5-2019

Reinforcement Learning Applications in Real Time Trading

Yutong Liu
University of Pennsylvania

Reinforcement Learning Applications in Real Time Trading

Abstract

This study focuses on applying reinforcement learning techniques in real time trading. We first briefly introduce the concept of reinforcement learning, definition of a reward function, and review previous studies as foundations on why reinforcement learning can work, specifically in the setting of financial trading. We demonstrate that it is possible to apply reinforcement learning and output valid and simple profitable trading strategy in a daily setting (one trade a day), and show an example of intraday trading with reinforcement learning. We use a modified Q-learning algorithm in this scenario to optimize trading result. We also interpret the output policy of reinforcement learning, and illustrate that reinforcement learning output is not completely void of economic sense.

Keywords

reinforcement learning techniques, trading

Disciplines

Business

REINFORCEMENT LEARNING APPLICATIONS IN REAL TIME TRADING

by

Yutong Liu

An Undergraduate Thesis submitted in partial fulfillment of the requirements for the

JOSEPH WHARTON SCHOLARS

Faculty Advisor:

Ryan Hynd

Associate Professor, Department of Mathematics

THE WHARTON SCHOOL, UNIVERSITY OF PENNSYLVANIA

MAY 2019

Abstract

This study focuses on applying reinforcement learning techniques in real time trading. We first briefly introduce the concept of reinforcement learning, definition of a reward function, and review previous studies as foundations on why reinforcement learning can work, specifically in the setting of financial trading. We demonstrate that it is possible to apply reinforcement learning and output valid and simple profitable trading strategy in a daily setting (one trade a day), and show an example of intraday trading with reinforcement learning. We use a modified Q-learning algorithm in this scenario to optimize trading result. We also interpret the output policy of reinforcement learning, and illustrate that reinforcement learning output is not completely void of economic sense.

1 Introduction

We aim to explore and apply reinforcement learning techniques to real time data set and output efficient trading strategies in this study. As the emphasis of the paper is on application, we will review the concept of reinforcement learning and how it works in **section 2**, briefly review a simulated example in **section 3.1**, and put the main focus on our application to real data sets and interpretation of reinforcement learning in finance, in **section 3.2** and **section 4**. All data sets used in this study are obtained from Wharton Research Data Service (WRDS).

2 Reinforcement Learning

2.1 Problem Formulation

In general, reinforcement learning is an algorithmic procedure used to train an *agent* to *act* in an *environment* in order to maximize a *reward function*. We do not provide explicit supervision for the algorithm, but rely on the rewarding function to indicate how well the agent is performing in the environment. In our problem, the agent is the trader, the actions are volumes of securities to trade, and the environment is the price movement of the security. To sum up, we want to train a bot trader which places optimal trades on the security according to a reward function, which we specify in the subsequent paragraphs.

There are a few parameters to specify and model in our reinforcement learning formulation, namely, the environment, actions, rewards, and policies. To model the environment, we discretize it into different states that indicate our current (and subsequent) conditions. The actions, as stated above, are the amount of securities to trade. The rewards come from a reward function, which we specify below, and the policies constitute the mapping from states to actions, i.e., the optimal move our agent needs to take to maximize rewards. Detailed information of parameters, specifically for our case, are as follows:

1. **States:** States are discretized version of market information. In this study, we choose the deviation of current stock price from moving average as the state space, and discretize to 0.01 precision. That is to say, the entire state space is consist of discrete values up to 0.01 precision, with

the minimum being the minimum deviation from moving average in the training period, and the maximum being the maximum deviation from moving average in the training period. We may not encounter every state in the state space in the training window, but the bound should sufficiently encapsulate all possible scenarios.

State representation reflects our knowledge of current condition, but say nothing more about the action we should take. It worths noting that our choice of state representation may not be, with large probability, the optimal choice.

