Machine Learning Engineer Nanodegree

Capstone Report

Abdel Affo

December 20th, 2018

Creating an automated trading agent using deep reinforcement learning

Definition

Project Overview

Algorithmic trading has grown in popularity the past few years, with an estimated 70% of US trade volume being generated through this type of trading. I've been interested in stock trading and technical analysis for a long time. Technical analysis is the belief that future stock prices can be inferred from past trading history. With my newly acquired knowledge of reinforcement learning, I believe that the stock trading problem can be formulated as a Markov Decision Process problem, with the market being the environment, and the trader or trading system being the agent.

Problem Statement

The goal of this project is to investigate if a reinforcement-learning agent can be trained to trade a given stock successfully. The agent will be trained on historical prices, and then will decide what decision to make in order to maximize profit.

Metrics

The criterion use to evaluate the models' performance will be the profit made by trading the stock over the entire period of time.

Analysis

Data Exploration

The dataset used in this project is a time series dataset provided by Quandl, free to download and use. The data set format is as follows:

Input Data fields

- Open – Opening price
- High – High price of the day
- Low – Low price of the day
- Close – Closing price
- Volume – Daily volume
- Ex-Dividend – Dollar amount of cash dividend
- Split Ratio – Ratio of the number of new shares to the number of old shares
- Adj. Open – Adjusted opening price
- Adj. High – Adjusted high price
- Adj. Low – Adjusted low price
- Adj. Close – Adjusted closing price
- Adj. Volume – Adjusted volume

Adjusted prices are back-adjusted assuming that all corporate actions are reinvested into the current stock.

The agent will be operating on a single stock, Altria Group (MO) and the dates will be from January 1st 2015 to February 28th 2017.

This data can be accessed at:

- https://www.quandl.com/databases/WIKIP/WIKI-PRICES/export

The data set will be handled as a time series set, and will be split into a training set and a test set. The training data interval will be from January 1st 2015 to December 31st 2016, and the testing interval will be from January 1st 2017 to February 28th 2017, to ensure that future data isn't used during training. There will be 504 data points in the training set, and 39 data points in the test set.

Listed below are the first five rows of our data set.

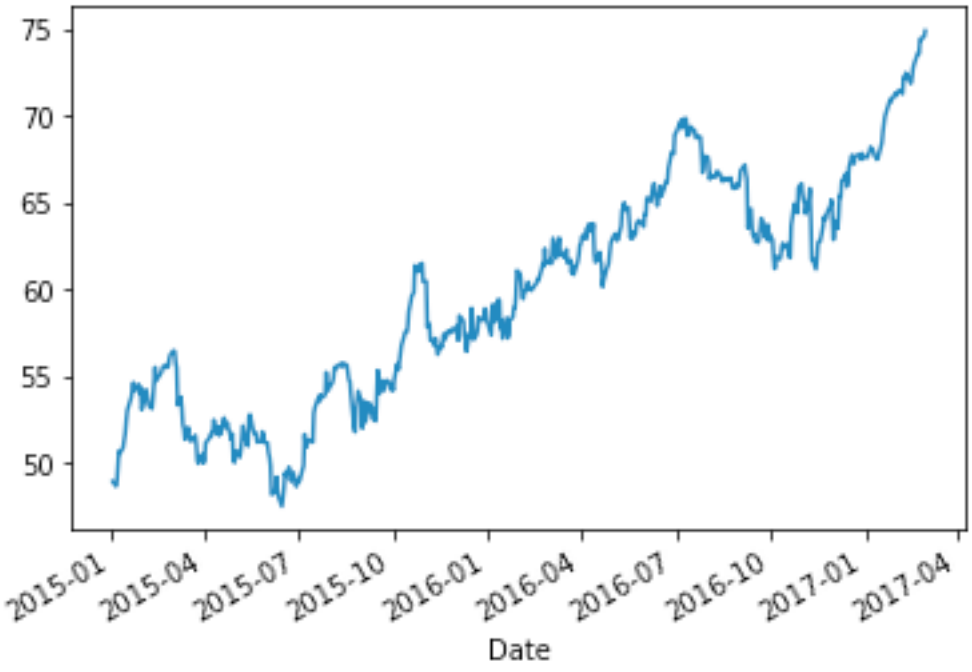
Date	Open	High	Low	Close	Volume	Ex-Dividend	Split Ratio	Adj. Open	Adj. High	Adj. Low	Adj. Close	Adj. Volume
2015-01-02	49.30	49.64	48.58	48.97	6077766.0	0.0	1.0	44.427541	44.733938	43.778701	44.130156	6077766.0
2015-01-05	48.64	48.99	48.52	48.69	6900488.0	0.0	1.0	43.832771	44.148179	43.724631	43.877829	6900488.0
2015-01-06	49.00	49.65	48.89	48.98	7239944.0	0.0	1.0	44.157191	44.742950	44.058063	44.139168	7239944.0
2015-01-07	49.33	50.03	49.32	49.88	5492436.0	0.0	1.0	44.454576	45.085393	44.445564	44.950218	5492436.0
2015-01-08	50.26	50.93	50.20	50.72	4693280.0	0.0	1.0	45.292662	45.896444	45.238592	45.707198	4693280.0

Exploratory Visualization

The features in our data can be grouped into four different categories, which are price, volume, dividend data and split ratio. Of all of these categories, only the most relevant will be visualized here. The data preprocessing step will determine which features will be used by the model, and which ones will be dropped.

