Design of an Autonomous Portfolio Manager Using Q-Learning

MLND Capstone Project

Alexander Rosenthal

4/26/2018

Table of Contents

I. Definition **1**
Project Overview 1
Problem Statement 1
Metrics 2
Moving Average Convergence Divergence (MACD) 3
Bollinger Bands 3
Directional Movement Index (DMI) 4
II. Analysis **5**
Data Exploration 5
Exploratory Visualization 5
Algorithms and Techniques 8
Benchmark 11
III. Methodology **12**
Data Preprocessing 12
Implementation and Refinement 12
State Generation 12
Boltzmann Temperature (τ) Annealing 13
Reward Function 14
Hyperparameter Tuning 18
IV. Results **20**
Backtest 1 21
Backtest 2 22
Backtest 3 23
Backtest 4 24
Backtest 5 25
V. Conclusion **27**
Reflection 27
Improvement 29
APPENDIX **31**
Backtest 1 31
Backtest 2 31
Backtest 3 32
Backtest 4 32
Backtest 5 33

I. Definition

Project Overview

Many individuals invest a percentage of their pre and/or post tax income into retirement funds that they cannot access without incurring significant penalties until they reach what is considered retirement age. While it is wise to plan for retirement, restricting oneself to this type of investing inherently prevents one from becoming economically independent. In order to become economically independent, or even not fully dependent on an employer's paycheck, one must be able to withdraw investments without incurring extreme and prohibitive penalties. Unfortunately, many people fear investing in the stock market and some even view it as a form of gambling. Many people are also inexperienced with investing and would rather let an entity that understands how markets operate handle their capital for them. Unfortunately, there are significant monetary barriers for entering an asset management fund that has the potential to produce high returns and does not require one to keep their assets tied up for multiple decades.

For the average person that wants to invest in the stock market and be able to withdraw capital with less restrictions than a retirement fund without worrying about the intricacies of managing their portfolio it would be necessary to reduce the barrier to entry of professional fund management. An autonomously managed portfolio may be able to offer the general public this luxury.

The algorithm developed for this project will begin with $10,000 at its disposal and will learn to manage a portfolio consisting of OFIX, TXT, and ROIC by placing trades with these securities and receiving rewards based on the impact these trades have on the performance of the portfolio. Each day the algorithm will be presented with a state for each security that is composed of various conditions for specific technical indicators that are derived from historical pricing data. Based on these states, the algorithm will decide whether it should buy 40 shares of a security, sell all shares of a security, or do nothing. Buying shares is not done on margin and therefore there must be enough cash in the account to cover the cost of a purchase. Selling short is not supported and therefore selling a security can only occur if shares of the security are owned.

Problem Statement

The algorithm developed for this project seeks to show proof of concept for a model free autonomous portfolio manager that trades based on the outcomes and predicted future impacts of its previous actions. Q-learning was implemented to allow the algorithm to learn from the actions it selects in an increasingly less random fashion until it has learned the appropriate action to select given an arbitrary state.

Quantopian was used as the development platform for this project. Quantopian offers access to extensive historical pricing data, a built in backtester, and built in functions specifically for market analysis.

During a backtest, technical indicators are calculated from historical pricing data and a state for each security is fed to the Q-learner's action selection function that determines whether it will buy shares in a security, sell its holdings in a security, or hold a security. Throughout the learning phase, the selected action is appropriately rewarded based on the impact it has on the portfolio. After learning is complete, the Q-learner immediately begins taking actions based on the Q-table it has built.

For each backtest detailed in this project, the performance of the autonomously managed portfolio is plotted along with the benchmark, SPY. SPY is considered an industry standard for benchmarking trading strategies and therefore this project seeks to develop an algorithm that can match SPY's performance during the testing phase.

Metrics

The returns of SPY and the autonomously managed portfolio will be plotted together over the course of the entire backtest as well as over the course of the testing phase for 5 representative backtests. It will be evident from these plots how the algorithm's performance compares to that of the benchmark. Accompanying each figure in the "Results" section will be a description explaining the plotted results.

The returns of the entire backtest are retrievable from Quantopian; however, Quantopian does not support recalculating the returns based on a different start date. It was therefore necessary to derive an expression that could determine returns assuming a start date later than the backtest start date. This derivation can be found below. Note that the equation for percent change and returns is the same.

Given an initial value x_i and a final value x_f it is known that the equation to determine the percent change or in the case of stocks the return is:

$$\frac{x_f - x_i}{x_i} = \frac{x_f}{x_i} - 1$$

Let the array X represent an array of values for which the percent change needs to be determined and the array Y represent the corresponding values for the percent change such that:

$$X = [x_0, \ldots, x_i, \ldots, x_n]$$

$$Y = [y_0, \ldots, y_i, \ldots, y_n]$$

where

$$y_n = \frac{x_n}{x_0} - 1$$

Suppose Y is given and X is unknown yet it is necessary to find Z where

$$Z = [z_i, \dots, z_n]$$

and

$$z_n = \frac{x_n}{x_i} - 1$$

It is therefore necessary to find an expression for $\frac{x_n}{x_i}$ using only the values found in Y.

$$\frac{y_n + 1}{y_i + 1} = \frac{x_n}{x_i}$$

Therefore

$$z_n = \frac{y_n + 1}{y_i + 1} - 1$$

Equation 1

Equation 1 made it possible to find the returns or percent change of the portfolio starting from the first day of testing. Therefore, it was possible to compare the performance of the autonomous portfolio manager with the benchmark during the testing phase regardless of the results of the full backtest.

Various technical indicators are necessary for the algorithm to build the states that the Q-learner will visit. While these indicators are not used in evaluating the performance of the algorithm, their importance warrants an explanation of how they are generated and applied. Therefore, descriptions of each technical indicator used to build the Q-learner's states can be found below. The structure of every possible state can be found in the "Algorithms and Techniques" section as Equation 3.

Moving Average Convergence Divergence (MACD)
The MACD is calculated by subtracting the 26-day exponential moving average (EMA) from the 12-day EMA. Generally, the MACD along with a 9-day EMA signal line are plotted and interpreted by a trader seeking to gain insight regarding the movement of a security. An EMA is a moving average that places more emphasis on the most recent data. In order to reduce the number of possible states while maintaining a valid signal, the algorithm determines whether the MACD minus the signal line is greater or less than 0. The following example was used as inspiration for this method, https://www.quantopian.com/help#macd.

Bollinger Bands
This indicator consists of three lines, or bands. The middle band is the N-period simple moving average (SMA), the upper band is K standard deviations above the middle band, and the lower band is K standard deviations below the middle band. For this project, the 5 day SMA is used along with bands 1.9 standard deviations above and below the SMA. The algorithm then determines whether the current price is above the upper band or below the lower band.

Directional Movement Index (DMI)
This indicator consists of three lines, the Average Directional Index (ADX), the Plus Directional Indicator (DI+), and the Minus Directional Indicator (DI-). The DI+ and DI- measure the direction of a trend and the ADX measures the strength of the trend. The algorithm determines whether the ADX is greater than 26 and whether the DI+ is greater than the DI-. More information regarding the DMI can be found at Investopedia and StockCharts.

II. Analysis

Data Exploration

The only data necessary for this project is the daily pricing data for each security in the portfolio (TXT, ROIC, and OFIX) and the benchmark (SPY). This data consists of the open, high, low, and close prices for each day of the backtest from 7/29/2013 – 3/12/2018, which is 1164 data points for each field for each security. The testing phase is from 7/20/2016 – 3/12/2018 and therefore the Q-learner is trained on 751 data points per security and tested on 414 data points per security. Since daily data is used for this project, the close price is considered the current price for any arbitrary day during a backtest as this is the convention used by Quantopian. This data is used to derive the technical indicators discussed in the “Metrics” section that are then used by the algorithm to generate the states for the Q-learner. The only fields required from a pricing pull are ‘high,’ ‘low,’ and ‘close_price.’ The ‘high’ and ‘low’ fields are only necessary for calculating the DMI, and the ‘close_price’ field is required for all the technical indicators. The table below shows an example of a full pricing pull from which the necessary fields are extracted. Notice how ‘low’ and ‘price’ are equal, which is consistent with what was previously mentioned. Plots of the percent change of the pricing data from the launch date of the algorithm for each security and the benchmark can be found below in the “Exploratory Visualization” section.

		open_price	**high**	**low**	**close_price**	**volume**	**price**
major	**minor**						
2013-07-29 00:00:00+00:00	Equity(8554 [SPY])	154.078	154.425	153.557	153.968	67089105	153.968
	Equity(7674 [TXT])	27.117	27.408	27.117	27.398	1503169	27.398
	Equity(34972 [ROIC])	11.843	11.888	11.599	11.725	211060	11.725
	Equity(5601 [OFIX])	28.05	28.05	27.22	27.36	152112	27.36
2013-07-30 00:00:00+00:00	Equity(8554 [SPY])	154.461	154.626	153.63	153.991	73753408	153.991
	Equity(7674 [TXT])	27.507	27.606	27.279	27.437	2124403	27.437
	Equity(34972 [ROIC])	11.775	11.801	11.583	11.641	275788	11.641
	Equity(5601 [OFIX])	23.22	23.39	21.83	22.955	1305655	22.955

Exploratory Visualization

The autonomous portfolio manager uses historical pricing data for the securities in the portfolio to calculate technical indicators and ultimately make decisions based on the state of the indicators. The performance of the portfolio manager is measured by its returns, which is the percent change in the portfolio between an initial and final date. The percent change or returns of all securities in the portfolio as well as the benchmark, SPY for the entire length of a backtest is plotted below.

Figure 1 is a plot of the percent change in pricing data for all securities including SPY. Figures 2-5 are plots for SPY and each individual security (TXT, ROIC, and OFIX) along with 1st and 3rd order lines of best fit. It is clear from these plots that SPY deviates the least from its trendline, whereas the other securities deviate quite a bit. These securities were intentionally selected for that very reason in the hope that this would provide the learning algorithm more opportunities to learn when to make an optimal trade. All securities, except for ROIC, exhibit overall positive trends. ROIC initially trends positively but begins to show a strong negative trend shortly after June 2016 that continues until the end of the simulation. This trend is made more apparent by the 3rd order fit. If the algorithm manages to maintain a positive trend during this time that will help show that it has learned an appropriate policy for trading.