2. **Actions:** We define our action space as a discretization of amount of securities to trade. Namely, we define a certain size of lot we want to trade at first (100 units of shares, for example), and spread that over a unit interval. For instance, we can have $\{-1000, 900, \ldots, 0, 100, \ldots, 1000\}$ as the action space. We also want to limit the amount of securities in our overall holdings to avoid too much exposure and liquidity shortage in a later time. If we select a trade that exceed that limit, we will just cut the trade such that the overall holding is still restricted to the predefined space.

3. **Rewards:** As aforementioned, the reward function serves as the indicator for how well our action is doing. Therefore it is crucial for the validity of our learning algorithm. Drawing inspiration from [3], we define our reward function as that of a risk-averse investor. We assume that the asset return follows a mean-variance equivalent distribution and our utility resulted from wealth is strictly increasing, and that we want to maximize our utility from wealth, which is equivalent to maximize final wealth. To maximize $\mathbb{E}(u(w_T))$ with respect to the underlying asset, which we assume to have a mean-variance equivalent distribution, it suffices to solve the equivalent problem [3][4]

$$\begin{aligned} &\max_{\pi} \ \mathbb{E}(w_T) - \frac{\kappa}{2}\mathbb{V}(w_T) \\ =&\max_{\pi} \ \sum_{t=1}^{T} \mathbb{E}(w_t - w_{t-1}) - \frac{\kappa}{2}\mathbb{V}(w_t - w_{t-1}) \end{aligned}$$

where w_T denotes the final wealth, and κ is a positive constant. We explicitly assume that returns (change in wealth) is independent between different periods, so maximizing the final wealth can be broken down into sub-problems where we maximize the expected wealth minus the variance of wealth in each sub-period. Note that since we aim to maximize the mean variance equivalence of the dollar amount wealth but not the return, we implicitly assume the concavity of our utility function, i.e., we

are risk-averse.

4. **Policies:** Our goal in reinforcement learning is to output an optimal policy (mapping) from given states to actions. Intuitively, we want to output a map so that when encountering new situations, we can trace the situation back in the map and adopt the optimal decision based on training data. Using Sutton's notation [7], we define the value of a state as

$$v_\pi(s) = \mathbb{E}(\sum_{i=0}^{T} \gamma^i R(s_i) \mid s_0 = s, \pi)$$

where γ is the discount factor, $R(s_i)$ being the reward received from choosing action in state s_i, and the sum can be finite or infinite. Certainly, we have finite sum in the implementation.

2.2 Q-learning

In this study we applied a specific algorithm for reinforcement learning, Q-learning, developed by Watkins [9]. The reason for choosing Q-learning over other frameworks is two folded: 1. Q-learning is model-free. We can learn directly from interacting with the environment without modeling the environment's distribution, i.e., the distribution of stock price. 2. Q-learning is off-policy. We do not need to initiate a whole trial of iteration for each state, and only update the visited pair of state and action. Note that on-policy algorithm would require us to simulate an entire trial for each state, and that is more computationally intensive. Moreover, as we only need to update state-action pairs which we visit, we can feed in the time series data (the raw data) as the training set without any simulation.

To implement Q-learning, we first initiate a Q matrix with each row corresponding to each state and each column corresponding to an action. When we encounter a state, we do epsilon-greedy sampling for the best action and update the Q value by

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$$

using Sutton's notations. α is the learning rate and R_{t+1} is the reward received after choosing action in state S_t. Q converges to the optimal value with probability 1 [7].

3 Detailed Examples

3.1 Foundation

We based the validity of applying reinforcement learning to finance algorithm on Ritter's observation that reinforcement learning can achieve near arbitrage for a discretized Ornstein-Uhlenbeck process [3]. Assume a simulated data as in Figure 1, generated around an equilibrium price 50. We can train the model on the input data with Q-learning and achieve near arbitrage in testing data. Remarkably, the model does not know the parameter of the mean reversing process, nor does it try to learn the parameters. It just learn the optimal strategy through repeated trials and errors. As a consequence, it is possible to exploit the stock price data without learning its distribution.