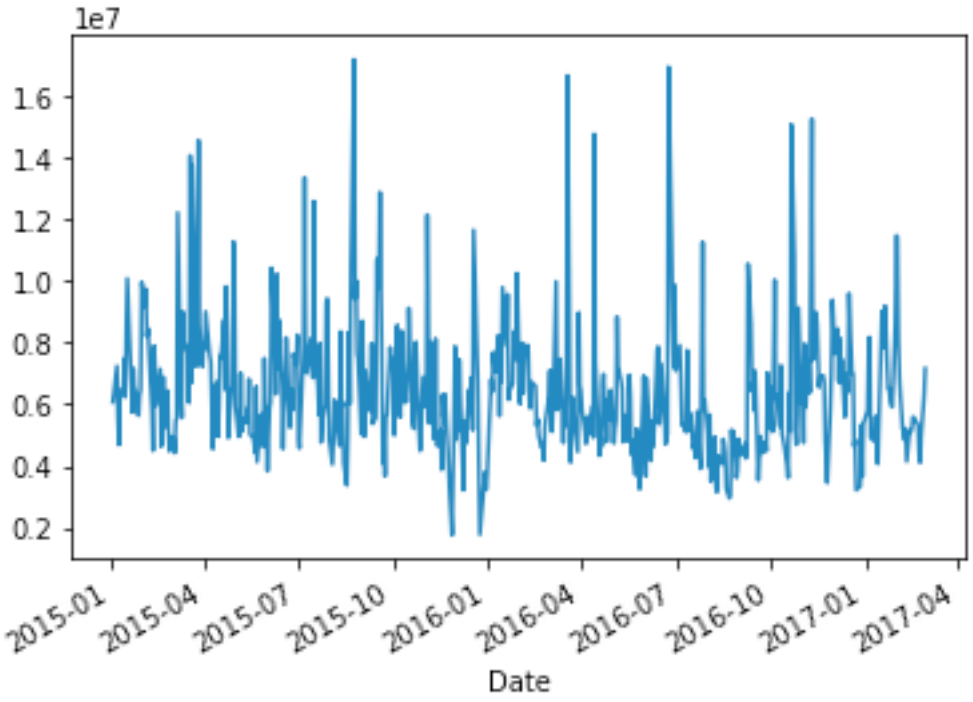
Price

Of all the different price features, the most relevant here is the closing price, as this is the price used by the agent to close the position, and will ultimately used to compute overall profit or loss. The figure below shows the closing price over the entire time period.



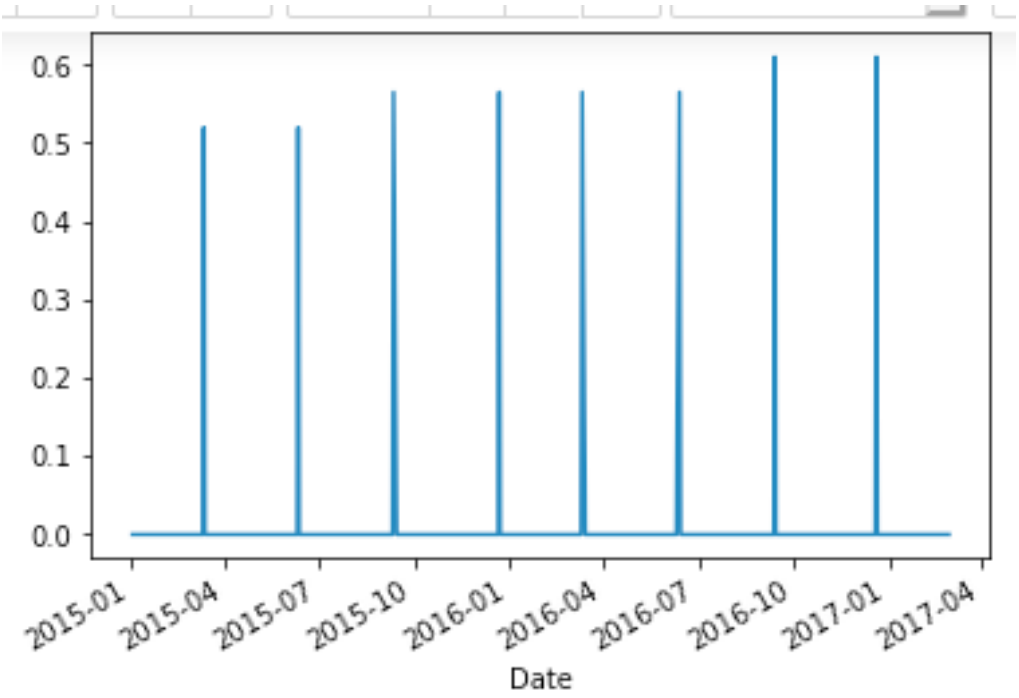
Volume

Given that the Adjusted volume is back-adjusted, it shouldn't be used as one of the features for prediction as this data would not be available if the system was to be run on live data. The actual volume over the entire period is plotted below.



Dividend

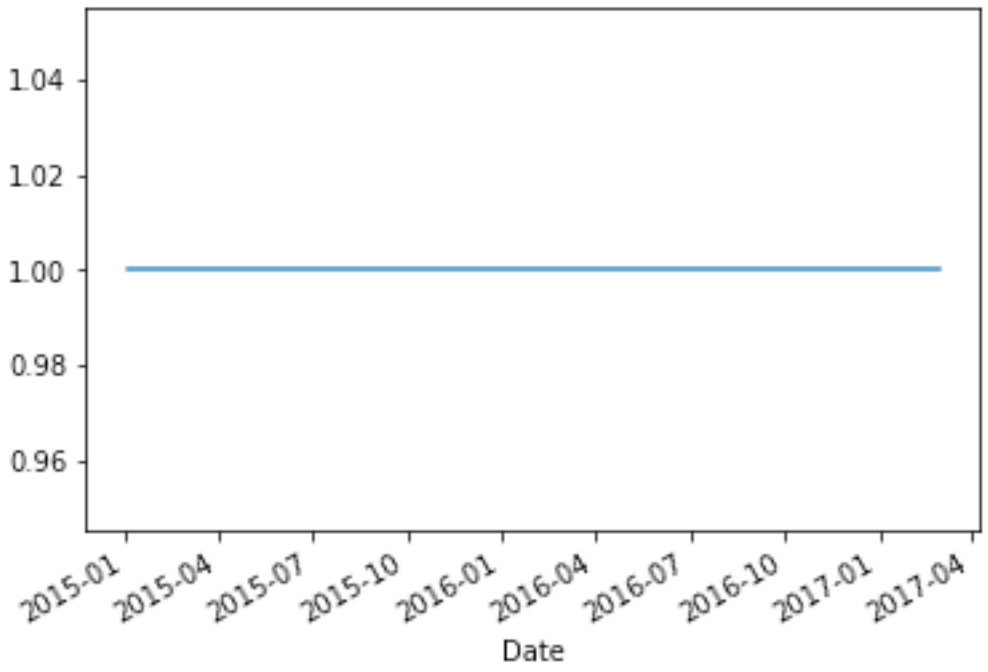
Dividend announcements can have a positive or negative effect in a stock price, so it makes sense to consider it as an option for predicting the security's price.



There are only 8 significant data points over the entire interval, which makes this feature less significant, and allows us to drop it.

Split ratio

As for the dividend data, stock splits can also have an impact on a stock's price and are worth considering.



The plot shows that there is no stock split over the entire interval, which means that this feature can be dropped as well.