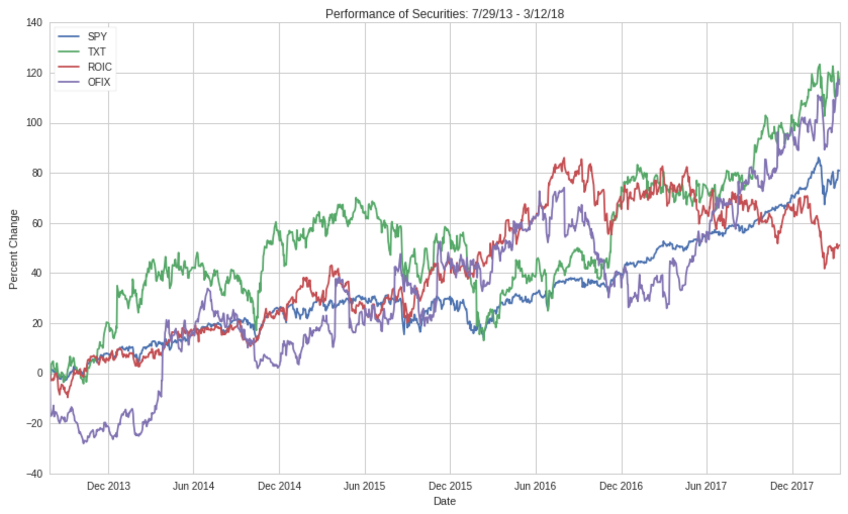

Figure 1

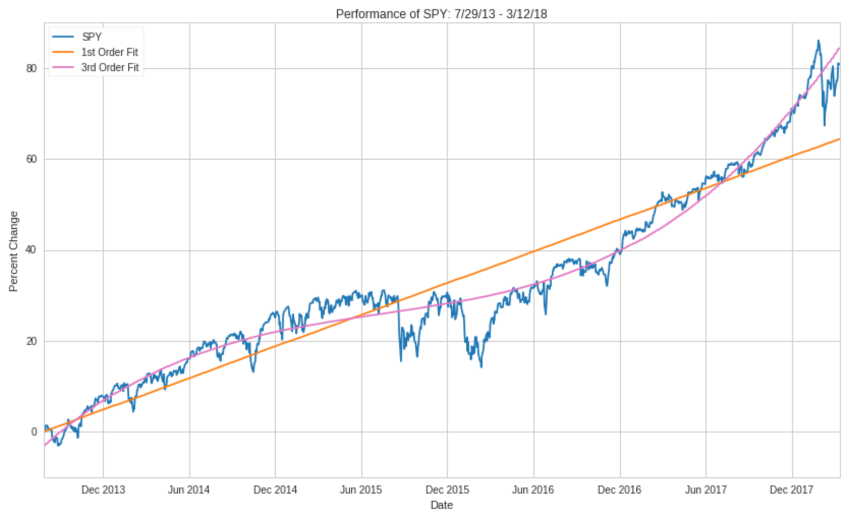

Figure 2

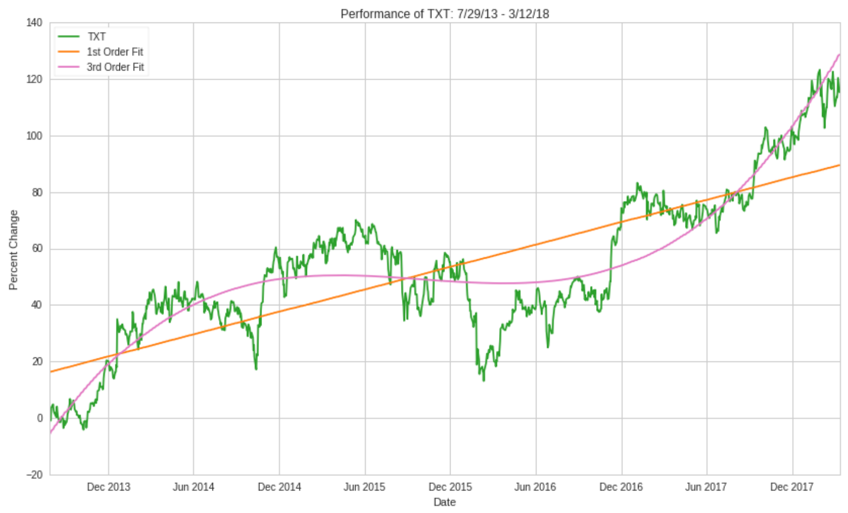

Figure 3

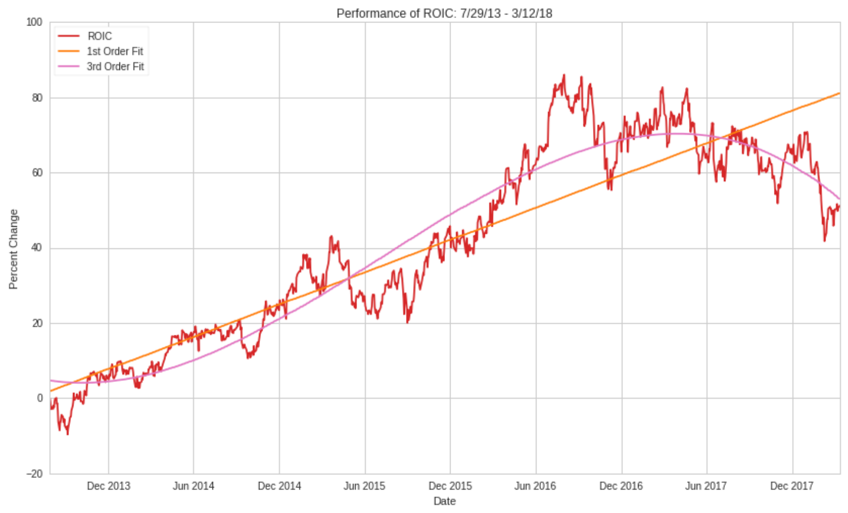

Figure 4

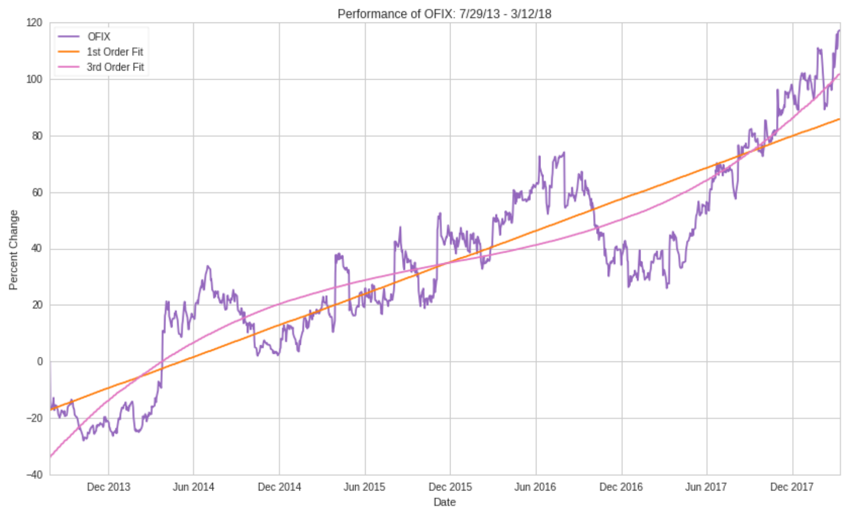

Figure 5

Algorithms and Techniques

The autonomous portfolio manager implements Q-learning in an attempt to learn and eventually select actions most likely to generate high portfolio returns. As described in the Reinforcement Learning section of this course, Q-learning functions by populating a Q-table with all available state-action pairs for the state being visited. An action is then selected based on the action selection function and the Q-table is appropriately updated with the reward earned by the action based on the learning function.

Q-learning was selected to drive the learning process of the algorithm because it is a model free approach that learns from the actions it selects. Eventually after sufficiently observing and interacting with the environment the Q-learner is generally able to develop optimal policies for the states it can expect to visit. This method is suited to the mechanics and stochasticity of the stock market and will allow the algorithm to learn trading signals by interacting with the market.

Equation 2 shows an example Q-table entry:

```
Q[example_entry] = {state: {‘buy’:buy_reward, ‘sell’:sell_reward,
                    ‘hold’:hold_reward}}
```

Equation 2

For this project, each state is a tuple of Booleans where each Boolean corresponds to whether an industry accepted technical indicator meets a specific condition. Various combinations of technical indicators and their conditions were tested. A state composed of conditions for the MACD, Bollinger Bands, and DMI generated the best results. Equation 3 is the pseudocode for every possible state for any asset:

```
state[asset] = (
          MACD[asset] > 0, closing_price[asset] < lower_BBand[asset],
          closing_price[asset] > upper_BBand[asset], ADX[asset] > 26,
          DI+[asset] > DI-[asset]
)
```

Equation 3

Action selection is done using a Boltzmann distribution so that the Q-learner can better adapt to the stochasticity of the stock market and can hopefully over time select the most optimal action for any given state. The Boltzmann distribution that gives the probability of selecting an action for a given state is defined in Equation 4 below:

$$P(a) = \frac{e^{Q[s,a]/\tau}}{\sum_a e^{Q[s,a]/\tau}}$$

Equation 4

In Equation 4, Q[s,a] represents the reward assigned to a particular action given the current state and τ represents the Boltzmann temperature. The higher the value for τ, the more randomly actions will be selected. Therefore, a higher value for τ promotes exploration and a lower value promotes exploitation. As τ approaches 0 the action with the highest reward is more likely to be selected. The annealing function that controls τ will be discussed further in the "Implementation" section.

The first time a state is visited, the reward for each possible action is initialized to 0 and therefore an action is randomly selected. The Q-value for that state is then updated based on the reward earned for that action. The reward values are based on the performance and estimated future performance of the security being analyzed. Future performance of the security is predicted by applying linear regression to the price of a security for the 3 most recent days. The reward function will be discussed in greater detail in the "Implementation" section. The Q-value is updated based on a slightly modified version of the standard Q-learning update rule, which can be found below.

Standard Q-learning update rule:

$$Q(s_t, a_t) \leftarrow (1 - \alpha) * Q(s_t, a_t) + \alpha * (r_t + \gamma * \max_a Q(s_{t+1}, a))$$

Equation 5

The modified Q-learning update rule used in the algorithm allows gamma (γ) to be tuned separately for each possible action that the Q-learner can take. For this algorithm gamma also impacts the importance of the reward assigned for the immediate impact of an action. See Equation 7.

The algorithm contains many parameters that required tuning. Below is a list of tuneable parameters with brief descriptions.