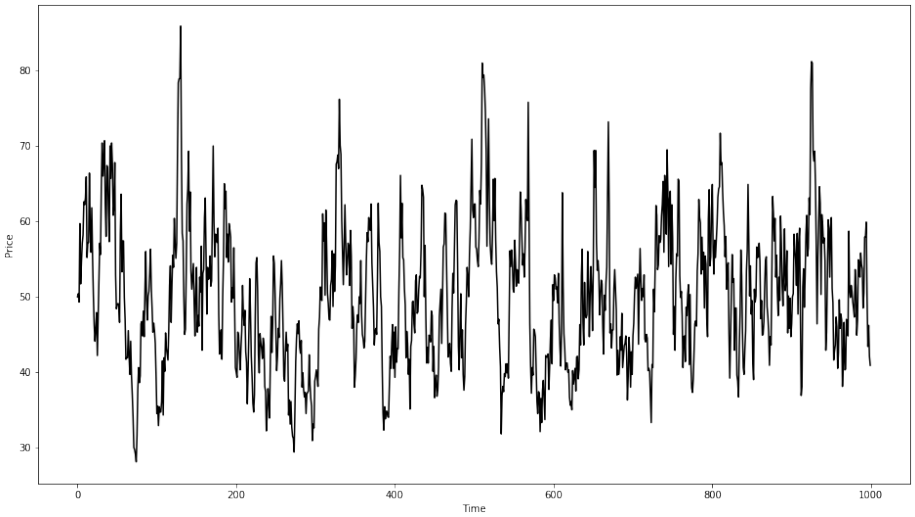

Figure 1: Ornstein-Uhlenbeck Simulation Around 50

One drawback from the simulation is that although we do not need to learn the distribution, we assume that the training and testing data have exactly the same distribution. This usually is not the case for stock prices, as the distribution of stock prices in a forward window is not exactly the same as the distribution over a past window. Therefore, we want to apply the algorithm to real data and test whether we will obtain favorable results. The basic assumption is that the difference between distributions of train/test data is not radical, and the algorithm can still exploit the structure of the data.

3.2 Real Time Data

We want to apply Q-learning framework on real data to test how it will perform. Noting that we are in no way attempting to train an "intelligent" bot that can think and weighs trading decisions, but merely output a decision rule which we systematically apply. We cannot tell what the policy is going to do prior to completing the training, and can only know how it works by inspecting the output Q-matrix.

3.2.1 Q-Learning with Daily Trade

We first applied Q-learning on daily stock price data. Since our state space is defined as the deviation from moving average, we would like to trade assets with more mean reversing characteristics. With that consideration, we trade equity with large capitalization, and hope the trading strategy will outperform a simple buy-and-hold strategy. Here is the basic set up of Q-learning algorithm:

Algorithm 1 Q-Learning

procedure Q-Learning(data, $\kappa, \gamma, \alpha, n$)
  Initialize empty Q matrix
  while not end of data **do**
    Choose action $a \in \mathcal{A}$ by ϵ-greedy sampling
    Compute cost c_t from action a
    $v_{t+1} = h_t \times (p_{t+1} - p_{t-1}) - c_t$
    $R_{t+1} = v_{t+1} - \frac{\kappa}{2} v_{t+1}^2$
    $Q[s,a] = Q[s,a] + \alpha(R_{t+1} + \gamma(\max_{a'} Q[s',a'] - Q[s,a])$
    $s \leftarrow s'$
  end while
  Return Q
end procedure

where h_t is the holding at period t, p_t is the price at period t, κ is a positive constant and α is the learning rate. We use a mofidied version of this algorithm to maximize profitable trades and reduce variance in reproducing strategies.[1] In execution, we train the learner on the data from previous year and test on the subsequent year. If a stage is visited before, we refer to the policy map for the best strategy; if not, we do nothing. Note that maintaining equal training and testing windows does matter since the reward function is designed to maximize the final utility in the training set, so we