Algorithm and Techniques

The algorithm I chose to implement the trading agent is the DQN, or Deep Q Network algorithm. Trading can be formulated as a Markov Decision Process problem by defining the right environment, states, actions and rewards. Typical reinforcement learning problems can be solved using a model-based or a model-free approach. Because the model-based approach requires that the agent have complete knowledge of the environment, the model-free approach will be used here. Q-Learning is a popular model-free approach to solving reinforcement learning problems. The approach consists in finding the optimal action-selection policy using a Q function. The goal is to maximize the value function Q. For a specific state action pair, the Q-value will return the expected future reward of that action at that state. The main weakness in Q-learning is its inability to estimate a value for unseen states. A DQN approach solves this problem by using neural networks to approximate the Q values.

The input of the network is the current state, and its output is an approximation of each of the Q-values for each action that the agent can take.

The Bellman Equation, which defines the Q-value at a given state S, can be expressed as follows:

$$Q(s, a) = r + \gamma \max_{a'}(Q(s', a'))$$

Where Q(s′,a′) is the Q-value of the next state, r is the reward, and γ is the discount rate.

The Outputs of the network are Q-values for each possible action. Q-values can be any real value, which makes it a regression task that can be optimized with the following squared error loss.

$$\frac{1}{2} * (r + \gamma \max_{a'}(Q(s', a')) - Q(s, a))^2$$

The Q-table update rule for Q-learning can be replaced by the following algorithm when using neural networks

1. Do a feedforward pass for the current state s to get predicted Q-values for all actions.
2. Do a feedforward pass for the next state s′ and calculate maximum over all network outputs maxa′Q(s′,a′).
3. Set Q-value target for action a to r+γmaxa′Q(s′,a′) (use the max calculated in step 2). For all other actions, set the Q-value target to the same as originally returned from step 1, making the error 0 for those outputs.
4. Update the weights using backpropagation.

To make the model's training stable, experience replay and a target network were used with this DQN algorithm.

Experience Replay

All experiences <s,a,r,s′> are stored in a replay memory. When training the network, random samples from the replay memory are used instead of using the most recent transition. This breaks the similarity of subsequent training samples, which could drive the network to a local minimum

Target Network

We create two deep networks. We use the first one to retrieve *Q* values while the second one includes all updates in the training. After a certain number of updates, we synchronize both networks. The purpose is to fix the Q-value targets temporarily to avoid having a moving target during training. This allows the model to converge faster.

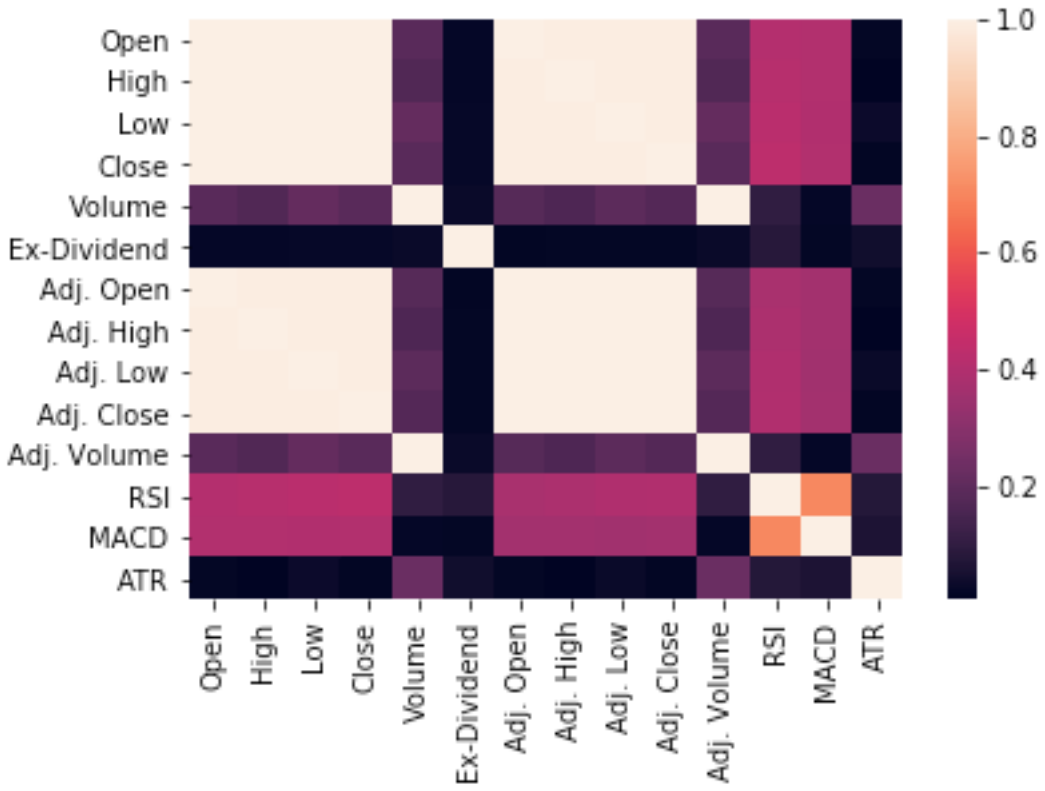
Benchmark

Two benchmark models will be used in this project. Because it's a reinforcement-learning problem the agent will be compared to an agent taking random actions at each time step. Because our problem is also a trading problem, we'll also be comparing it to an agent applying a buy and hold strategy, which buys the stock at the initial time step, and hold the position the entire time.

Methodology

Data Preprocessing

To allow the model to train faster by reducing the state size, redundant features were identified, and removed from the list of features. The following correlation matrix was used to identify the degree of correlation between the features and drop the ones that were heavily correlated with others.