Tuneable parameters found in `def reward(context):`

`gamma:`
Controls emphasis of reward for predicted future performance and current impact, a smaller number places more emphasis on future predictions and less emphasis on the current impact (see Equation 7). Values were tuned separately for each action and the values arrived at are found below:

```
'buy' – gamma = 0.7
'sell' – gamma = 0.4
'hold' – gamma = 0.7
```

Tuneable parameters found in `def initialize(context):`

`context.assets:`
Controls the assets to be traded. Currently consists of TXT, ROIC, and OFIX. Changing this has the potential to drastically affect performance as the parameters were tuned based on the performance of trading these securities. Therefore, changing the assets may require retuning many parameters.

`context.alpha:`
Value is set to 0.42. Controls balance of importance between the new reward and current reward.

`context.hold_limit:`
Value is set to 6. Limits the number of consecutive `'hold'` actions regardless of the reward for that value. Necessary to help prevent the algorithm from holding a security for too long. Holding a security is necessary but holding for too long can cause the returns to stagnate.

`context.buy_num_shares:`
Value is set to 40. The number of shares the algorithm should buy when a `'buy'` action is initiated.

`context.temp:`
Value is set to 90. Initial Boltzmann temperature for the step portion of the annealing function.

`context.init_step_delta_t:`
Value is set to 40. The number of time steps, in days, to wait prior to decreasing `context.temp`. The annealing function is in two parts, a decreasing step function and a decay function.

`context.step_tol:`
Value is set to 2. The tolerance of the step annealing function.

`context.step_decay:`
Value is set to 0.5. The value by which the Boltzmann temperature is multiplied at each step during the step annealing function.

`context.step_len_mult:`
Value is set to 1.2. The value by which the step length is multiplied for each successive step.

`context.delay_decay_count:`
Value is set to 140. Number of days between the step and decay portions of the annealing function.

`context.init_decay_temp:`
Value is set to 30. Initial Boltzmann temperature during the decay portion of the annealing function.

`context.decay_rate:`
Value is set to 0.98. The value by which Boltzmann temperature is multiplied during the decay portion of the annealing learning.

`context.tol:`
Value is set to 0.5. Tolerance value for the decay portion of the annealing function.

Benchmark

As stated in the Capstone Proposal:

> *"The performance of the autonomous portfolio manager will be compared to the performance of the securities in the portfolio as well as the S&P 500 ETF, SPY. SPY is generally considered a standard for benchmarking trading strategies. If a strategy beats the performance of SPY it can be regarded as capable of beating the market."*

The "Results" section of this project contains figures that plot the performance of the algorithm together with the performance of SPY throughout the testing phase, which happens between 7/20/2016 – 3/12/2018. Over this period SPY realizes a return of 32.2%. It is therefore considered a success if the algorithm realizes similar returns over this period while maintaining a generally constant growth rate that does not exhibit the sporadic nature of the securities that make up the portfolio, which can be seen in Figures 3-5.

III. Methodology

Data Preprocessing

All necessary data preprocessing is handled by Quantopian. From the Quantopian help pages under “Data Sources”:

> *“When your algorithm calls for historical equity price or volume data, it is adjusted for splits, mergers, and dividends as of the current simulation date. In other words, if your algorithm asks for a historical window of prices, and there is a split in the middle of that window, the first part of that window will be adjusted for the split. This adjustment is done so that your algorithm can do meaningful calculations using the values in the window.”*

Implementation and Refinement

Each day the algorithm pulls 40-day historical pricing data that is used to generate the MACD, Bollinger Bands, and the DMI for each security in the portfolio (TXT, ROIC, and OFIX). These technical indicators are generated using `talib`, a technical analysis package that is available on Quantopian. The technical indicators are then analyzed and converted into a state for each security that is stored in a dictionary where the keys are the security and the values are the states. Each state is then looked up in the Q-table. If a state is not present it is added to the Q-table and the reward for each available action (‘buy’, ‘sell’, ‘hold’) is initialized to 0. The states for each security are then passed to the action selection function. A random number is generated for each security that is used to select an action, ‘a,’ with probability $P(a)$ as defined by Equation 4 in the “Algorithms and Techniques” section. It is clear from this equation that $P(a)$ is highly dependent upon τ, which controls the tradeoff between exploration and exploitation. Following action selection, a reward is generated based on the current and predicted future performance of the algorithm that is then used to update the Q-table.

Designing an algorithm that could successfully match and occasionally beat the benchmark during the testing phase was quite the challenge as there were many interconnected parts (some of which were not as obvious as others) that had to be finely tuned and selected in order for everything to function properly. Descriptions on state generation, the annealing of the Boltzmann temperature, the reward function, and hyperparameter tuning can be found below.

State Generation
Initially each state was comprised solely of conditions for Bollinger Bands where N = 20, and K = 2. This state structure did not provide the Q-learner with sufficient data to develop an optimal policy. After much testing a state comprised of conditions for the relative strength index (RSI) and DMI was found to generate desirable results; however, while optimizing other parameters and fine tuning the reward structure and the annealing function for the Boltzmann temperature it

was found that this state structure could not reliably produce the desired results. Bollinger Bands were therefore added back to the state structure, which produced suboptimal results fairly regularly. Initially, it seemed that adding the conditions for Bollinger Bands to the state structure was the issue, but this did not seem logical since creating a similar algorithm using Bollinger Bands is the final project for the Machine Learning for Trading course on Udacity. At this point the algorithm was trading 3 securities whereas the final project for the trading course only trades IBM over a predefined timeframe. Therefore, it was assumed that a relatively ideal timeframe for IBM had been selected for the final project of the trading course, or at least one where a Q-learner could extract relevant signals from Bollinger Bands. The Bollinger Bands were then removed from the state for an extended period. After further tuning of the reward function, annealing function, and other parameters it became clear that the states needed to present the Q-learner with more information. Rather than use an untested technical indicator, Bollinger Bands were revisited as a candidate to add to the state structure. Bollinger Bands with N = 5 and K = 1.9 were tested with combinations of the MACD, RSI, and DMI. Finally, the state described by Equation 3 in the "Algorithms and Techniques" section was arrived upon.

Boltzmann Temperature (τ) Annealing

Developing a method for controlling the annealing of the Boltzmann temperature such that the Q-learner was able to explore and interact with the state space sufficiently yet behave properly during the testing phase was integral to the proper performance of the algorithm and was one of the more time-consuming tasks of this project. Initially a standard decay function of the form found below was used to anneal the temperature.

```
context.temp*=context.decay_rate
```

Equation 6

This was found to only produce optimal results occasionally and required a relatively high initial temperature. It was then decided to include a second learning phase where the Boltzmann temperature was reheated and annealed after a predefined number of days had elapsed since the previous learning phase. The second learning phase used the same annealing function found in Equation 6. Better results were achieved with this structure; however, the initial temperature was still relatively high and the results seemed to be fairly dependent on how the algorithm performed shortly after deployment. Upon careful and prolonged observation of the behavior of the algorithm and how it reacted to changes in the parameters of the annealing function, a modified approach came to mind. This approach allows the Boltzmann temperature to remain constant for several days prior to cooling to a lower temperature and was partially inspired by the effects time and temperature have on the resulting microstructure of cooling steel. This method allows the Q-learner to explore more states at each temperature with the reasoning being that it will be able to more effectively build an optimal policy by being given the opportunity to select actions for multiple states at each temperature, or level of randomness. A detailed description of the annealing function along with a figure plotting the Boltzmann temperature for the duration of the training and testing phases can be found below.

The initial temperature value is set to 90. The temperature is then annealed to 2.81 in a way such that lower temperatures are held for progressively longer periods of time. The initial temperature is held for 40-time steps and each successive temperature is held for approximately 1.2 times as many time steps as the previous temperature and is 50% of the previous temperature. It is approximately 1.2 times as many time steps because the value is rounded to the nearest whole number. After the first annealing process the temperature is held constant at 1 for 140 timesteps where no learning occurs. The temperature is then reheated to 30 and annealed to 0.5 using a decay function where the current temperature is the previous temperature multiplied by 0.98. The algorithm completes the learning process on 7/20/2016. The annealing function that controls the Boltzmann temperature is plotted in Figure 6 below.

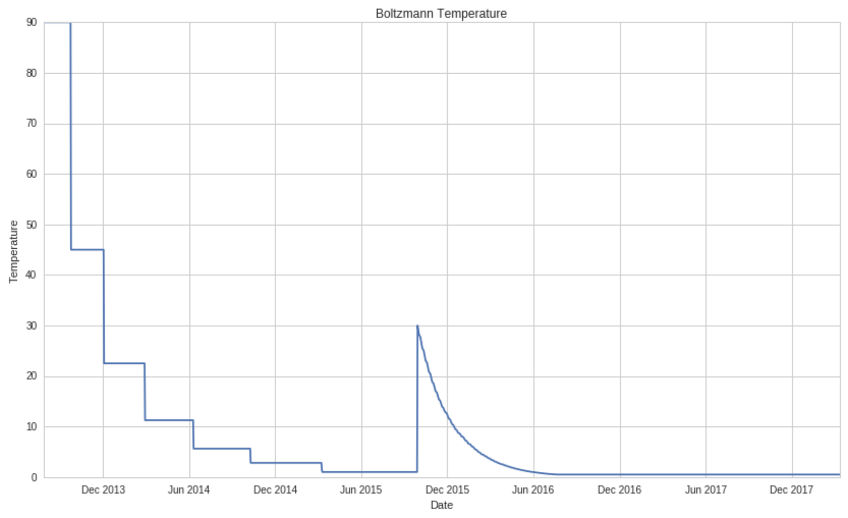

Figure 6

Reward Function

The reward function was the most difficult and arguably most important component to develop for this algorithm. In order to determine whether the algorithm was functioning properly, initially an extremely basic reward function was used as a placeholder. This function assigned a reward of 10 to the buy and hold actions if the price of the security rose after purchase, and a reward of 10 to the sell action if the price fell after a sale. Opposite events for these actions incurred a reward of -10.

After determining that the algorithm was functioning properly, the first attempt at a possible reward function was made. Despite occasionally generating successful results, this reward structure was also fairly basic, was not very uniform in how the three separate actions were rewarded, and was by nature more of a guesstimate than the final reward function. For this reward function, the buy action was rewarded by multiplying the difference between the price on the day after purchase and the purchase price by a constant that was determined through trial and error, the sell action was rewarded by multiplying the percent change between the initial cash in the portfolio and the cash in the portfolio upon sale of the security by a constant that was determined through trial and error, and the hold action was rewarded based on how long a security was held and the price movement after the hold action was selected.