[1] Since we use ϵ-greedy sampling and training data is rather limited, variance exists for training and testing on same data sets.

had better use a test window of equal size. In backtesting, we train a Q-matrix using one-year data and apply it on the following trading year. Following is an example we obtained from applying the algorithm to IVV (S&P 500 ETF):

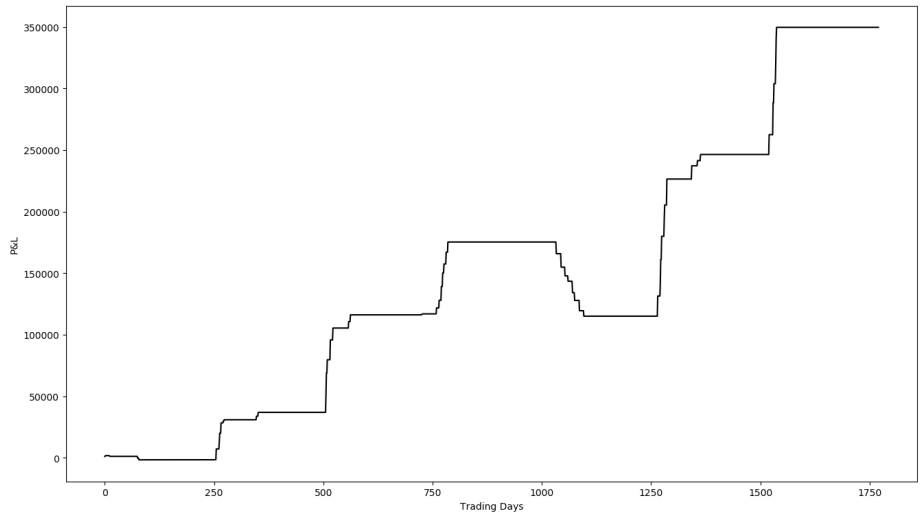

Figure 2: Cumulative Profit of Trading IVV Since 2010

We define the profit generated by a trade as $a_t \times (p_t - p_T)$ if $a_t < 0$, $a_t \times (p_T - p_t)$ if $a_t > 0$. a_t is the amount traded at time t by the trade, p_t is the stock price at time t, and p_T is the terminal price when we liquidate our position. In this case, the terminal price of the trading year on which we test. Transaction costs for assuming a position and liquidation are modeled.

To evaluate the effectiveness of our trading algorithm, we calculate win rate (number of profitable trades divided by number of loss trades) and profit factor (gross profit divided by absolute value of gross loss) on a daily Here are some results we obtained from applying the algorithm to equities, using data from 2015 Jan to 2018 Dec (except for MFST, IVV and GLD, which we use data from 2010 for better generalization, ITOT is from 2010 Jan to 2015 Dec):

	FB	GOOGL	AAPL	XOM	MSFT	ITOT	IVV	GLD
Win Ratio	100%	100%	100%	96%	69%	69%	79%	61%
Profit Factor	NA	NA	NA	144.13	8.5	22.4	6.5	1.58

Table 1: Performance Evaluation

The only reason for not using ten-year data for the other stocks is that they have stock split

between 2010 and 2018. Another caveat worth noting is that we also do not include data from the 2008 recession, because our fundamental assumption requires (approximate) homogeneity in distribution between training and testing set. Therefore this strategy will not work best if we encounter a recession, since the stock price distribution and distribution of deviation from moving average should be drastically different. Since in reality we only do one trade a day at most, if this is ever put into work, the algorithm can be terminated without much loss when recession hit.

This strategy is easy to implement and the result is realizable even for retail traders, as we at most do one trade a day when the exchange open, and trading cost is also modeled in the backtest. Furthermore, as we only keep long position, we do not need to worry about shorting cost and additional opportunity cost for putting up a margin.

3.2.2 Q-Learning with Intraday Trading

We have shown in the previous section that Q-learning can be applied to daily data with very simple action and decision rules: one trade a day when the exchange opens, and liquidate current position at the end of trading period (252 trading days). Since our trades generate positive return on average, we would expect that more trades will lead to more profit. Therefore a very promising research direction is to apply the algorithm on intraday data.