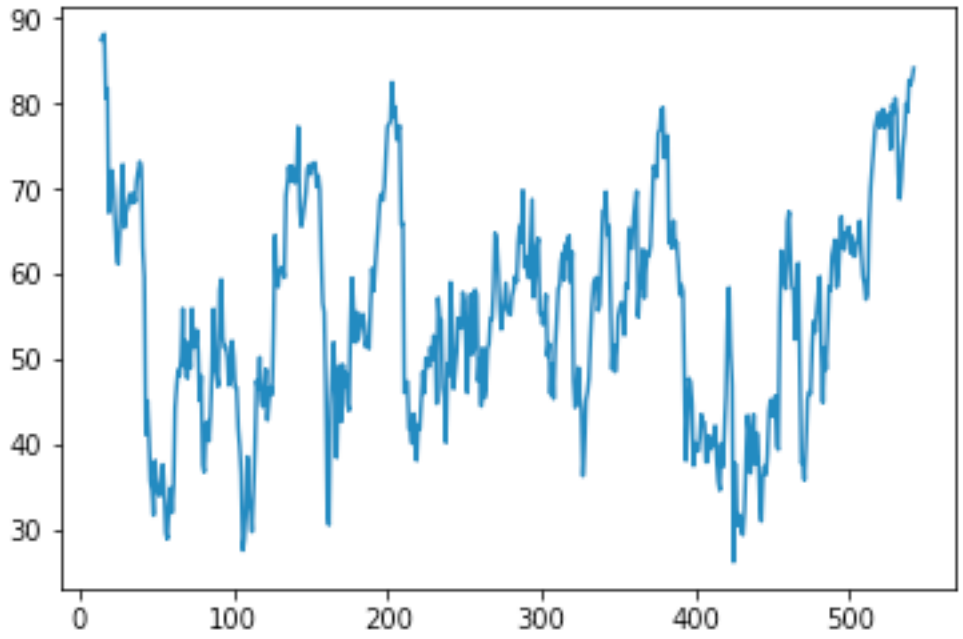
The correlation matrix shows that all the price-related features are highly correlated with each other, and the Volume and Adjusted Volume are as well. After dropping non-significant features such as the Dividend and Split Ratio earlier, we're left with the Closing Price and Volume as our features to use.

Feature engineering

In additional to the features provided in the dataset, I added additional features that are commonly used in technical analysis to determine momentum in stocks.

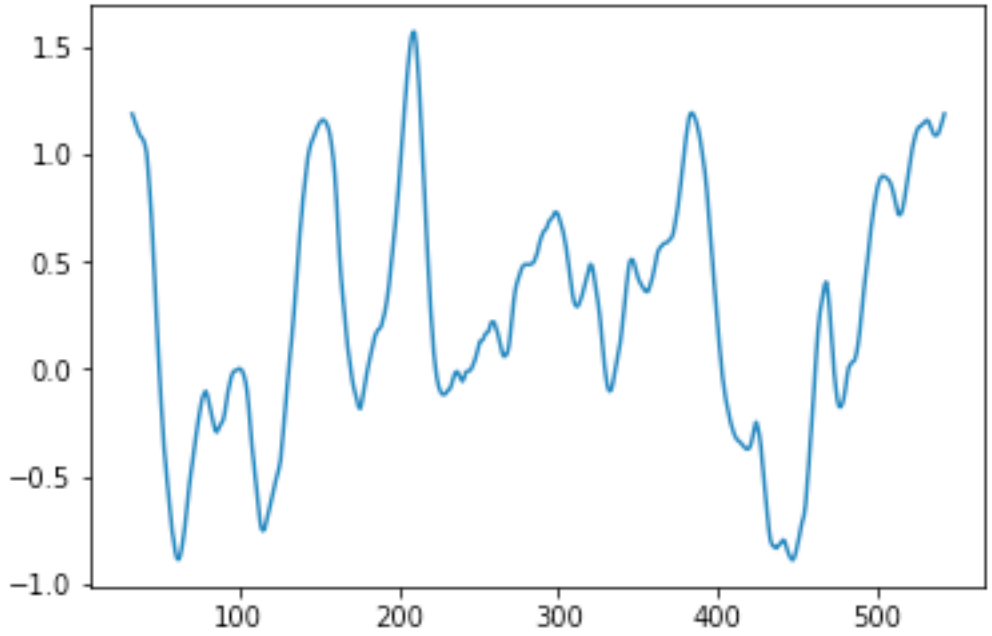
RSI

The relative strength index, or RSI is intended to chart the current and historical strength and weakness of a stock based on the closing prices over a certain trading period. The plot below shows the RSI over a 14 day period.



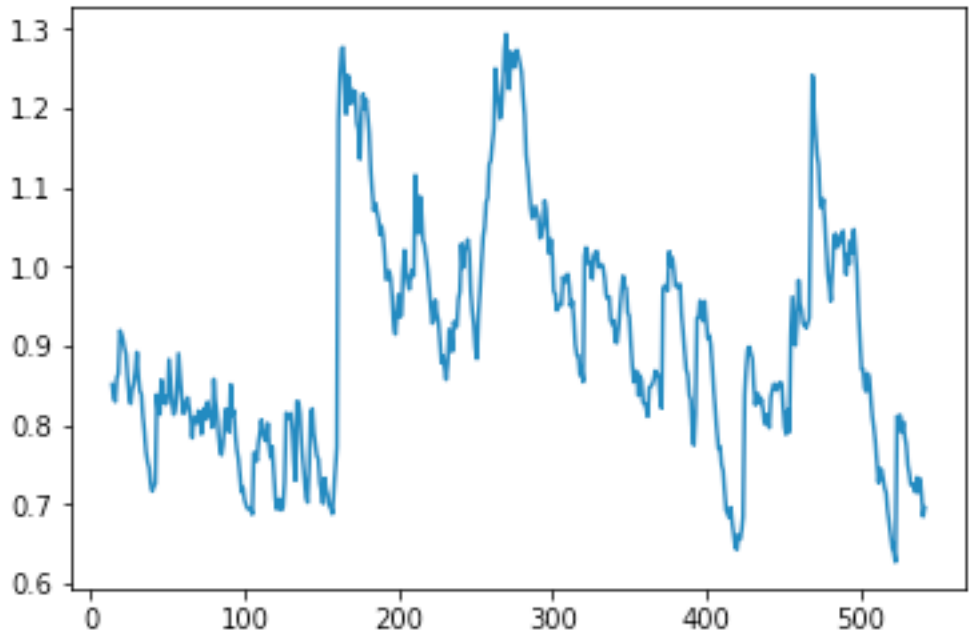
MACD

The MACD, or Moving Average Convergence Divergence is an indicator made up of three moving averages, and used to determine the momentum of a stock.



ATR

The ATR, or Average True Range, is a technical analysis indicator that measures volatility by decomposing the entire range of an asset price for that period.



Implementation

Environment

Each row of the time series dataset will be a considered a state of the environment. The updated dataset after preprocessing looks as follows:

Date	Close	Volume	RSI	MACD	ATR
2015-02-20	55.61	6882578.0	69.427910	1.189718	0.838562
2015-02-23	55.51	5265659.0	68.280760	1.168106	0.809665
2015-02-24	55.72	6443684.0	69.423324	1.146991	0.790403
2015-02-25	55.64	5125110.0	68.412332	1.123402	0.765374
2015-02-26	56.09	4479595.0	70.972991	1.104119	0.753062

The agent can take a single action each day, with the daily profit or loss computed following the action. The possible actions that the agent can take are to buy a position using all the available funds, hold that position, or sell the entire position. The daily profit or loss will be used as the reward, with the goal of maximizing the total profit.

