Using Deep Reinforcement Learning for Quantitative Trading

Ryan Conn
rconn478@tamu.edu

Zihao Zheng
zzh523710043@tamu.edu

Link to youtube video:
https://www.youtube.com/watch?v=Uc-mBkDuvmo

Our GitHub repo:
https://github.com/ryan-conn/RLQuant

Libraries and External Code Usage

We built our trading environment as a child class to Google OpenAI's gym environment. Backtrader was used to fetch and parse historical stock data, as well as computing indicators for each stock, and Tensorflow was used to create neural networks for Q value approximation. We also used pandas and NumPy to help with additional data parsing and numerical calculations, and matplotlib to generate the graphs used during testing.

ABSTRACT

Quantitative trading using deep reinforcement learning (DRL)-based approaches have become a hot topic for machine learning researchers in the past few years **[1,2,3]**. Many different approaches have been taken to try to model and profit off of the unpredictable nature of the stock market, and there is still debate over which DRL approach gives the best results in predicting market trends or individual stock **[1,2]**. Still, it is generally agreed that DRL-based approaches are well suited for quantitative trading **[1,2,3]**. In this paper, we present our attempt to recreate some of the state-of-the-art approaches in using Deep Reinforce Learning to predict market trends and trends of individual security. Using publicly available tools and data, we were able to base our work on some of the best-performing DRL approaches to quantitative trading and perform our own fine-tuning to these models. In our evaluation, we will demonstrate that in backtesting, our model can take advantage of investment opportunities in individual stocks, refrain from investing in poorly-performing stocks, and exhibit better than market average performance when applied to the stock market as a whole.

CCS CONCEPTS

Machine learning - Reinforcement learning - Deep Reinforcement learning - application of deep reinforcement learning in quantitative trading

MOTIVATION

The motivation to train a DRL agent for quantitative training is simple— there is always unrealized profit in the stock market, and being able to take advantage of market conditions can result in large profit. However, the volatility of the stock market poses a real risk to all investors: there is always a risk that investments will decrease in value, losing part of your investment. As such, this research is motivated by the desire to have a more reliable tool to help predict the trend of individual stocks as well as the stock market as a whole, or even take over investment management entirely.

1 INTRODUCTION

The stock market is infamously hard to predict: traders can often lose or gain a large amount of money within a single day, and even large stockbrokers make catastrophically bad investment decisions. Many different statistical methods have been developed to try to predict stock market futures, including various indicators representing how well a stock is doing **[14]**, but due to the high stochasticity of the stock market, none of these methods can predict future stock prices with 100% accuracy.

Reinforcement learning is often used to make a series of decisions in a Markov decision process (MDP), where an agent is presented with a state and set of possible actions, then made to optimize expected reward over a distribution over successor states and their associated rewards. It has been used to solve various problems, including playing games, process control, and autonomous driving. Given the recent successes of deep reinforcement learning and its suitability for sequential decision processes, it makes sense to model the stock market as an MDP and try to maximize monetary gains.

2 BACKGROUND

The idea that machine learning models can successfully predict the trend of individual stock as well as market trends using market data alone

(including open and closing price, the amount of stock traded, etc.) is based on two hypotheses. The first of these is that numeric stock market data alone is enough to model the future of the stock market: attempts to create such models go back to 1965 or earlier, and such models have been used historically in order to profit off of the stock market **[14]**. The second hypothesis is that a machine learning model can successfully predict the outcome of the stock market: given that such a model exists, a sufficiently sophisticated machine learning algorithm should be able to understand these trends and capitalize on them.

Due to both hardware and practical limitations, modeling all the dynamics of the stock market is infeasible. Theoretically however, the numerical data provided by the stock market alone should be sufficient for training a machine learning agent to be able to predict the general trends of an individual stock symbol or the market as a whole. Such approaches have been tried before and demonstrated to be successful on historical data, and we hope to apply these findings to both individual stock trading and use on the market as a whole **[4, 6, 8]**.

3 RELATED WORK

3.1 Deep Learning

Deep learning is the concept of using a neural network with multiple layers as a method of nonlinear function approximation **[12]**. These neural networks are essentially interconnected graphs of nodes with various edge weights and are trained by adjusting these nodes via gradient descent and backpropagation through each layer of the neural network.

Traditional feedforward neural networks have become attractive to scholars and industry professionals after demonstrating strength in areas such as pattern recognition in solving problems such as graphic recognition and audio to text translation.

Recurrent neural networks (RNNs) are a class of neural networks that are specifically good at processing a sequence of input, making them suitable for sequential tasks such as handwriting and speech recognition and market forecasting. Different from the feed-forward structure of standard neural networks, the interconnected layer of an RNN allows it to hold internal memory. This property allows RNNs to predict future trends with more accuracy than standard neural networks, as they can consider states before the previous for a more robust analysis of input data.

Improvements to RNNs such as long short-term memory (LSTM) layers have been proposed as improvements to conventional RNN structure, solving previous existing obstacles faced by RNNs such as the vanishing gradient problem **[11]**, allowing the agent to pick up correlation within a long sequential time series more efficiently, and are now the leading approach when it comes to making predictions based on trends: more than 40% of papers published in recent years using deep learning to predict the stock market used LSTM layers **[11]**.

3.2 Reinforcement Learning

Reinforcement learning allows the automation of goal-directed learning and policy-based decision-making as a process. Essentially, an RL agent interacts with the environment through a predefined series of actions, while updating its agent through the reward function as well as observing the change of the state returned by the environment. In general, RL can be classified as model-based or model-free. As the generalization of highly stochastic environments into a model is very challenging, model-free RL approaches are more popular in such environments due to the ease of implementation and training **[1]**.

In recent years, deep reinforcement learning has picked up steam starting with Google's DeepMind team creating deep reinforcement learning by integrating deep learning with Q learning and achieving above-human performance on many Atari games **[15]**. The usage of a neural network within an RL agent allows it to represent complex data in a multidimensional environment. In the work of Wadhwani, Gupta, etc., Deep Reinforce Learning enables them to train a market-sentiment aware agent. This novel approach allows the agent to evaluate the public opinion on the market trend in integrating this evaluation into its decision-making process **[7].** All these advantages made DRL a suitable candidate for creating and training a quantitative trading agent.

4 SOLUTION OVERVIEW

4.1 Modeling the Stock Market

Our first step towards creating an agent that could make good decisions on the stock market was to train a neural network to predict how to make good decisions on a single stock. The high stochasticity of the stock market means that price alone cannot be used as a reliable measure of how a stock will do: instead, several market indicators, which are used by financial analysts to represent trends in stock movement, are used in conjunction with the stock price to represent the state of an individual stock.

We selected a combination of indicators shown to have good performance in quantitative trading literature, and to make the agent more generalizable, mostly percentage-based indicators were used so that stocks with incredibly different price values would still have similar indicator values: these include percentage-based Bollinger Bands (BBPct), the exponentiated moving average of the stock price (EMA), percentage price oscillator (PPO), and relative strength index (RSI).