However, there are a few significant deterrences in producing a real profitable strategy in intraday trading through reinforcement learning: 1. Although we have more training samples, we are also trading at a more frequent basis. In fact, we trade at a minute base and the transaction costs can erode the paper-trading profit. With some relief, we still model the transaction cost, but the accuracy of our cost model is less in an intraday setting. 2. We do not have the actual trailing price at intraday level, but only the trades that happened. It is very possible that lots of the trades occurred are results of market-making and have very small sizes, and we cannot trade at such price or liquidate our position for large size. 3. Our reward function assumed independence between different periods of return, but higher correlation exists when the data is more granular. However there are still some advantage in applying reinforcement learning to high-frequency data. Firstly, we have more training samples, thus a better chance to capture trading signals. Secondly, we can incorporate more market information in Q-matrix. Since we have more training examples, the augmented Q-matrix will not be

too sparse. Last but not the least, the success of our algorithm relies on the similarity of distribution between training and testing data, so training on a day and test on the subsequent day satisfies our assumption more than training on a year and testing on the subsequent year. Last but not the least, as we use more granulated data, we are expect to see more triggers - situation similar to past scenarios happen again, and therefore execute more trades. Of course, the advantage is based on the premise that we have an efficient representation of state space that produces more true trading signals than false positive ones.

In this section we only show one trial of application to intraday data to demonstrate the potential of studying granular data, especially when more detailed information is available. We use actual intraday trading data (trades that occured) of IVV in June 2014. Due to a lack of information on the direction of trade, we penalize each trade with additional trading cost. Nevertheless, the result obtained here may not be realizable since the algorithm could be buying at ask price and selling at the bid, even though transaction costs are modeled. We only present no more than a possibility of applying reinforcement learning to high frequency data here, and if supplied with more complete data, the algorithm should be able to handle it. Ideally if we can reconstruct the orderbook, reinforcement learning can output more accurate result, and can be applied to other problems such as optimal execution [6]. Here is the result of trading IVV at a intraday level, where we train on a given day and test on the subsequent day, repeating for the whole month:

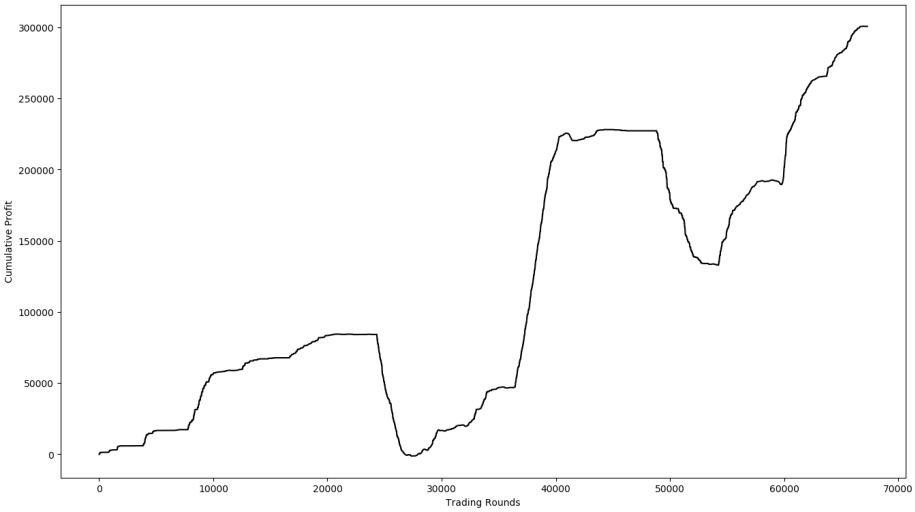

Figure 3: Cumulative Profit of Trading IVV in Jun 2014

Different from daily data, where we use the difference between trading price and the last price in test window (last day of trade) as terminal liquidation price to indicate profit, we use the average of the last two prices as terminal liquidation price. We do so to mitigate the possibility that the price of the last trade might not be a price at which we can liquidate our position.