Model

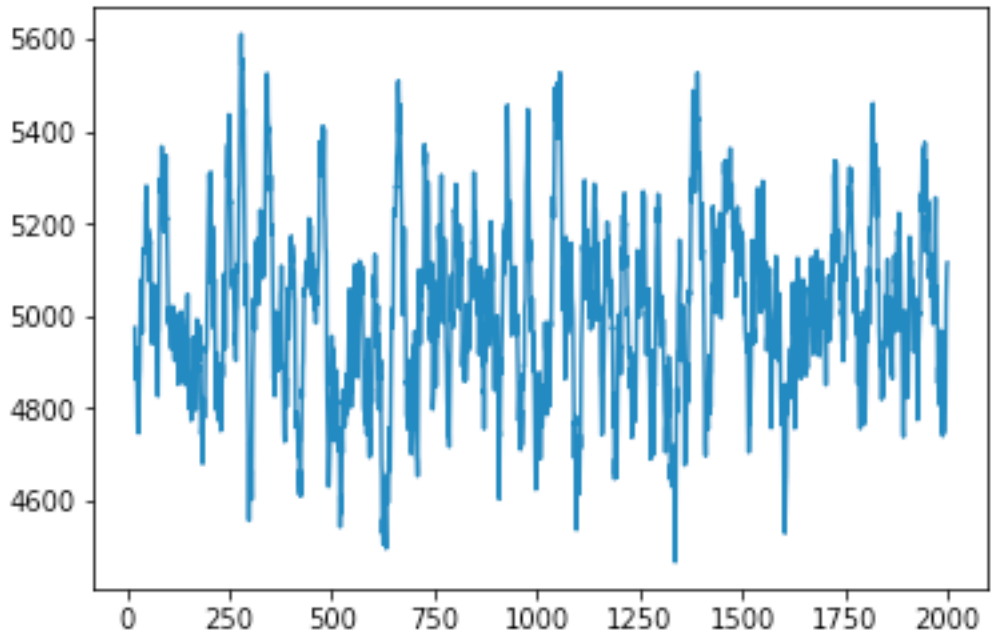
The model created for this project will be using the DQN(Deep Q Network) approach. Using this approach, two models are defined to decouple the action selection from the target Q-value generation. The first model is used to select what the best action to take for the next state is. The second model, called the target network, is used to compute the target Q value resulting from the action taken. The DQN approach with target network helps reduce the overestimation of Q-values and achieve faster convergence. The two networks will have the same architectures, with their hyperparameters selected and tuned in order to achieve stable learning.

The architecture selected for the models was a 3 layer architectures, with dense layers of 32, 32, and 64 units, with RELU activation between the layers. The output was then passed through a linear activation function. The loss function selected as expected was a Mean Squared Error function, and the optimizer was an Adam optimizer due to its ability to converge faster.

Refinement

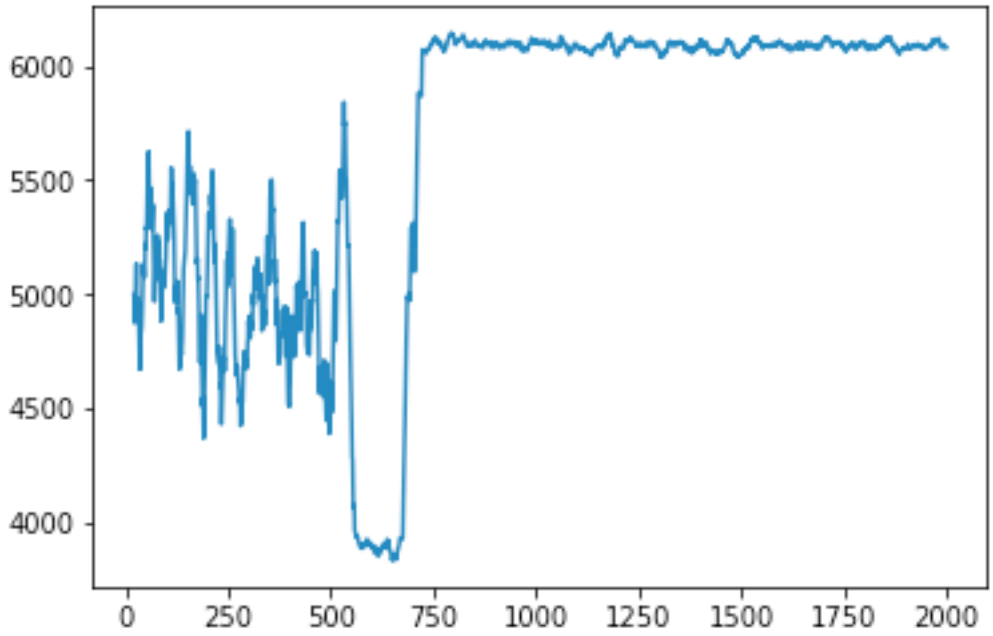
A set number of 2000 iterations were set to test various hypotheses. The initial choice for the optimizer was the SGD optimizer, which fails to converge after the maximum number of

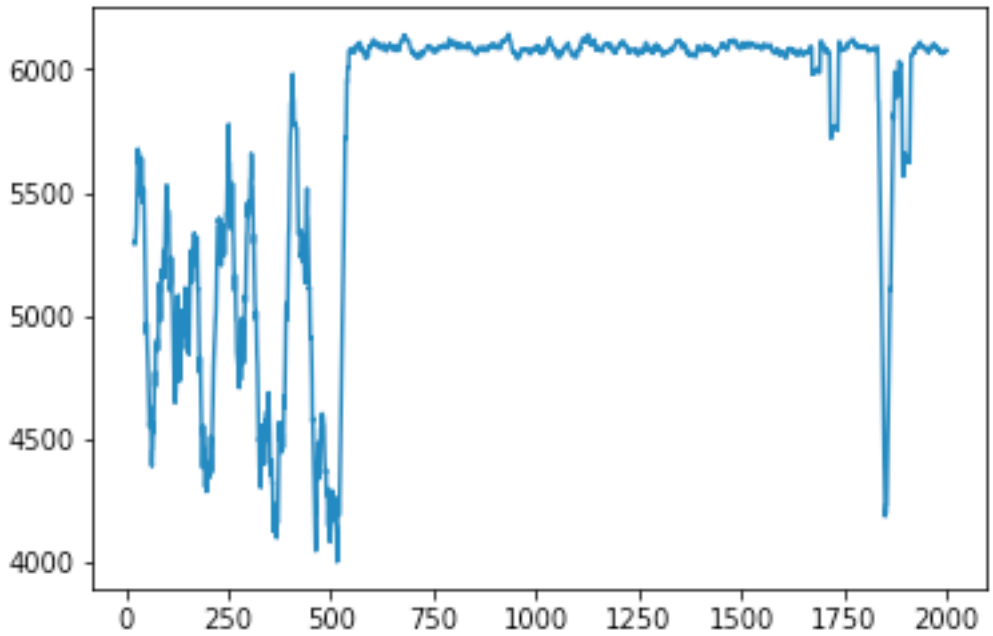
iterations for a learning rate set at 0.001. A rolling mean on a period of 20 days will be use to remove some of the price volatility. The initial results with a SGD optimizer can be showed below.