The hold reward was the most difficult reward to develop as the algorithm frequently learned to hold a security for far too long. It was eventually realized that the length of time that a security is held for should be controlled by the probability function rather than penalizing the algorithm for holding a security for too long as this did not provide accurate hold rewards for a state in which the hold action was selected. After many iterations of improving upon the entire reward function the `context.hold_limit` parameter described in the “Algorithms and Techniques” section was implemented. This parameter sets the number of consecutive days a security can be held before the probability of selecting the hold action for that security is set to 0. A full description of the final reward function can be found below.

The final reward structure for each action is the sum of two separate rewards, a reward for the immediate impact of the selected action (`r2` in the code) and a reward for the predicted future performance of the portfolio given the selected action (`r1` in the code). The structure for the total reward of any action can be found in Equation 7.

```
context.reward[asset] = (1-gamma)*r1 + gamma*r2
```

Equation 7

Where `gamma` controls the importance of the two rewards. As described in the “Algorithms and Techniques” section, `gamma` can be adjusted for each action.

The addition of `r1` to the reward structure was made in an effort to give the Q-learner a form of insight into possible price movements for the securities in the portfolio by rewarding it for possible advantageous price movements given the selected action for a particular state. In order to predict the near future price movement of a security, an ordinary least squares regression is performed on the three most recent prices of the security using the OLS class from the statsmodels module. The slope is then determined to be positive or negative and `r1` is assigned based on the selected action.

Equation 8 is the `r1` assignment method for a buy or hold action:

```
if slope > 0:
    r1 = 10
elif slope <= 0:
    r1 = -10
```

Equation 8

Where `slope` is the slope of the line of best fit. When purchasing or holding a security, the algorithm should be looking for positive price movement to help ensure that purchasing and holding is done only while the security is increasing in value. The algorithm should avoid purchasing or holding a security if it is predicted to decrease in value.

Equation 9 is the `r1` assignment method for a sell action:

```
if slope <= 0:
    r1 = 10
elif slope > 0:
    r1 = -10
```

Equation 9

The algorithm should sell a security if it is predicted to decrease in value in order to ensure that a profit is realized, or to ensure that losses are minimized.

The assignment methods for `r2` are more intricate and evolved from preliminary reward structures discussed earlier. These preliminary reward structures were not standardized and did not limit the maximum possible reward as they essentially just multiplied pricing differences by a constant. In an attempt to standardize the reward structure, a reward based on the percent change between the volume weighted average price paid per share, or the cost basis and the price immediately following an action was implemented for buy and sell actions. This provided upper and lower reward limits but were unable to provide a very strong action signal as the rewards were assigned in a linear manner. A reward structure that tapered toward the desired maximum or minimum reward as it approached specific values was therefore necessary. Passing the percent change of a buy or sell action and the profit and loss (PnL) realized for a hold action through finely tuned logistic functions resolved this issue. Equation 10 is the logistic function and Equation 11 shows how this was coded.

$$f(x) = \frac{L}{1 + e^{-k(x-x_0)}}$$

Equation 10

```
def sigmoid(L,k,x0,x):
    return (L/(1+math.exp(-k*(x-x0)))) – L/2
```

Equation 11

The `L/2` term adjusts the logistic function such that the inflection point occurs at the origin. The parameters of the logistic functions for each possible action were tuned using Excel. Equations 12 – 14 show the `r2` assignment structure for all possible actions.

Equation 12 is the `r2` assignment method for a buy action.

```
if prcnt_change <= 0:
    r2 = sigmoid(30, 0.3, 2.3, prcnt_change)
elif prcnt_change > 0:
    r2 = sigmoid(20, 0.25, 0, prcnt_change)
```

Equation 12

Where `prcnt_change` is the percent change between the cost basis and the price immediately following action selection.

The `r2` assignment for a buy action is particularly unique because it is dependent upon the value of the percent change term. The algorithm is punished for selecting a buy action where the subsequent price movement is naught or negative. Figure 7 shows how r2 is assigned for a buy action.

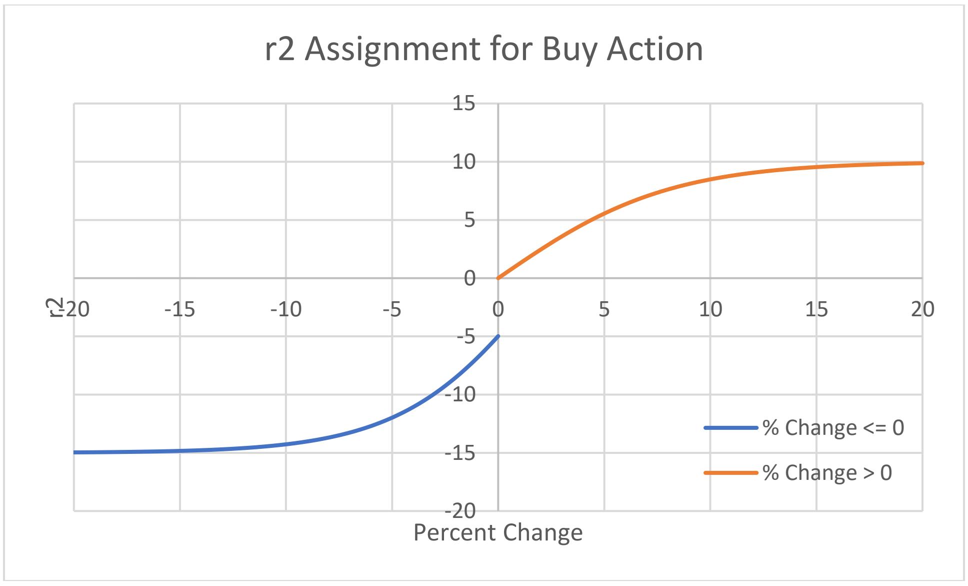

Figure 7

Equation 13 is the `r2` assignment method for a sell action.

```
r2 = sigmoid(20, 0.3, 0, prcnt_change)
```

Equation 13

Figure 8 shows how r2 is assigned for a sell action.

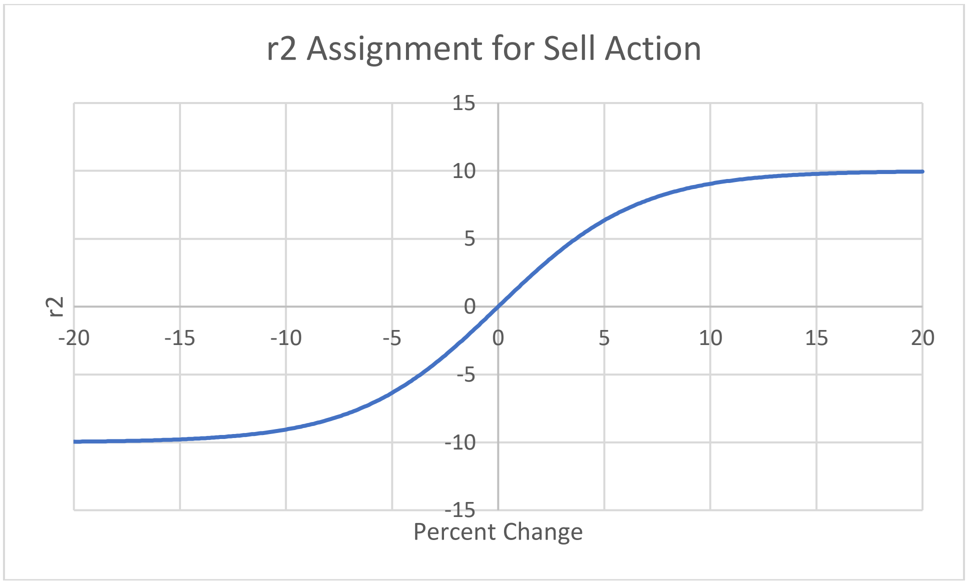

Figure 8

Equation 14 is the `r2` assignment method for a hold action. `r2` for the hold action is referred to as `hold_pnl_sig`.

```
hold_pnl_sig = sigmoid(20, 0.01, 0, hold_pnl)
```

Equation 14

Where `hold_pnl` is the PnL realized by holding a security.

Figure 9 shows how r2 is assigned for a hold action.

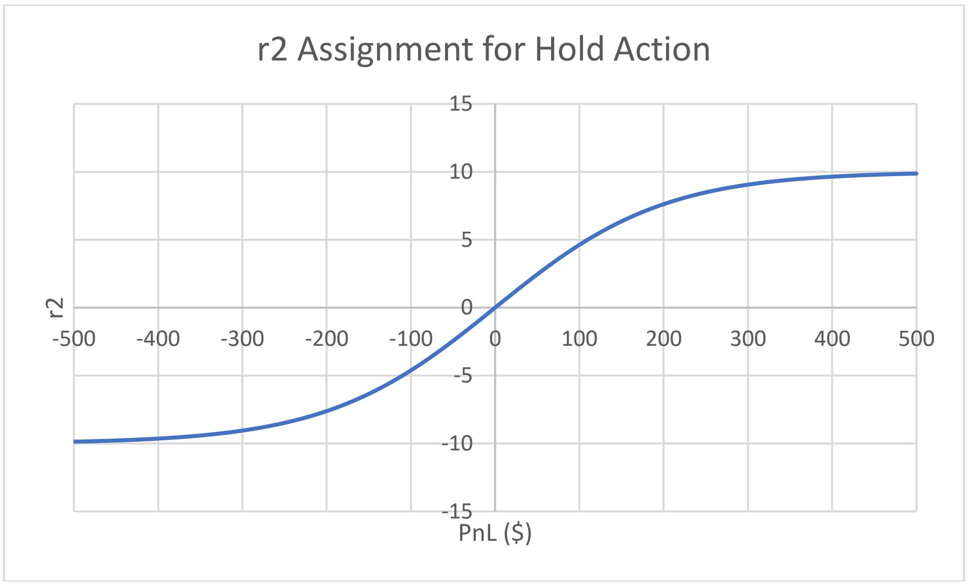

Figure 9

Hyperparameter Tuning

Arriving at values for most of the hyperparameters in the initialize function began with a reasonable guess based on what a hyperparameter was controlling. The impact of the value on the performance of the portfolio was then tested by running a back test. The value was then incrementally increased or decreased until more desirable performance began to emerge. For instance, `context.alpha` controls the importance between the current reward and the current Q-value during a Q-learning update. Due to the nature of the equation, this value is always between 0 and 1 and therefore initially a value of 0.5 was tested. When this did not produce the desired results, a value of 0.6 and then a value of 0.4 were tested. Using a value of 0.4 produced better results than 0.6 and therefore 0.45 was tested to ensure that the values between 0.4 and 0.5 were explored. The value was then reduced to 0.42 and remained there as this seemed to produce the best results. Most values were arrived at in such a manner, by beginning with a reasonable assumption and making incremental adjustments based on the observed performance of the algorithm.