We also included the number of stock owned in the state representation, as the stock owned has an effect on the reward observed (described below), and as such should be known by the agent. Notably absent from the state is the money on hand: this simplifies the representation of reward as the agent doesn't have to take into account how much money is on hand, only determine which action will result in the best change in portfolio value. When trading on real stocks, if the agent believes the best action is to buy a stock and not enough money is on hand to do so, the neutral option can be taken instead. Thus, a state is represented as:

$$\{h,\ p,\ BBPct,\ EMA,\ PPO,\ RSI\}$$

To simplify decisions for the agent and decrease sample complexity, the action set is represented as the following set of trading signals:

$$A \in \{long, hold, short\} = \{-1,\ 0,\ 1\}$$

When applied to the actual trading, these signals can be interpreted in various ways, since the agent's decision simply states what the best action is to take in each situation: buying or selling more of a stock only multiplies the expected return due to that action.

The reward is represented as the difference in holding value between the current and next values of stock held. For each action, this is defined as following, where h represents the stock currently held, p_t and p_{t+1}represent the stock price at the current and next time step respectively, and c represents the added transaction cost of buying or selling as a fraction of stock value:

$$r_t = \begin{cases} (h+1)(p_{t+1} - p_t) - c * p_t & a_t = long \\ h(p_{t+1} - p_t) & a_t = neutral \\ (h-1)(p_{t+1} - p_t) + (p_t - p_{t+1}) - c * p_t & a_t = short \end{cases}$$

4.2 Training

To train an agent that has generalizable knowledge and avoid overfitting to a particular stock's trends, training must be performed across a wide variety of stocks undergoing different pricing trajectories. As such, each episode used during training is the trajectory of a single unique stock from February 1, 2017, to August 30, 2020, with a single data point at the end of each day.

To avoid training on many samples from the most recent stock (as transitions within a single stock trajectory are very highly correlated), we decided to use an Experience Buffer and trained from the experience buffer rather than individual episodes.

We decided to use DQN (**Figure 1**) as a training algorithm for several reasons: first, in comparison to on-policy learning algorithms, it can make effective use of an experienced buffer, which is important to avoid overfitting to individual stocks. We had initially tried to use Advantage Actor-Critic (A2C) for training, but results were very volatile given the wide range of possible trajectories. Second, it has been experimentally determined to have comparatively better performance than other reinforcement learning algorithms in this domain **[2, 5]**. Third, during the real application of the model, trading on multiple stocks on a limited budget allows us to determine which stocks are relatively better to purchase by comparing Q values between stocks.

```
DQN:
    Initialize replay buffer D
    Initialize/Load Q value approximation LSTM model
    Initialize/Load target model Q_hat
    For each episode
        s = environment.reset
        While s is not terminate state
            Choose best action a by model.predict_best_action(s,epsilon)
            Generate state s_new, reward r by performing a
            Add [s_new,r] to D
            s = s_new
        End For Loop
        Get mini batch from D
        Initialize train_array_x
        Initialize train_array_y
        For each sample in mini batch
            Append state s to train_array_x
            Get estimate_future_reward gamma * Q_hat.predict(s) if s is not terminal
            Append y + estimate_future_reward  to train_array_y
        End For Loop
        Perform gradient descent by batch train with train_array_x,train_array_y
        Perform epsilon decay
        if episode call for update target model
            Update Q_hat to Q
    End For Loop
```

Figure 1: Pseudocode for DQN algorithm

Through experimentation, we determined that a learning rate of 0.0001, $\gamma$of 0.99, and decaying ϵ from 0.99 to 0.01 over 100 episodes could sufficiently encourage exploration and converge on a profitable policy. One other important parameter to tune was the transaction cost: setting it too low or to 0 led to the agent making very risky decisions that led to slight amounts of profit, while setting it too high made the agent reluctant to make any trades due to the high cost. We determined that a transaction cost of 1% (0.01) could find a good balance between these two extremes.

We used an LSTM network for Q value approximation to capture the movement of the stock rather than only a single value, as research shows that LSTM networks have superior performance in quantitative trading [11]. We found that a sequential architecture consisting of a single 64-node LSTM layer taking in the 8 most recent states followed by two dense layers with 128 units was sufficient to give predictions accurate enough to make profit substantially above general market growth.

6 EVALUATION

Our results are split into two parts: performance on individual stocks and the market as a whole. Training was done on individual stocks, so these individual results better illustrate the agent's performance, while performance on multiple stocks better represents profit one could realistically expect to achieve using this approach to make trades on the stock market.

6.1 System setup

To train and perform our platform, we trained and tested the DRL models individually on our systems, which includes an Nvidia GeForce RTX 2070 and an Nvidia GeForce RTX 2080.

6.2 Data set

Evaluation was done similarly to training: by using historical stock data. Initially, we trained and evaluate our model on dataTo make sure the model is not using market trends from the time frame of the training set and giving itself an unfair advantage in the testing phase, a separate model was trained with the data from February 1, 2017 to August 30th of 2020. Then, testing was done in two sets: one on unseen stocks from the same time frame as the training data, and another on stock data from September 1st, 2020 to April 1st, 2021 (after the end of the training period). Testing was done in two sets to see how much of the model's prediction was based on current market trends vs. insights into stock trends overall. All testing results are shown in **Appendix I and II**.

The model was trained over a total of 300 episodes, where each episode represents the trajectory of a single unique stock from the start to the end of our training timeframe.

To demonstrate that our agent performs better than simple random investments, we compared the observed performance to that of an agent that simply invested in random stocks by either holding or buying at each time step with equal probability. This action set does not include selling, as the mean result of this action set is for the value of holdings to stay roughly the same: instead, this simulates investment in stocks and waiting it out to see the results. A comparison between our agent and this baseline can be seen in **Picture 5**. Note that because of the high volatility of this baseline due to its randomness, figures showing the outcome of random agents are chosen from a representative random run, where the outcome was similar to the mean of 10 test runs.

6.4 Results

Figure 3 demonstrates the performance of our model against 4 declining stocks.

We observed that our model generally does well when stocks rise in value (**Figure 4** illustrates its performance on AAPL and TSLA, which saw large gains in the testing period), but is not always robust at cutting its losses on stocks with large losses as shown in **Figure 3**. While it managed to keep roughly the same value or gain slight profits on some stocks that saw large declines, this was not always the case, which hurts the general performance of the model when testing it against random pooled market data. As a result, we further fine-tuning our model to handle these cases, which we demonstrated the result in the above demonstration.

Figure 6 demonstrates the performance of the model we trained in general. In this case, we pooled 300 randomly selected symbols in 3000+ symbols in our pool as a control group. While performing averaging when train again the random sample stocks, we can see how it excels in stock exclusively pooled from DOW, which is a group of well-known

and strong symbols with known good performance in the past. Our models consistently outperformed the DOW industrial average during testing by a large margin, with most achieving over double the gains compared to the DOW average over the same time period.