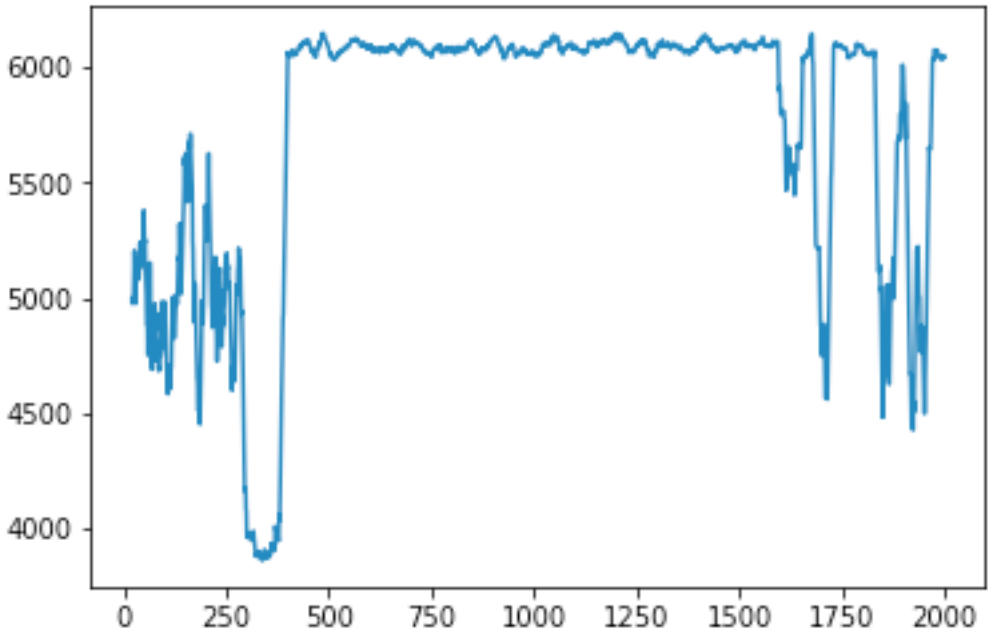


The optimizer was substituted for an Adam optimizer, which yielded much better results Different values were used for the learning rate, starting at 0.001, and increasing the value to see the impact of the learning rate on the results.

The following images show results for learning rates of 0.001, 0.003, and 0.005 respectively.