There were a few hyperparameters that were much more troublesome to tune as there were no known rules of thumb to guide the tuning process and/or the equations they were a part of did not intrinsically impart limits on the possible values. The majority of these hyperparameters were used to control the annealing function and therefore the most difficult to tune of these, `context.temp`, will be primarily discussed. This hyperparameter is the initial Boltzmann temperature that controls how randomly actions are selected. See Equation 4. This hyperparameter (as well as the others) was tuned endlessly and therefore only the major breakthroughs will be discussed.

While parameter tuning with the initial annealing function still in place there was a strong urge to keep the initial Boltzmann temperature value under 30. Therefore, to extend the learning period a large value for `context.decay_rate` was necessary. The decay rate was raised as high as 0.998, which did not have the desired result on the performance of the algorithm. The initial Boltzmann temperature was revisited and a value of 50 was used as there was still unwarranted hesitation to raise the value by much. This seemed to have a positive impact on the performance and therefore the value was doubled and the decay rate was lowered and tested at values between 0.99 – 0.996. Even though the performance had improved, it was still unable to compete with the benchmark. This was when the initial method of reheating the Boltzmann temperature was implemented. This greatly changed the performance of the algorithm and values as high as 210 were tested for the initial Boltzmann temperature, and 150 for `context.init_decay_temp`, which is the initial value for the second learning phase. After much observation and parameter tuning the Boltzmann annealing function described above was developed. As was predicted, allowing the Boltzmann temperature to remain constant over an extended period allowed for a reduction in the initial temperature. `context.temp` was able to be set to 90. Due to the more effective initial learning phase, the initial temperature and decay rate for the secondary learning phase could also be reduced to 30 and 0.98 respectively.

IV. Results

Given the nature of this project, the “Model Evaluation and Validation,” “Justification,” and “Free-Form Visualization” sections will be combined in this “Results” section as these components are inherently inseparable and must be discussed together.

This section relies on various figures to illustrate the performance and reliability of the model free autonomous portfolio manager. Over 50 full backtests have been completed with the provided code. The results of 5 representative full backtests will be discussed using two figures per backtest.

The first figure in each backtest subsection (Figures 11, 13, 15, 17, and 19) plots the performance of the algorithm and SPY over the entire backtest. The red line in each of these figures represents the testing phase and the probability cone shows the predicted performance of the algorithm during testing based on its previous performance. These figures were generated using Quantopian’s `create_full_tear_sheet`, which outputs various plots for analyzing an algorithm’s backtest performance.

The second figure in each backtest subsection (Figures 12, 14, 16, 18, and 20) plots the percent change or returns of the autonomously managed portfolio with that of SPY beginning at the date the algorithm completed the learning phase and entered the testing phase. Equation 1 in the “Metrics” section shows how the returns for the testing phase were calculated when the returns for the entire backtest were all that was available.

From plotting the post learning performance of the backtests and the benchmark it is clear that during the testing phase the returns for Backtests 1 and 5 essentially match the performance of SPY, Backtests 2 and 3 are 10-15 percentage points short of the benchmark, and Backtest 4 outperforms the benchmark. Backtests are arranged in order from highest to lowest returns over the course of the entire backtest. The results of each backtest are discussed in further detail below.

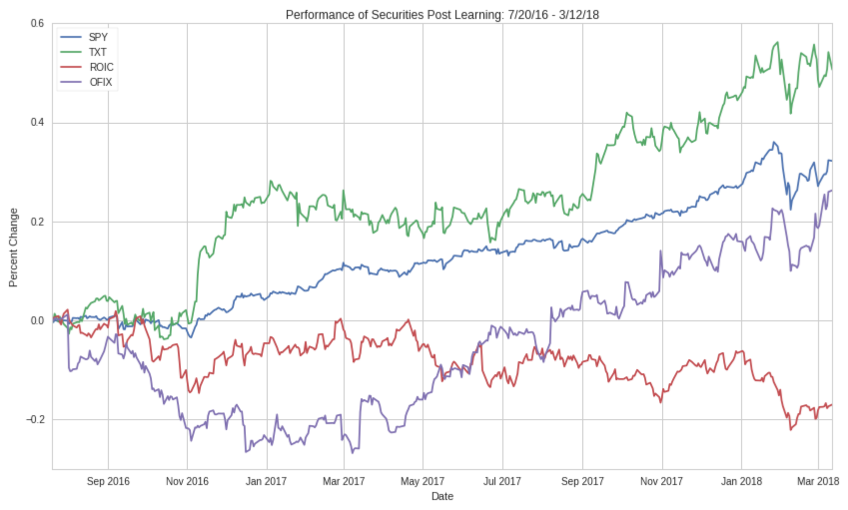

Figure 10

Figure 10 shows the returns of each security and the benchmark during the testing phase. This figure along with figures in the Appendix can be used to determine the impact of specific securities on the performance of the algorithm. In order to compare SPY to the backtests, it is important to note that the final return value for SPY is 0.322, or 32.2%.

Backtest 1

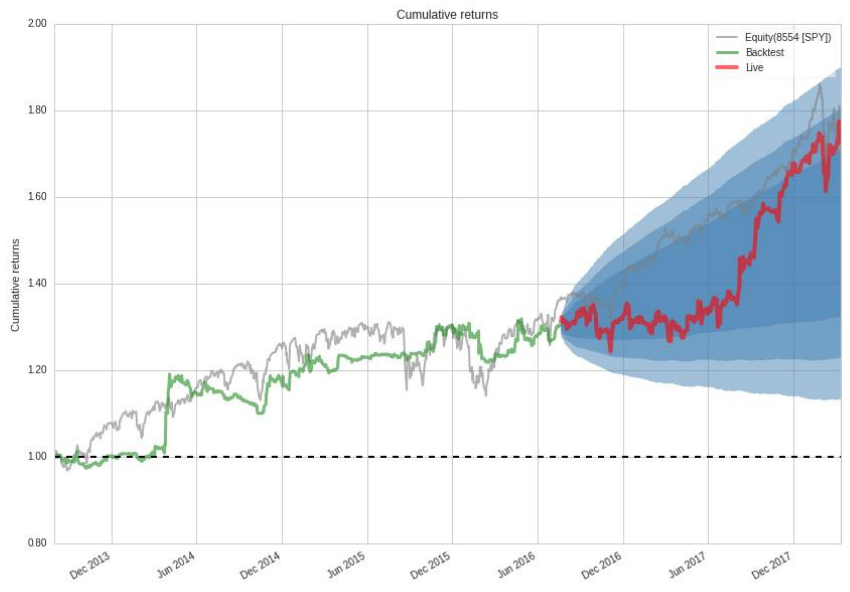

Figure 11

Figure 11 shows the results of the backtest with the highest cumulative returns over the course of the entire backtest. This backtest produced returns of 74.8%. It is likely that the Q-learner was relatively fortunate with a few of its selected actions throughout the learning phase. For instance, it is clear from the initial large spike in the plot that the Q-learner was lucky enough to open a position in OFIX prior to a leap in that security's value. See Figure 1 for evidence of this. The behavior of this backtest contrasts that of Backtest 4 where the Q-learner struggles throughout the learning phase to generate positive returns but is still able to learn an optimal policy and generate excellent results during testing.

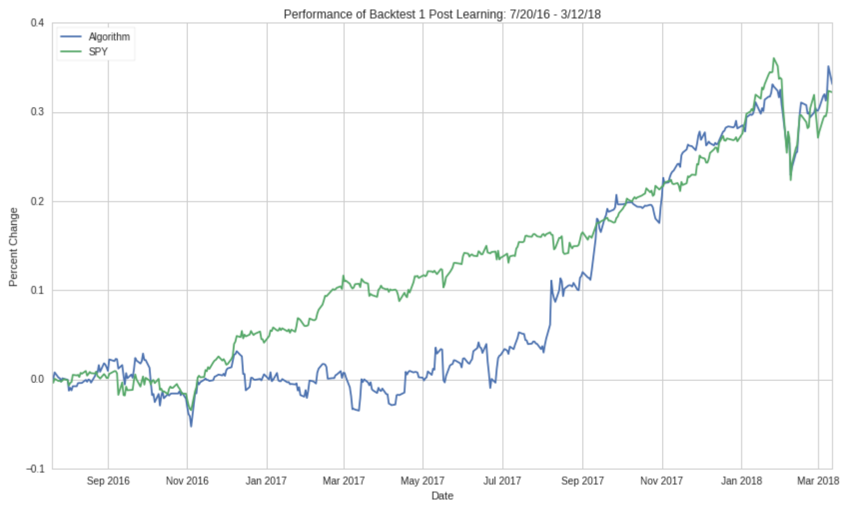

Figure 12

Figure 12 shows that despite a bit of initial stagnation likely due to the performance of the securities during this time, the Q-learner was able to catch up to and match the performance of the benchmark during the testing phase. The algorithm produced a return of 34.1% during this period.

Backtest 2

This backtest was included to show that despite impressive performance during the learning phase it is possible for the Q-learner to learn to perform actions for certain states that could be detrimental during testing.

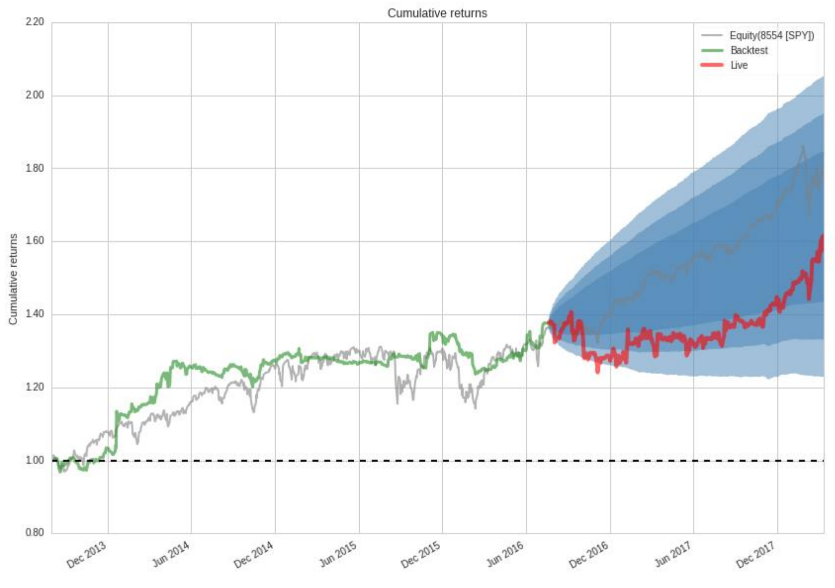

Figure 13

Figure 13 shows that during the learning phase the Q-learner appears to be learning a policy that allows it to outperform the benchmark but through a series of suboptimal actions during the testing phase it quickly incurs losses that prevent it from matching the performance of the benchmark during testing. Perhaps the seemingly excellent action selections made by the Q-learner did not allow it to learn a policy that was able to detect an oncoming loss. It is also possible that the Q-learner did learn appropriate policies but could not predict the oncoming downturn because a clear signal is not evident from the presented states. Despite the initial downturn during testing, the algorithm does recover relatively quickly. It is also worth noting that all 3 securities in the portfolio trend negatively during the first few months of the testing phase, which is evident in Figure 10.