One the second part of testing, where testing data are gathered after the training time range, we observed our model's behavior deteriorated from the impressive results observed within the training time period. These results can be found in **Appendix II**

In **Figure 7**, we observed that while our model can break even or even make money on some stocks that trend downward, it still takes heavy losses in some situations.

In **Figure 8**, we observed that while being tested in a more realistic scenario, our model can sometimes fail to invest in certain high gain stock which results in losing profit, but despite these setbacks, the RL agent still performed exceptionally well in trading some volatile symbols, as shown in **Figure 9**.

This degradation in performance can be clearly seen in our tests on multiple stocks. As displayed in **Figure 10**, our model's performance is acceptable(as it still generated profit). but while it is following the market trend in general, it failed to outperform random action, which means further work is necessary to make our approach feasible for real portfolio management.

7 CONCLUSIONS

Through the experiment we conducted, we prove that with a limited amount of training and fine-tuning, we can train a reinforcement learning model to perform relatively well on actual market data. Our method of modeling the stock market is unique among the papers we read in a couple ways: the lack of money as part of state and a reward function based on time-step differences in holding values that performed better with our model than those suggested in other quantitative trading literature.

There are two main shortcomings to our current approach, however: while our model is sometimes able to protect against losses on stocks, this is not always the case, and this failure to detect future losses leads to the model sometimes losing large sums of money. Second, our model performed slightly worse on stocks after the training set: while single-stock performance remained comparable for single stocks and random stocks (generally outperforming the average), performance on the market as a whole suffered significantly in our testing on DOW stocks, performing similarly to random investments. As a result of this, more testing must be done before this can be considered a feasible approach to quantitative trading.

8 FUTURE WORK

One weakness of our current approach is that it fails to consider the opportunity cost of investing in stocks: since the agent is trained on a single stock, when applied to multiple stock trading it cannot take into account whether investing in a different stock at a later time might result in a higher return. Generalizing our approach to multi-agent reinforcement learning could allow agents to gain more profit by waiting out a small immediate return for a more promising future opportunity in a different stock. This flaw is especially evident when looking at performance when the performance of our model on multiple stocks is compared to the random baseline: while in most cases our model does exceptionally well at minimizing losses in individual stocks, the greedy choices taken by our model working on multiple stocks in parallel resulted in worse-than-average performance in some of our test results.

Another weakness of our current approach is that the lack of a stop-loss (automatically selling when price drops below a certain threshold) makes our agent reluctant to sell the stock it's holding unless value decreases dramatically (as general market trends are upwards, and most of the time small downward trends are followed by a rebound in price), making losses worse than they optimally could be: this was the most common reason our agent would lose money. Recently, some blogs (not publications) have demonstrated using DRL to find the optimal stop-loss threshold: high and low value thresholds to sell stocks to maximize profit. Not much actual research has been done in this area, however: there has been a publication about optimizing pair trading (trading two similar performing stocks) taking this approach, but it is not generalizable to the market as a whole **[17]**. Future

research could focus on adding a stand-alone neural network to finding the stop-loss threshold on top of our current approach.

Even though the performance of our actor-critic approach (A2C) was poor during experimentation, some papers have shown actor-critic methods such as proximal policy optimization to show improvements on critic-only approaches **[4, 5]**. Further research could be done into the performance of such approaches to more conclusively show the most effective DRL approach for quantitative trading.

References

[1] Quantitative Trading on Stock Market Based on Deep Reinforcement Learning, Jia WU; Chen WANG; Lidong XIONG; Hongyong SUN, 2019 International Joint Conference on Neural Networks, https://ieeexplore.ieee.org/abstract/document/8851831?casa_token=d-UNxz3E99EAAAAA:9Qku7P06eyjlcTeJJXtWn-1ls3sEMD3NAQVAm3cC2IhqDl6QAzR_n8wjFKtlA3oLYrJtoWJTGA

[2] Deep Reinforcement Learning for Trading, Zihao Zhang, Stefan Zohren and Stephen Roberts, The Journal of Financial Data Science Spring 2020, https://doi.org/10.3905/jfds.2020.1.030

[3] Adaptive stock trading strategies with deep reinforcement learning methods, Xing Wua Hao lei, etc, Information Sciences Volume 538, October 2020 https://doi.org/10.1016/j.ins.2020.05.066

[4] Practical Deep Reinforcement Learning Approach for Stock Trading, Zhuoran Xiong, Xiao-Yang Liu, Shan Zhong ect. https://arxiv.org/abs/1811.07522

[5] A multi-layer and multi-ensemble stock trader using deep learning and deep reinforcement learning, Salvatore Carta, Andrea Corriga, Anselmo Ferreira, etc. https://link.springer.com/article/10.1007/s10489-020-01839-5

[6] Deep Direct Reinforcement Learning for Financial Signal Representation and Trading, Deng, Bao, Kong ect. IEEE Neural network and learning system https://github.com/nithinsethu/Deep-Direct-Reinforcement-Learning-for-Financial-Signal-Representation-and-Trading/blob/master/Paper.pdf

[7] Market sentiment-aware deep reinforcement learning approach for stock portfolio allocation, Prahlad Koratamaddi, Karan Wadhwani, Mridul Gupta, Engineering Science and Technology, an International Journal, Available online 26 March 2021, https://www.sciencedirect.com/science/article/pii/S2215098621000070

[8] Deep reinforcement learning-based trading agents: Risk curiosity-driven learning for financial rules-based policy; Hirchoua etc.; Expert Systems with Applications Volume 170, 15 May 2021, 114553; https://doi.org/10.1016/j.eswa.2020.114553

[9] Time-driven feature-aware jointly deep reinforcement learning for financial signal representation and algorithmic trading; Lei, Zhang, etc.; Expert Systems with Applications Volume 140, February 2020, https://doi.org/10.1016/j.eswa.2019.112872

[10] Vargas, Manuel R., et al. "Deep leaming for stock market prediction using technical indicators and financial news articles." 2018 International Joint Conference on Neural Networks (IJCNN). IEEE, 2018.

[11]Hu, Zexin, Yiqi Zhao, and Matloob Khushi. "A Survey of Forex and Stock Price Prediction Using Deep Learning." Applied System Innovation 4.1 (2021): 9.

[12] Khadjeh Nassirtoussi, A.; Aghabozorgi, S.; Wah, T.Y.; Ngo, D. Text mining of news headlines for FOREX market prediction: A multi-layer dimension reduction algorithm with semantics and sentiment. Expert Syst. Appl. 2015, 42, 306–324. [CrossRef]

[13] Naik, Nagaraj, and Biju R. Mohan. “Optimal Feature Selection of Technical Indicator and Stock Prediction Using Machine Learning Technique.” *Emerging Technologies in Computer Engineering: Microservices in Big Data Analytics*, 2019, pp. 261–268., doi:10.1007/978-981-13-8300-7_22.