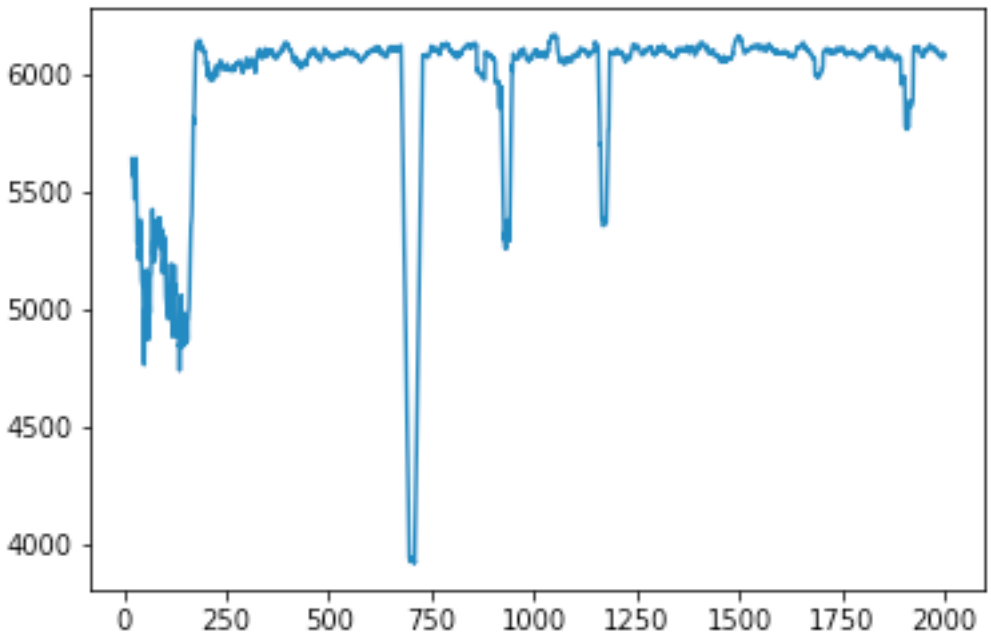

6000
5500
5000
4500
4000
0
250
500
750
1000
1250
1500
1750
2000


6000
5500
5000
4500
4000
0
250
500
750
1000
1250
1500
1750
2000

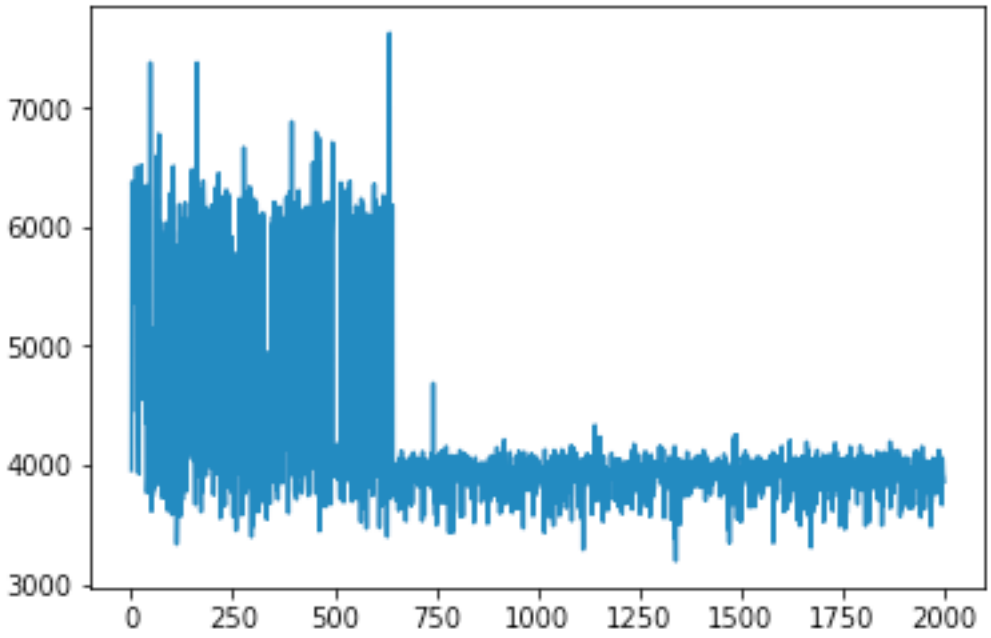


The results show that as the learning rate increases, the model converges faster, but learning is unstable the higher the learning rate is. As a result, the optimal learning rate was kept at 0.001.

The architecture was also modified to increase the number of layers and the size of the layers. This also resulted in training being less stable, probably due to overfitting by the network. The following picture shows the result of training using an architecture of 32, 64,128,256 and 512 layers.



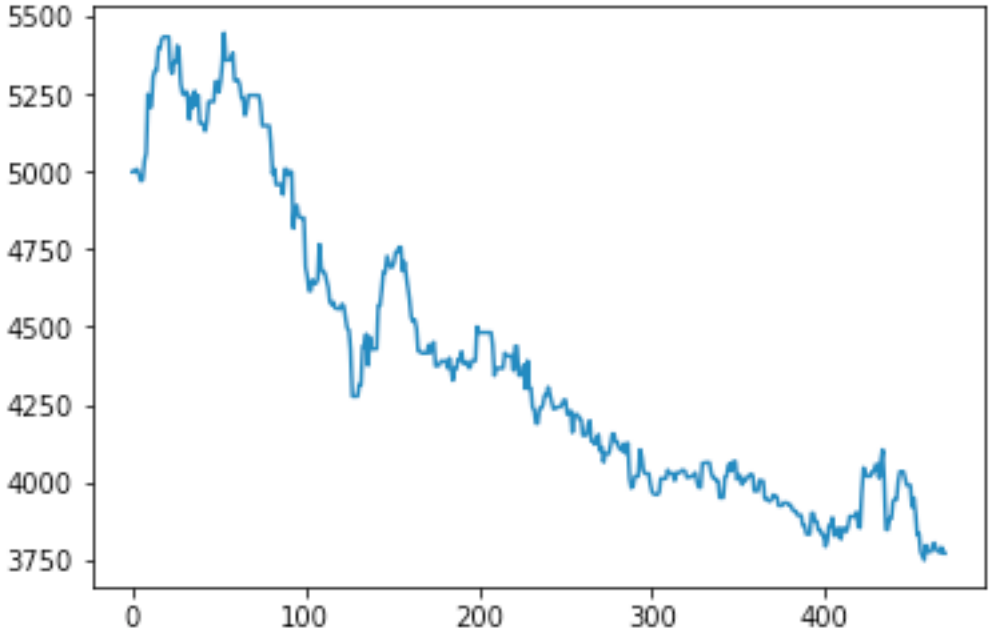
The current reward function involved using the profit/loss as the reward for each state transition. An attempt was made to clip the rewards, using -1 for losses, and 1 for profits. Even though the reward plot showed convergence and positive rewards, this did not translate to a profitable strategy, as evidenced in the plot below