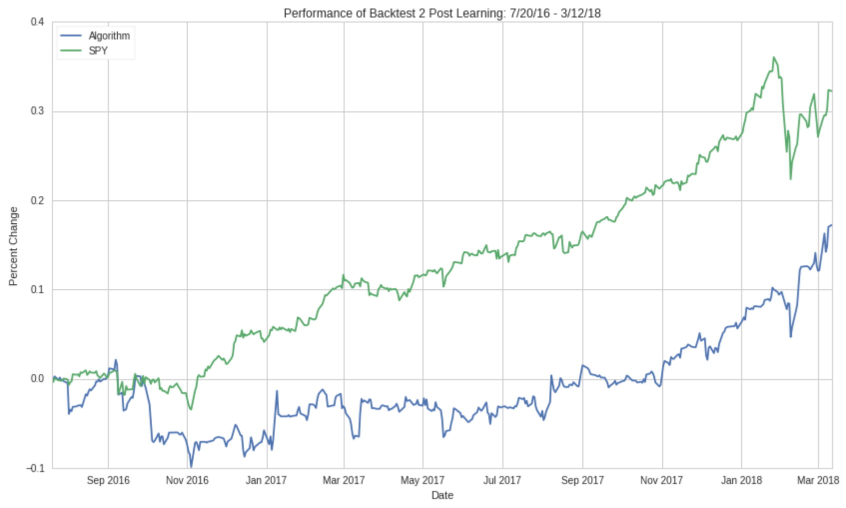

Figure 14

Figure 14 shows the performance of the Q-learner throughout the testing phase. Even though the algorithm is unable to match the returns of SPY, it recovers from the initial loss it sustained and does not experience another similar downturn for the remainder of the testing phase. This is a testament to the robustness of the policy learned by the Q-learner.

Backtest 3

This backtest was included to illustrate the potential strength of the developed algorithm. This backtest produced returns of 21.8% during the testing phase, which is 10.4 percentage points lower than the benchmark. Despite underperforming the benchmark, this backtest is the only one in which the performance of the algorithm did not in some way reflect the large downward spike that SPY experiences near the end of the testing phase. This behavior is clear in Figure 16 and shows that the Q-learner was able to learn a robust policy that, in this instance, was not affected by a market downturn. From approximately April 2017 to the end of the testing phase, the performance of the algorithm is relatively constant and linear. It seems possible that this backtest would have outperformed the benchmark if all three securities did not trend negatively at the beginning of the testing phase.

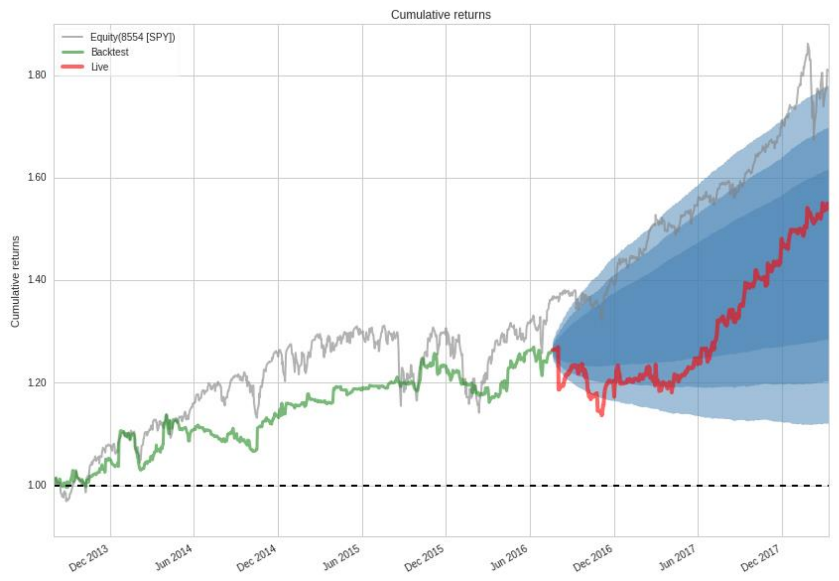

Figure 15

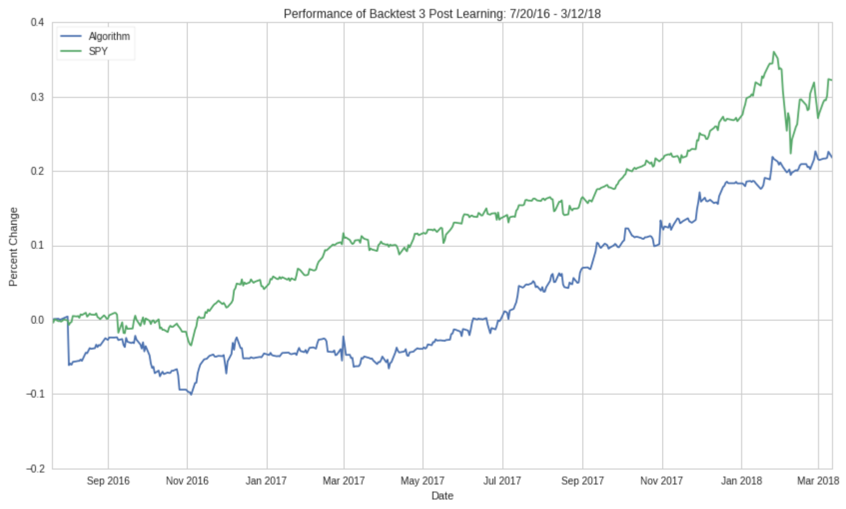

Figure 16

Backtest 4

This backtest shows that the Q-learner can learn an effective policy despite underwhelming performance during the learning phase.

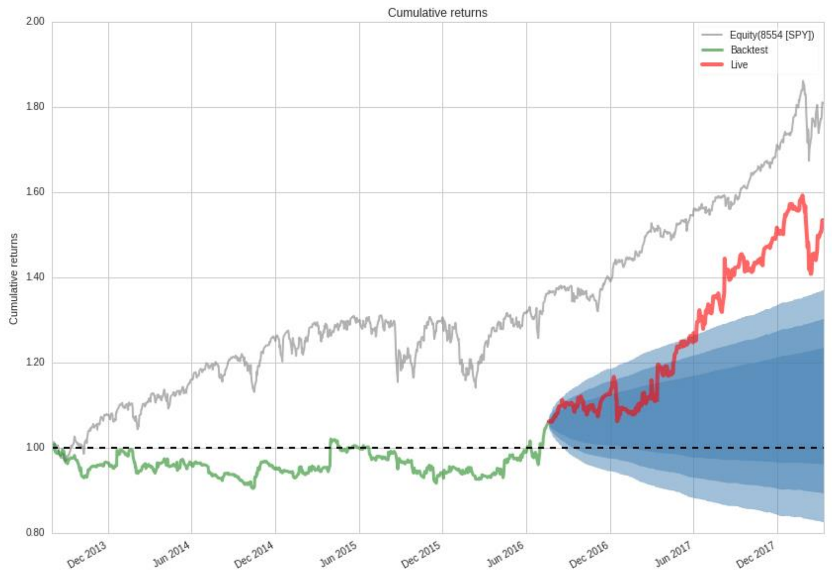

Figure 17

Figure 17 shows a seemingly terribly performing backtest that struggles to obtain and retain positive returns while learning. One would therefore assume that the Q-learner could not have possibly learned a useful policy.

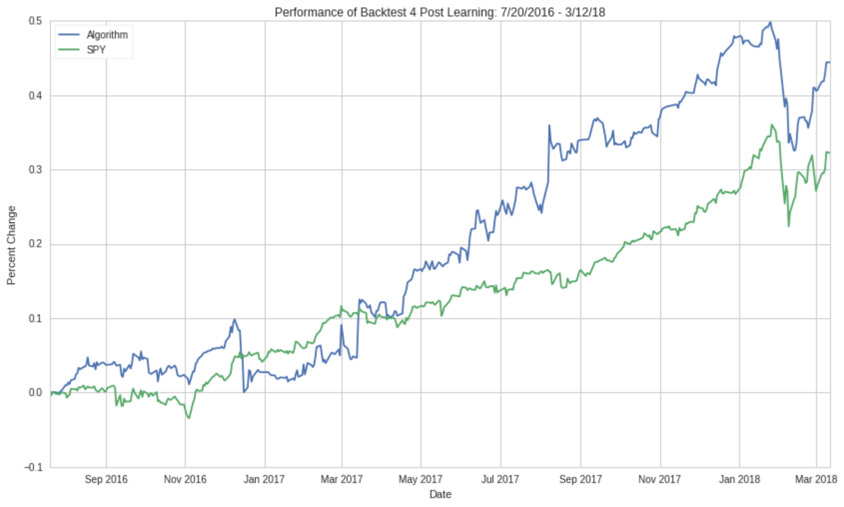

Figure 18

Figure 18 shows that despite apparent poor performance during the learning phase the Q-learner was able to learn a policy that allowed it to outperform the benchmark with a 44.6% return during the testing phase. The performance of this backtest during testing is more erratic (contains more significant spikes) than that of Backtest 3 and is significantly influenced by the downward movement of the market toward the end of the backtest.

Backtest 5

This backtest was included as another example of a backtest that matches the performance of the benchmark. For this backtest the algorithm realized 30.1% returns, which is only 2.1 percentage points below the benchmark. The performance of the algorithm during this backtest is more desirable than that of Backtest 1 as it does not experience a period of stagnation at the beginning of the testing phase. Instead, the algorithm immediately begins to recover from the loss it incurred.