[14] Fama, Eugene F. "Portfolio analysis in a stable Paretian market." Management science 11.3 (1965): 404-419.

[15]Mnih V, Kavukcuoglu K, Silver D, et al.Human-level control through deep reinforcement learning. Nature, vol.518, no.7540, pp.529-533, 2015

[16]Richard S. Sutton and Andrew G. Barto. Reinforcement Learning:An Introduction,The MIT Press.

[17]Taewook Kim, Ha Young Kim, "Optimizing the Pairs-Trading Strategy Using Deep Reinforcement Learning with Trading and Stop-Loss Boundaries", Complexity, vol. 2019, Article ID 3582516, 20 pages, 2019. https://doi.org/10.1155/2019/3582516

Appendix I: Testing on dates inside training period

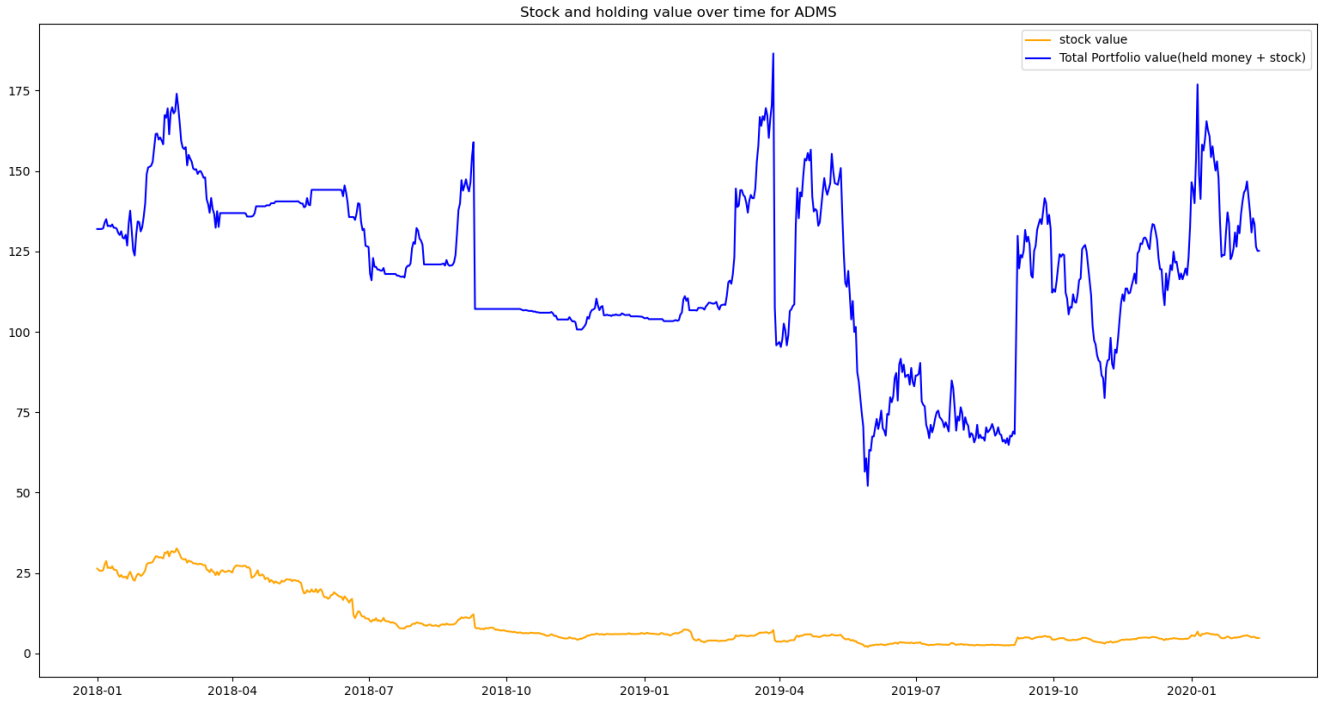

Figure 3A, Performance of model on ADMS

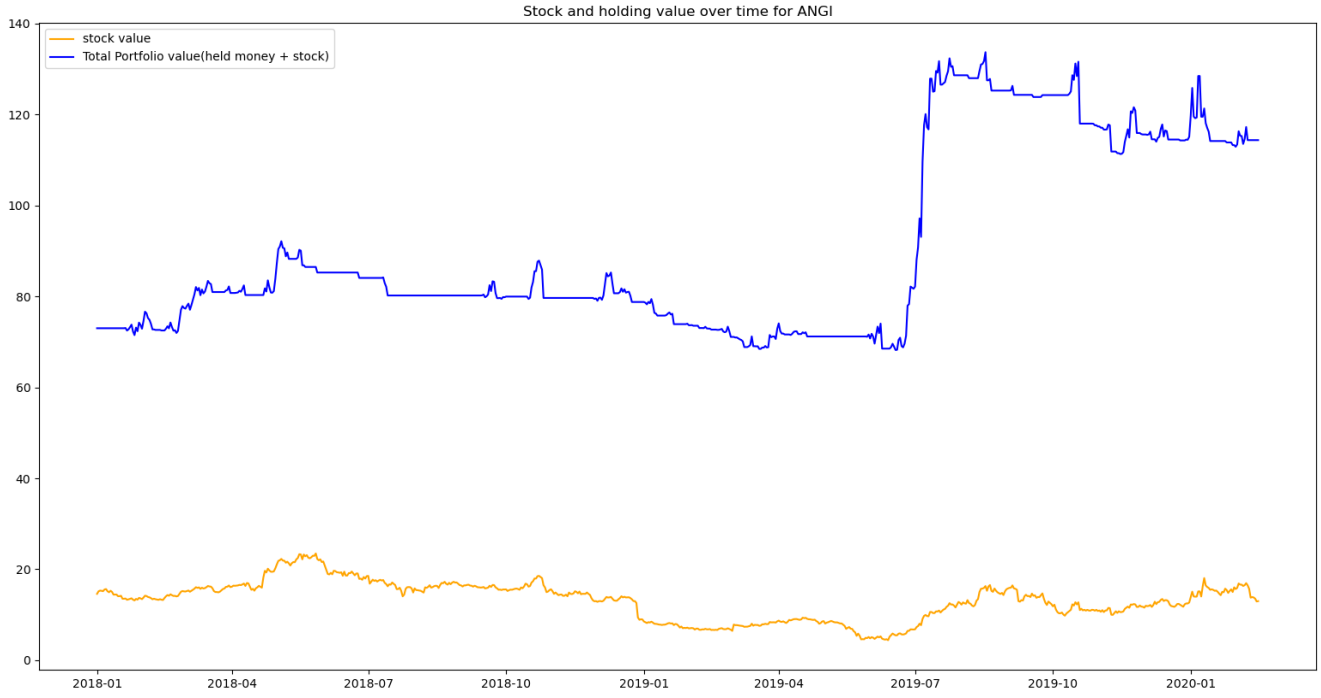

Figure 3B, Performance of model on ANGI

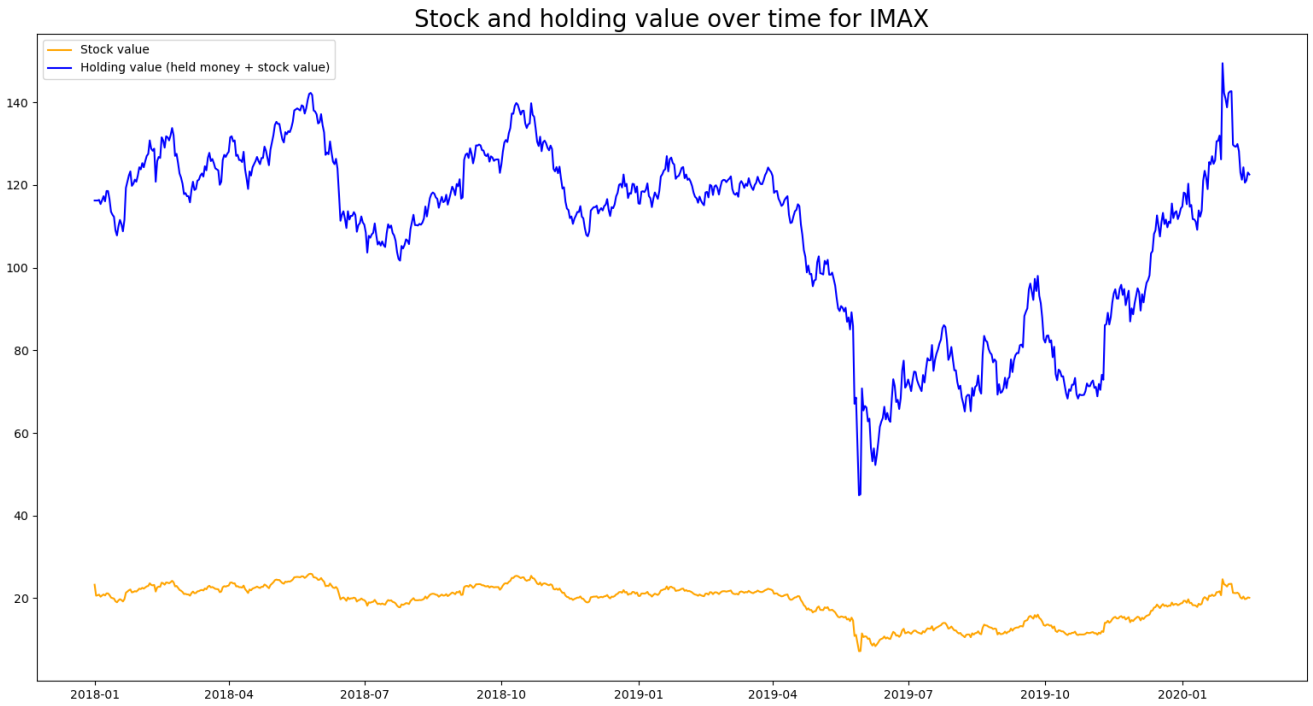