4 Interpretable Reinforcement Learning For Finance

One problem for implementing reinforcement learning in finance is its interpretability, since we do not follow a simple decision rule which sells at high positive deviation from moving average and buys at high negative deviation from moving average. What we do is to let the learner tries historical data, and apply the information from past data to future decision making. Namely, we learn from past deviation from moving average, and observe reward from such decision. When we arrive at such deviation from moving average in the future, we trace back to the decision map and select the best decision learned in the past. If the distribution in the past is similar to distribution in the future, we expect to make profits.

In fact, our algorithm output a systematic policy based on the distribution of training data, and should not be treated like a blackbox. In this section, we attempt to explain the behavior of our

trading agent. We take the trading result for IVV from 2010 as an example,[2] and plot the trades in test period, with x-axis being deviation from moving average, and y-axis being the *immediate* profit from trade. Linear models are fitted by `statsmodels.OLS` in python 3. Following are the results from regression:

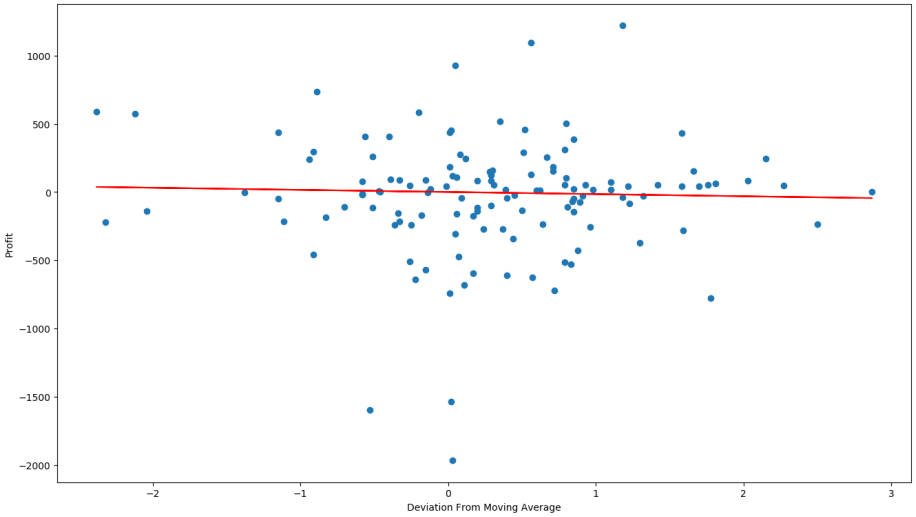

Figure 4: Buy Direction

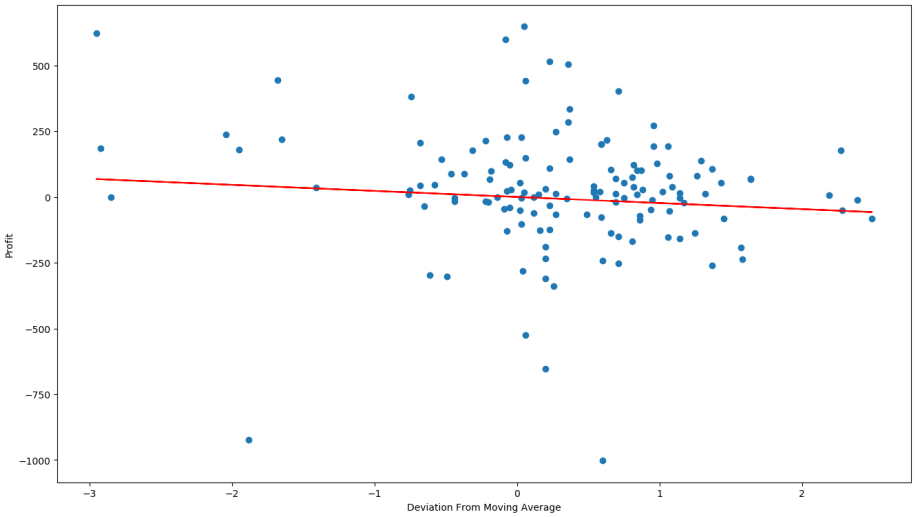

Figure 5: Sell Direction

[2]We use a new run, which will be slightly different due to ϵ-greedy learning, but our modified algorithm control this variance by increasing n.