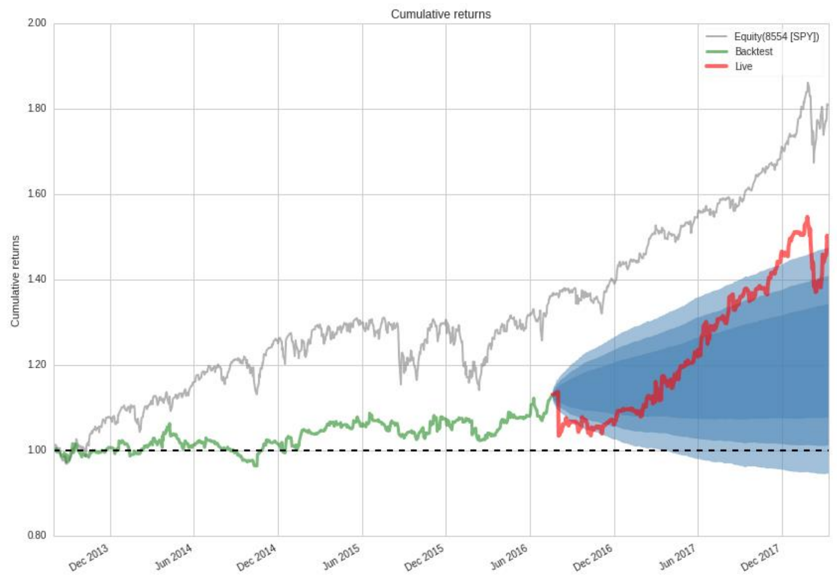

Figure 19

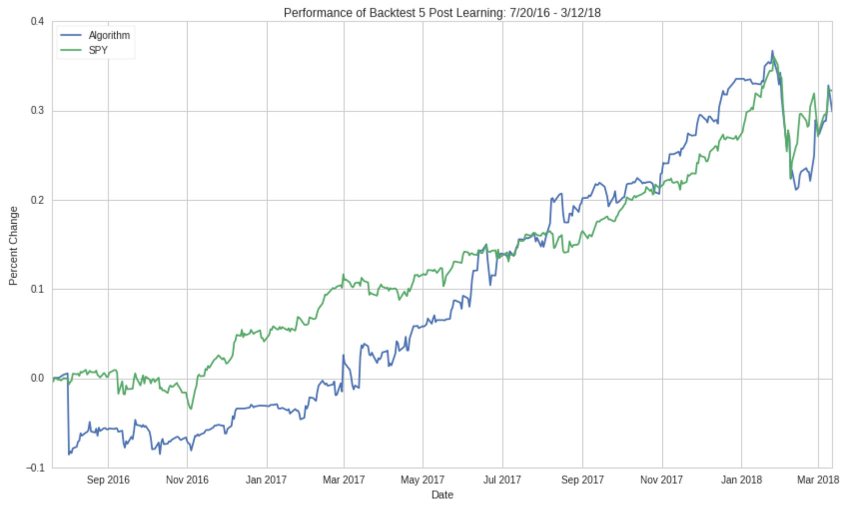

Figure 20

V. Conclusion

Reflection

Through careful development of the state function, Boltzmann temperature annealing scheme, and the reward function it was possible to develop an algorithm that can successfully match and occasionally beat the performance of SPY. This project successfully demonstrates proof of concept for an autonomous portfolio manager.

Prior to algorithm development it was necessary to become familiar with the Quantopian platform. The algorithm was developed in the Quantopian IDE and the plots and tear sheets were generated in Quantopian’s Jupyter Notebook Research environment. Quantopian was selected as the platform because it has extensive historical pricing data that is appropriately preprocessed, a built in backtester, and built in functions specifically for market analysis. This saved a great deal of time that was put towards developing the algorithm.

Q-learning was selected to control the autonomous portfolio manager. This model free method learns by exploring the environment it is made to interpret. A Q-learner interprets an environment through the states presented to it and learns an optimal policy be receiving rewards that correspond to how appropriate a selected action is for a given state. Action selection is initially highly random, or in other words exploratory and over time transitions to being entirely exploitatory. Since action selection was done using a Boltzmann distribution, this tradeoff is controlled by the Boltzmann temperature, τ.

In order to implement the Q-learner it was necessary for it to be able to sense the environment it was expected to interpret and act in, which in this case was the stock market. Providing the Q-learner with a state composed of conditions for industry accepted technical indicators that are used by professional and amateur traders to assist in the determination of buy and sell points provided the Q-learner with the necessary information about the environment. The state structure is presented in Equation 3. After converting the pricing data into interpretable states for the Q-learner, the Q-learner needs to act on the states it is presented for each security. The Q-learner first determines whether the state is present in the Q-table. If the state is not yet present it is added to the Q-table and the reward for each action is initialized to 0. An action is then selected based on Equation 4, the Boltzmann distribution. The tradeoff between exploration and exploitation is controlled by τ, which is annealed by a function visualized by Figure 6. A reward is then assigned according to the rules set forth in the “Reward Function” subsection found above. The Q-table is then updated with the new reward values in accordance with Equation 5, the Q-learning update rule. This process is then repeated until the testing phase is reached and the Q-learner must act based on the Q-table it has built without any rewards to indicate whether the actions it takes are a benefit or detriment to the performance of the portfolio.

Aside from the endless parameter tuning, developing an appropriate annealing function for the Boltzmann temperature and developing the reward structure were the most difficult challenges of the project and consumed an enormous amount of time.

Creating the current annealing function required observing countless backtests with the intermediary annealing function that had an initial learning phase and relearning phase that were both standard decay functions of the form presented in Equation 6. Eventually after methodically, incessantly and relatively unsuccessfully (the algorithm was not performing as consistently as desired and required a much higher initial τ than desired) tuning the parameters and observing the various outcomes of the predecessor to the current annealing function it became clear that first a break was in order, and second a different approach was required. A thought arose based off the fact that the function that needed perfecting was called an annealing function. Annealing is a heat treatment process used to reduce internal stresses and increase ductility in metals. Immediately (after the break(s)), steel temperature transformation diagrams came to mind. These diagrams are used to predict the final microstructure of a steel sample that is cooled. Many problems involving these diagrams ask for the resulting steel microstructure given that the sample begins at a specific temperature, is quenched to another temperature and held there for a specific amount of time, then is quenched to another temperature and held there for a specific amount of time, etc. It was then decided to attempt this process with the Boltzmann temperature and after dialing in the parameters it worked. It also made the Q-learner more reliable and did not require such a high initial temperature. It was extremely exciting and interesting to see that this previous knowledge (despite taking so long to think of) proved useful for a machine learning problem.

Developing the reward structure was daunting and predicted to be one of the more challenging tasks of the project as there was no prior experience developing a reward structure. For the SmartCab project the reward structure was provided and therefore this was looked at for inspiration. It was obvious the reward structure needed to reflect the impact an action had on the performance of the portfolio and the values of the reward needed to be limited or rarely reach values outside a specified range. Knowing the difficulty this would pose, a very simple reward structure was used as a place holder until other portions of the algorithm were confirmed to be functioning properly. After struggling with many different linear methods of rewarding the Q-learner, the final structure was developed. The linearity of previous reward structures was unable to provide the Q-learner with the necessary reinforcement to determine whether a state was providing a strong action signal. Previous reward structures were also loosely bound. Generally, the values fell within a desired range, but occasionally there were large negative or positive outliers. In order to provide a more uniform reward structure these issues needed to be eliminated. Thinking about how to get the reward structure to assist the Q-learner in determining action signals caused the word ‘signal’ to resonate. A sigmoid, or logistic function can place limits on the reward, would assign rewards such that values closer to the targets would receive rewards close to either the upper or lower limits, and would be highly tunable. Excel was then used to tune the sigmoid functions for the available actions. After some trial and error, it was clear that this reward structure was superior at providing the Q-learner with valuable feedback. Designing an appropriate reward structure proved to be a more difficult and important task than initially assumed. It was known that the Q-learner would not function properly without an appropriate reward structure, but it was assumed that the Q-learner would be able to function with a simpler structure as long as the performance of the portfolio due to an action was reflected in the reward.

Overall this project was quite the challenge but provided many excellent learning opportunities. At many stages in the process the desire to resubmit the Capstone Proposal and select something that may have been easier arose. Initially the determination to get a working autonomous portfolio manager working drove the project forward, later the time invested in the project kept the momentum going. Using the Quantopian platform and the packages it contained provided practice learning and applying new APIs, using a backtester, and coding in a manner that required the use of specific functions to run properly. This was an interesting challenge on its own but was considered worth the time investment due to plans to potentially use the platform later. Throughout the project, knowledge of various technical indicators was reinforced and learned, and the importance of thoughtful parameter tuning became excruciatingly clear.

Improvement

Since this project was mainly to show proof of concept for an autonomous portfolio manager, there are many improvements that could be made that would bring it closer to something that would be considered acceptable by a hedge fund. Primarily, it would be necessary to allow the algorithm to select stocks to trade from a predetermined universe of liquid securities. Quantopian has two such built in universes referred to as the Q500US and Q1500US. Another machine learning exercise could be to train an algorithm to make a dynamic portfolio that rebalances itself by constantly selecting stocks based on learned criteria from one of these universes. The actions taken with these stocks could then be determined by the Q-learner. Implementing methodology for short selling securities would also improve the performance of the autonomous portfolio manager. Short selling involves selling a security that is not currently owned and buying it back (short covering) at a (hopefully) lower price in order to generate a profit. Short selling can therefore also be used as a method of hedging against negative price movements. The fees and other intricacies for short selling can be complex, which is one of the reasons it was not supported by this algorithm.

Using the Q-learner to trade a larger portfolio would require a training method slightly different than the one currently implemented. It would not be feasible to train the Q-learner on a large number of securities. Therefore, initial training would be done by having the algorithm select 5-20 representative and non-correlated securities with the intention of creating a general Q-table. After an initial testing phase another learning phase would occur with a new representative set of securities. Ideally after sufficient learning a method for updating the Q-learner's training with a background learning phase while it was live trading would be developed. This could possibly include a method for detecting when a relearning phase should occur.

To take advantage of all market movements it would be desirable to allow the algorithm to use minute data for training and trading. This would significantly increase the processing power required by the algorithm, which may not be supported by Quantopian. Using minute date would also require retuning the technical indicators and possibly restructuring the state function.

Allowing the algorithm to select the number of shares to trade would allow it to minimize risk and maximize rewards. Currently the algorithm can only purchase 40 shares of a security at a time and must sell all shares when it chooses to sell. This would likely require a separate machine learning function for selecting the number of shares. Currently, it is unclear what factors would need to be analyzed to determine the optimal number of shares. However, it would be necessary to ensure that the portfolio did not become overexposed to any security or industry.