Figure 3C, Performance of model on IMAX

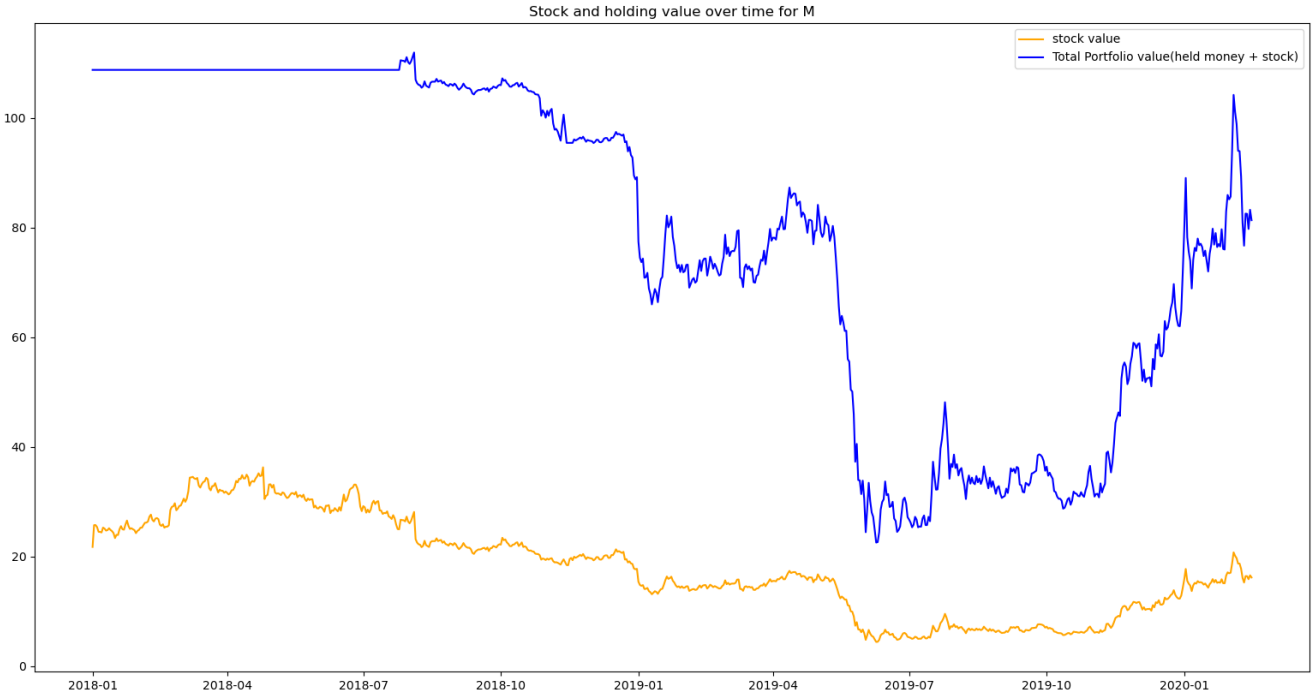

Figure 3D, Performance of model on M

Figure 3, Performance of model on poorly performing stock

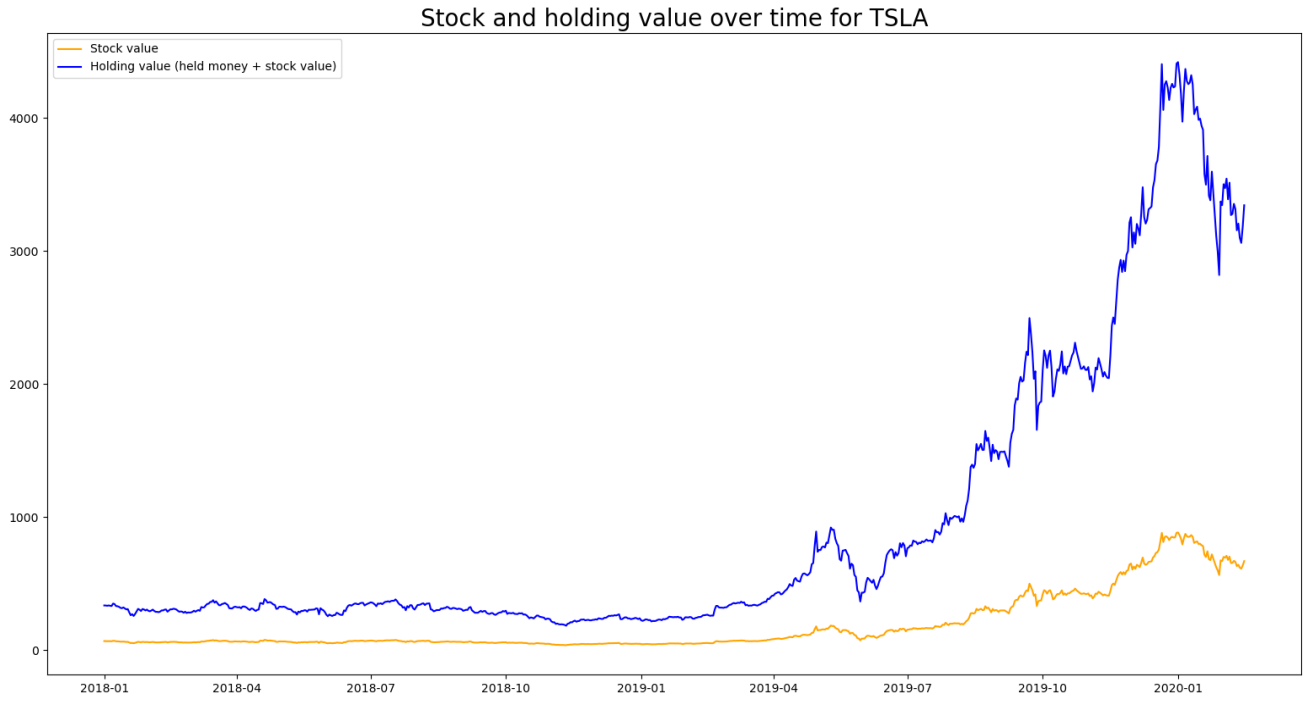

Figure 4A, Performance of model on TSLA

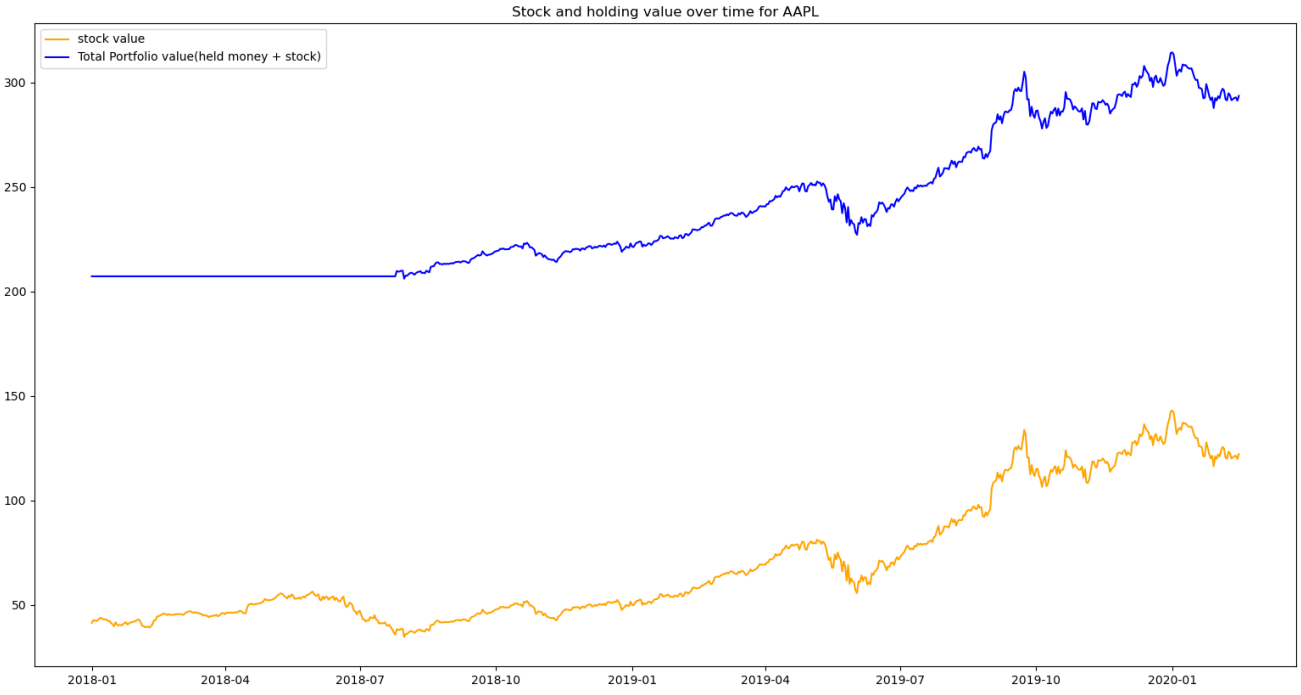

Figure 4B, Performance of model on AAPL

Figure 4: Performance model on stocks with large gains

