 shows robustness in optimal combination selection, decreasing the factor to 0.1 gives completely different results.

	Test Period Profits			*Train Period Profits*		
Hidden Size\Epoch	**5**	**70**	**120**	**5**	**70**	**120**
10	97550.3	265044.5	215135.3	574846	725895.9	682465.7
50	338305.7	244030	207883.7	952642.1	665029.6	558940.3
100	249874.3	274474	242955.3	988432.3	734970.6	788963

Table 1: Discount factor = 0.5

	Test Period Profits			*Train Period Profits*		
Hidden Size\Epoch	**5**	**70**	**120**	**5**	**70**	**120**
10	343157.9	189856.7	136558.6	921601.4	812006.5	730646.7
50	229440.6	147386.3	159337.4	629853.4	368414.5	511313.6
100	141027.1	154131.5	289756.2	541711.3	518510.2	826954.2

Table 2: Discount factor = 0.1