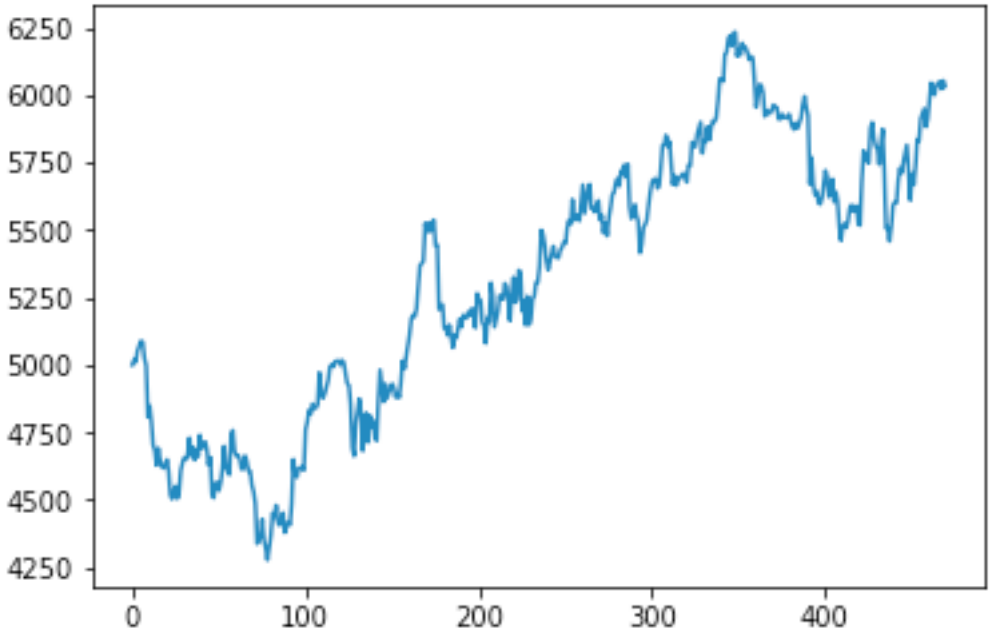


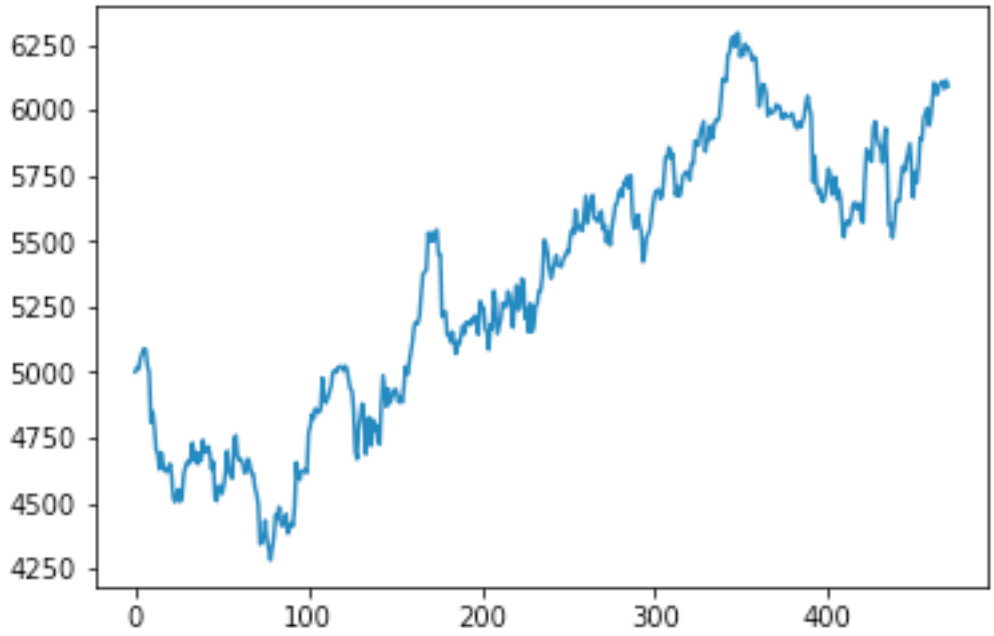
Results

Model Evaluation and Validation

The results using the optimal architecture (Adam optimizer, 3 layers, 0.001 learning rate) were compared to the two benchmark models, which are a model taking random actions, and one using a buy and hold strategy. The results for each of these agents over the training period can be seen below.


5500
5250
5000
4750
4500
4250
4000
3750
0
100
200
300
400


6250
6000
5750
5500
5250
5000
4750
4500
4250
0
100
200
300
400



Using an initial balance of 5000 dollars, the final balances are 3771 dollars, 6090 dollars, and 6032 dollars, for the random, buy and hold, and reinforcement learning agent respectively.

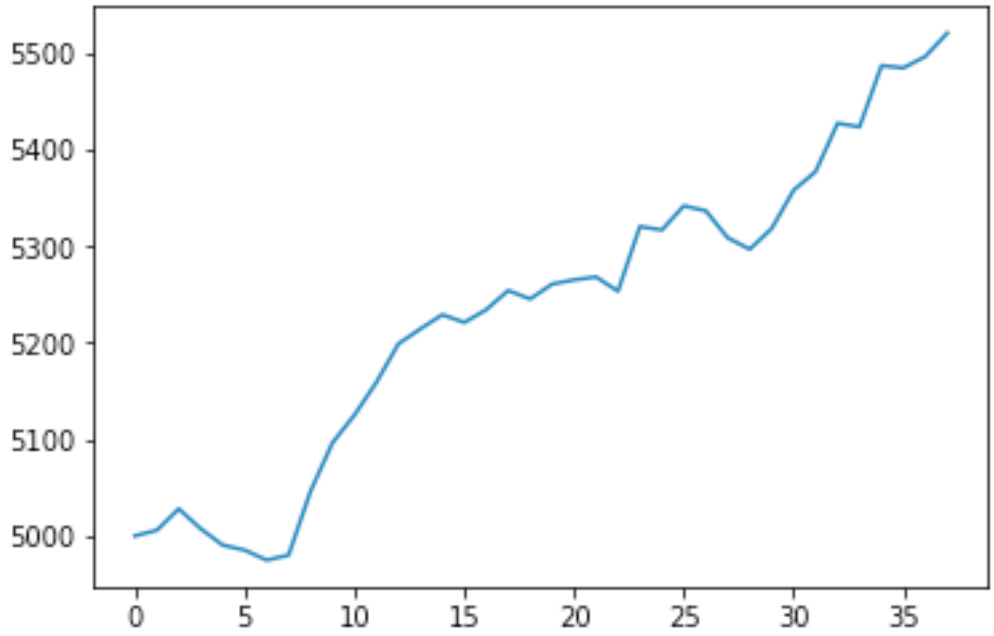
Justification

The results obtained with our trading agent are satisfactory, as they are clearly superior to random agent, and in line with the buy and hold strategy of a stock in a clear uptrend. The goal of this exercise is to be able to generate a profitable strategy, and the trading agent seems to do just that.

Conclusion

Free-Form Visualization

Using the trained model, a simulation was made to determine how the agent would perform on data that wasn't used for training. The figure below shows the agents trading for two months, again using an initial balance of 5000 dollars.



The trading strategy used seems to be profitable, with a final balance of 5521 dollars. This shows the effectiveness of the model in trending environments, assuming that the trend is continued during the testing period.

Reflection

Even though trading of a single stock was traded the entire time, the agent was tested on a different number of securities, and here are my conclusions:

-The trading agent seems to be using a very simplistic strategy

-The agent does well when trends are clearly defined

-Performance is poor when the stock is ranging.

In our current example, the trading agent follows a strategy that's very close to a buy and hold strategy, given that the stock is in an uptrend. The same hold true when the stock is in a clear downtrend, with the agent using a strategy that's equivalent to shorting the stock and then holding until the end. In that case, the agent's performance clearly beats the buy and hold strategy, which would essentially be holding on to a losing stock. A scenario that the current agent is unable to handle well is a stock having similar prices at the beginning and end of the training intervals, regardless of the price swings in between.

Improvement

The main improvement that can be made here is the implementation of a better reward function. The poor performance of the agent in

non-trending environments can be attributed to the fact that the reward function used does not steer the agent toward an optimal solution from state to state. Instead, the reward is only considered once the training is done over the entire interval, which might explain why the agent seems to stick with simpler strategies rather than more elaborate and complicated ones.

References:

Zhengyao Jiang, Dixing Xu, Jinjun Liang. A Deep Reinforcement Learning Framework for the Financial Portfolio Management Problem. arXiv:1706.10059, (2017)

Saud Almahdi, Steve Yang. An adaptive portfolio trading system: A risk-return portfolio optimization using recurrent reinforcement learning with expected maximum drawdown. Expert Systems with Applications. 87. 10.1016/j.eswa.2017.06.023. (2017)

Zhuoran Xiong, Xiao-Yang Liu, Shan Zhong, Hongyang (Bruce)Yang, Anwar Walid .
Practical Deep Reinforcement Learning Approach for Stock Trading. (2018)