	coef	t	$p > \|t\|$	no.observations
Buy Direction	-15.57	-0.39	0.69	130
Sell Direction	-23.06	-1.16	0.25	136

Table 2: Regression Results

Since our decision rule is actually non-linear, we do not expect to see a significant linear relationship between deviation from moving average and immediate profit. It is almost impossible to see why we should make a trade at all in short horizon, and there is almost no linear relationship between deviation from moving average and immediate profit in the buy direction at all. How ever, we see some weak linear relationship in the sell direction and the positive coefficient is what we expect: the trades are betting on mean reversion.

To further interpret the implication of a trade, we want to inspect the relationship between deviation from moving average and the lifetime value of a trade, i.e., $a_t \times (p_t - p_T)$ if $a_t < 0$, $a_t \times (p_T - p_t)$ if $a_t > 0$. In this case, both regression gives more significant result:

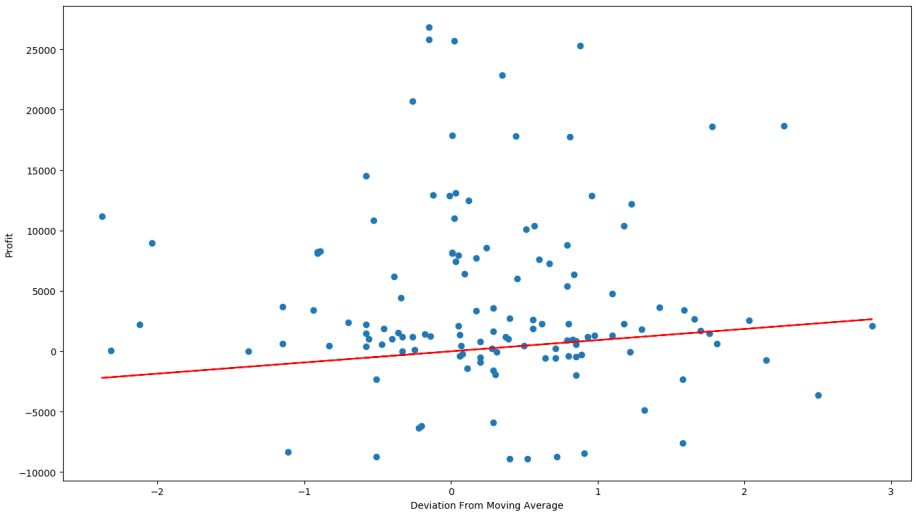

Figure 6: Buy Direction Life Time Value

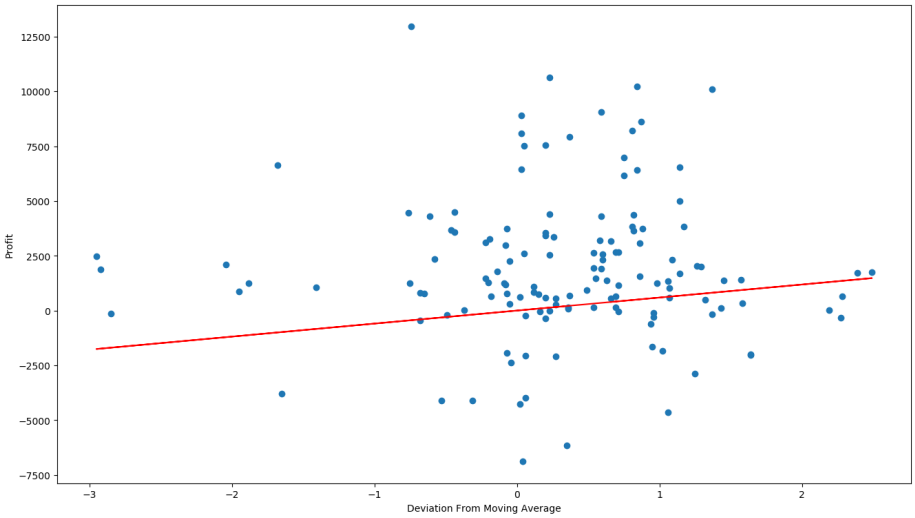

Figure 7: Sell Direction Life Time Value

	coef	t	$p > \|t\|$	no.observations
Buy Direction lft	924.8	1.21	0.23	130
Sell Direction lft	594.4	1.82	0.08	136

Table 3: Regression Results

Since we choose reward function to maximize final wealth, it makes sense that lifetime value has a more significant linear relationship with the state space, compared to the immediate change in wealth. We also observe that buy direction trades weakly follow a momentum strategy, and sell direction trades follow a mean reversion strategy [1][8]. More specifically, the positive coefficient in the buy direction indicates a momentum strategy: we buy when the deviation is high, betting that it will go even higher; the positive coefficient in the sell direction indicates a mean reversion strategy: we sell when the deviation is high, betting that it will revert back to the mean. We have to point out that the linear relationship in buy direction is not significant, so momentum may not be at work at all; but the sell direction is significant at a 0.1 level. The difference between significance in buy and sell direction might be partially explained by our representation of state space: deviation from moving average, and thus a mean reversion strategy should be more obvious. One interesting observation is that most buy direction trades happen between -1 to 1 deviation from moving average.

Regrettably we have no further interpretation with respect to why the algorithm is good at spotting