Figure 5 A., Result of Randomly trading DOW listed stock

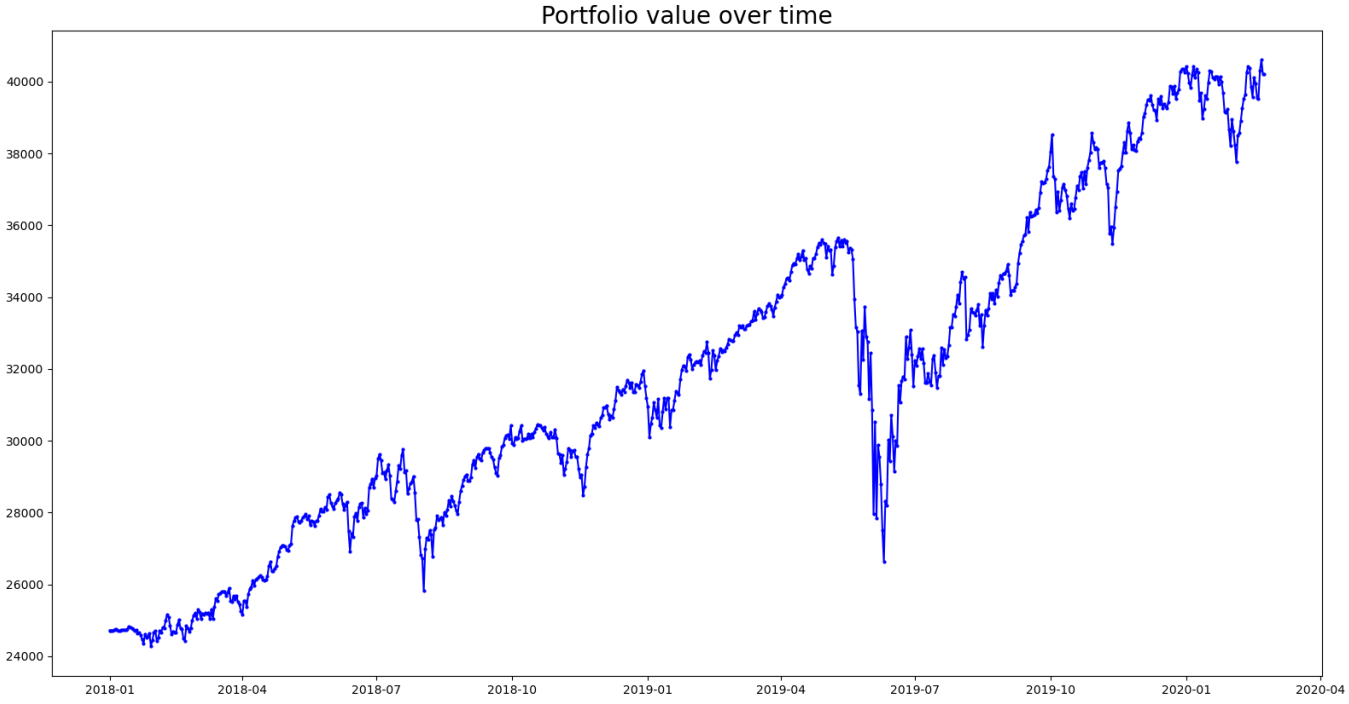

Figure 6 A, Result of using the model to trade DOW listed stock

Figure 5 B., Result of Randomly trading 300 randomly selected stock

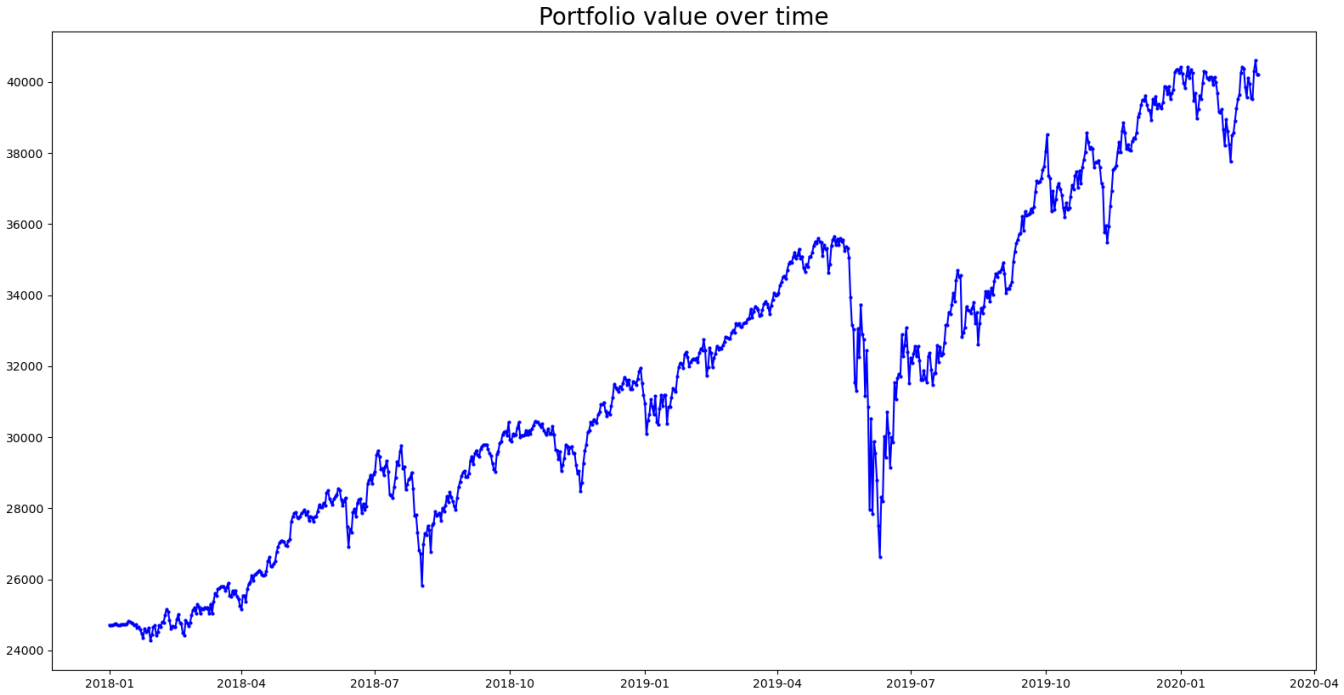

Figure 6 B., Result of using the model to trading 300 randomly selected stock
Figure 5,6 Baseline comparing to the performance of our model