Implementing a method for dynamically changing technical indicator parameters and weighing the importance of technical indicators during action selection based on market conditions and previous experiences could greatly improve the performance of the algorithm. However, this would be extremely difficult to implement and would therefore be one of the very last improvements made to the algorithm.

These improvements would undoubtedly require the entire reward structure to be revised. The reward for the immediate impact of an action, `r2`, would require far less revision than `r1`, which is the reward for predicted future performance due to the selected action. `r2` would likely only require tuning of the sigmoid functions. To improve `r1`, it may be desirable to implement a more powerful predictive method than an ordinary least squares regression. A more powerful machine learning technique could possibly allow `r1` to be assigned in a manner that represents the magnitude of the impact that an action is predicted to have on the future performance of the portfolio.

The above-mentioned improvements would ideally produce an autonomous portfolio manager that exhibits steady growth similar to Backtest 3, yet outperforms the market, similar to Backtest 4 and is resilient to unexpected events similar to Backtest 5. Backtest results are also expected to be more consistent across various backtests if these improvements were implemented. By training the algorithm to detect when it needs to undergo another learning phase, it would be possible to generalize the algorithm to any given minimum date range that allows for sufficient learning opportunities. This would also allow the algorithm to run for an (ideally) indefinite amount of time as it would be able to update the Q-table as needed.

APPENDIX

Backtest 1

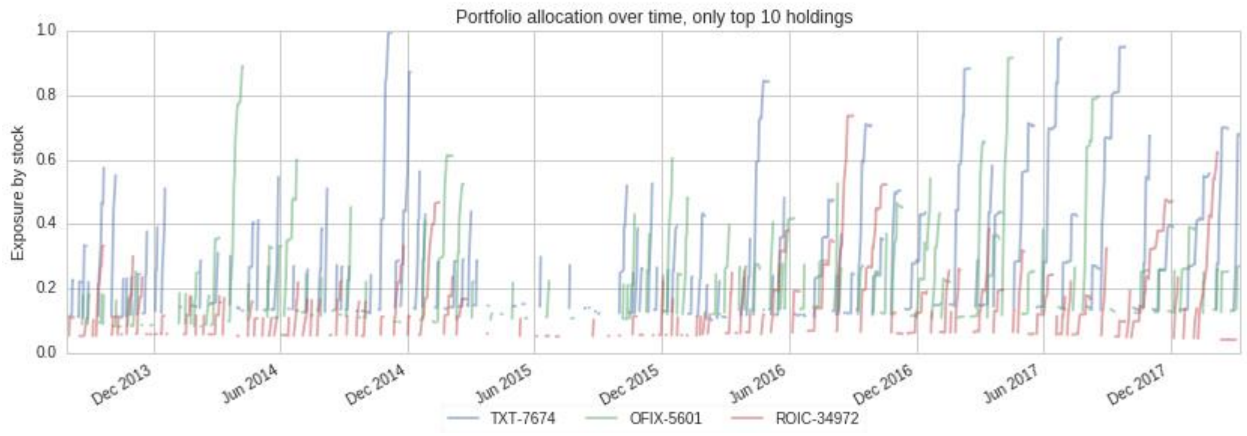

Figure A-1

Backtest 2

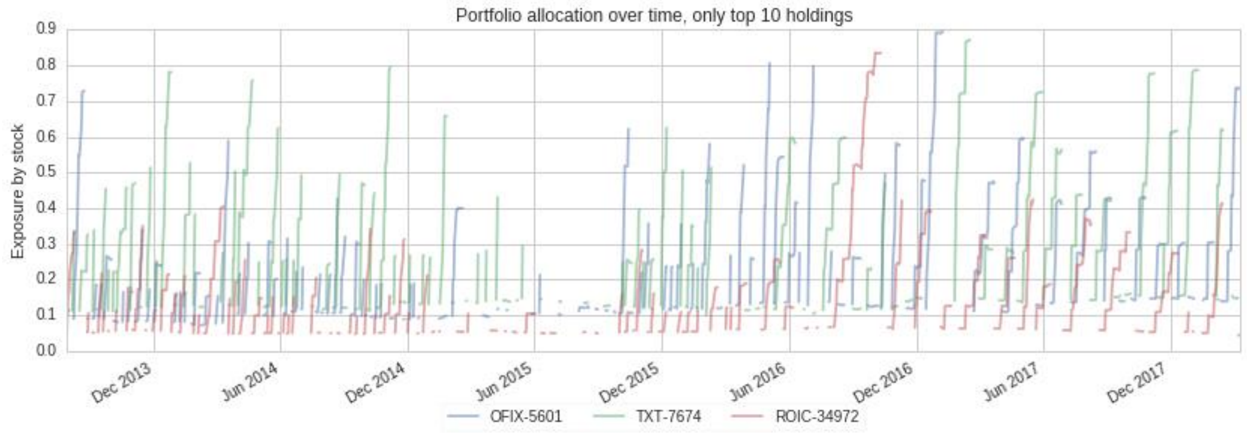

Figure A-2

Backtest 3

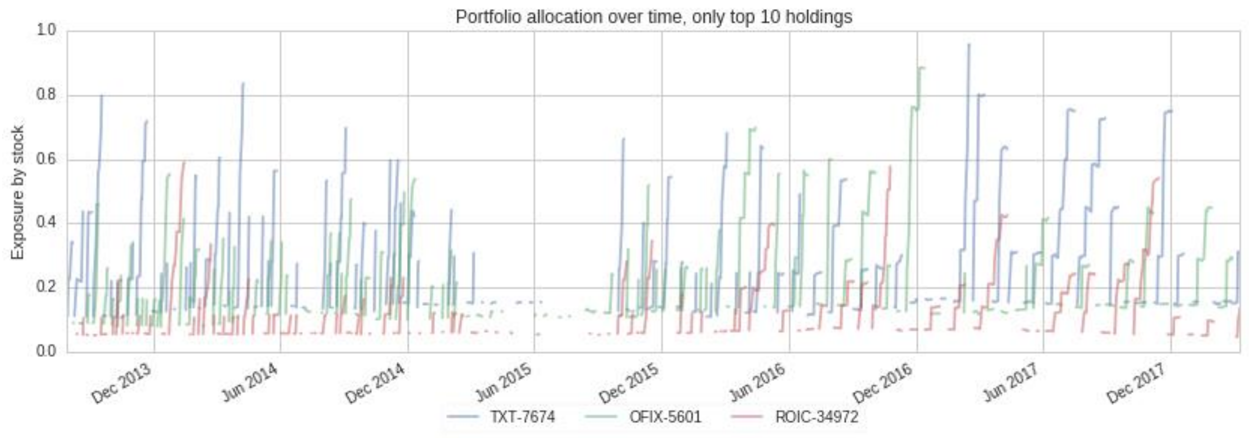

Figure A-3

Backtest 4

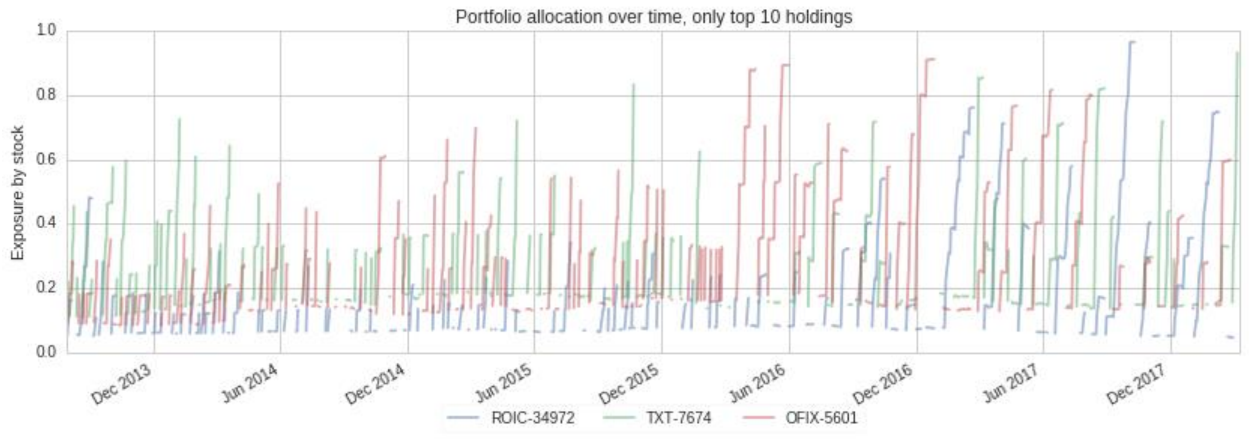

Figure A-4

Backtest 5

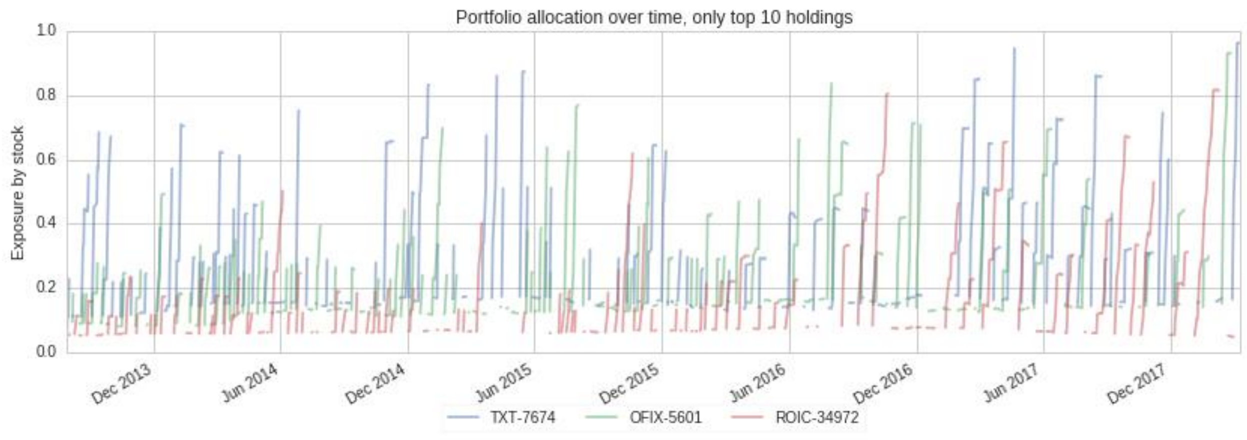

Figure A-5