Appendix II: Testing on dates after training period

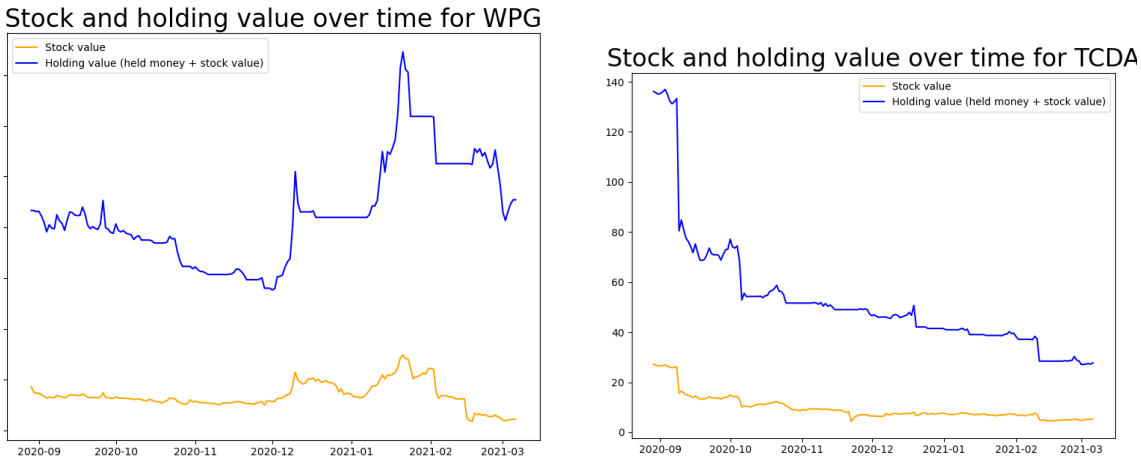

Figure 7: Performance on stocks with high losses after training time period

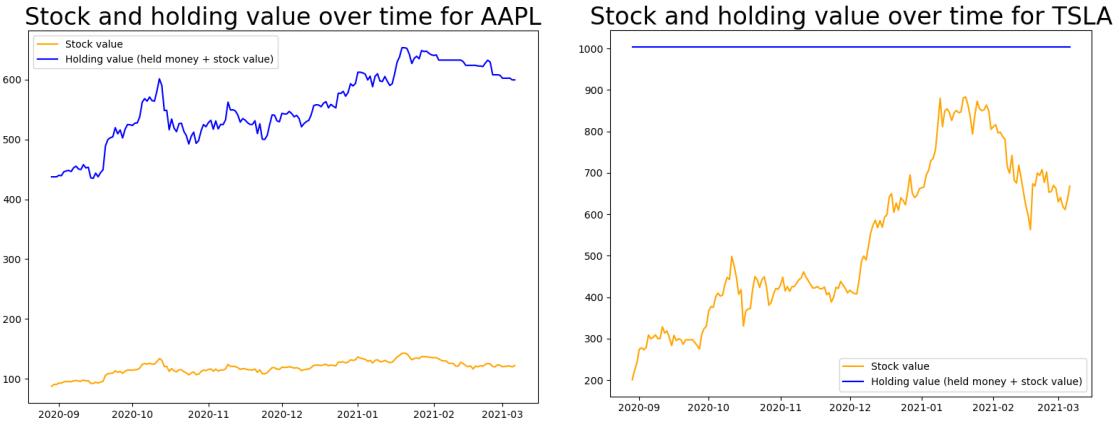

Figure 8: Performance on stocks with high gains after training time period

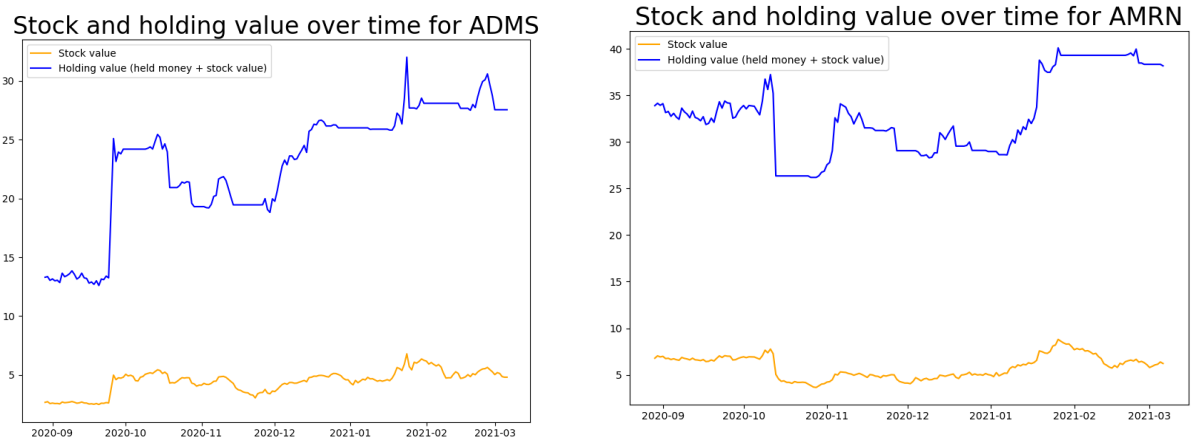

Figure 9: Performance on volatile stocks after training time period

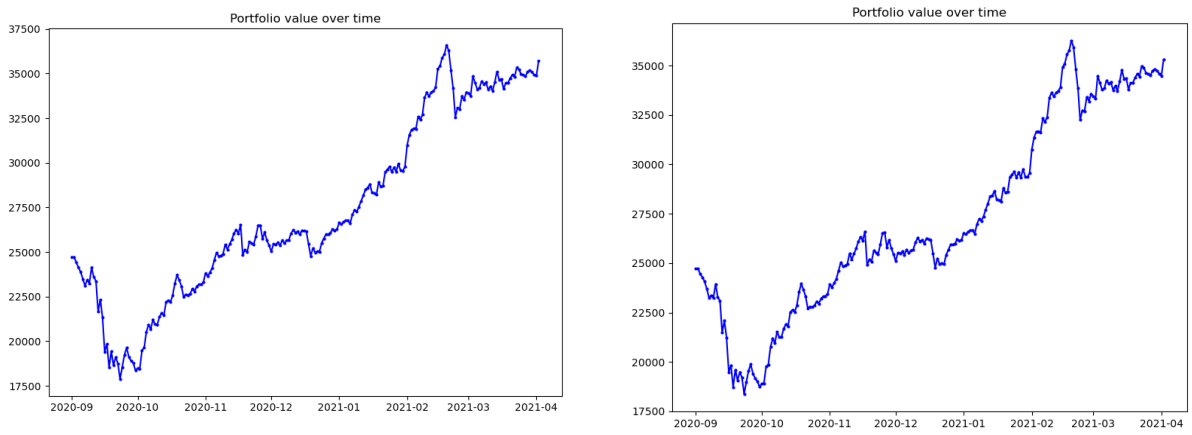

Figure 10: Performance of random actions (left) vs. model (right) on 300 random stocks after training period

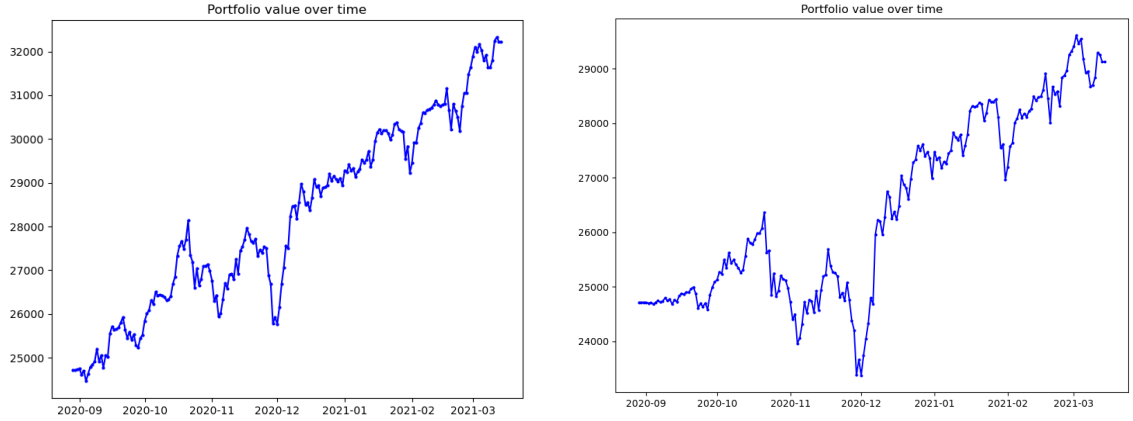

Figure 11: Performance of random actions (left) vs. model (right) on DOW stocks after training period