those trades, beyond the fact that we chose a reward function which aims to maximize terminal wealth. Now referring back to the theory of reinforcement learning, we know that Q-learning actually converges to the optimal policy, which is the final wealth maximizing policy. If the distributions in the training and testing sets are identical, we are expected to see better results. However, since we outperforms the benchmark, we infer that the distributions from two data sets are actually similar.

5 Discussion

In this paper, we show the validity and profitability of applying reinforcement learning techniques in finance. Most importantly, we show that the output policy follows economic intuition from well-established theories. In retrospection, we identify three major area of potential research and improvement: 1. State Space Representation. 2. Reward Function. 3. Learning Algorithm.

The economic intuition of the learner is largely a result of our representation of state space, and therefore a future research direction can be better state space representation. We use a very simple definition - deviation from moving average - as a representation, while certainly other market information such as volatility can be considered. With comprehensive high frequency data and fully reconstructed order book, use market microstructure data will also increase the effectiveness of Q-learning [6].

Another potential research direction is the choice for reward function. Our implementation assumes independence between stock prices in different period, while this is usually not the case in reality. Since reward function is essential for the correctness of our learning algorithm, which aims to maximize the defined reward, out of sample testing result will improve with a reward function that can better capture a real trading scenario.

Lastly, the algorithm applied in this paper can be improved further. Our modification on Q-learning is rather rudimentary, compared to many advancements in this field. Recent development in reinforcement learning includes experience replay [2][5], which can further randomize the training data and reduce correlation between observation sequence, thus leading to better fit with the reward function's premise and better generalization. Provided enough training data, memory replay should further improve the trading result, for both daily and intraday cases.

Moreover, possible improvement on the current trading algorithm also includes creative use of learning process and synthesize reinforcement learning output with other factors to make final trading decisions. When should we cut loss, and what kind of assets should we apply the algorithm to, under what market condition? How should we optimize training and testing window? Reinforcement learning answers none of these questions. Better risk control need to be implemented in other ways, but we have shown that integrating the output policy from reinforcement learning into decision-making does lead to some good trades.

A Daily Trading Result

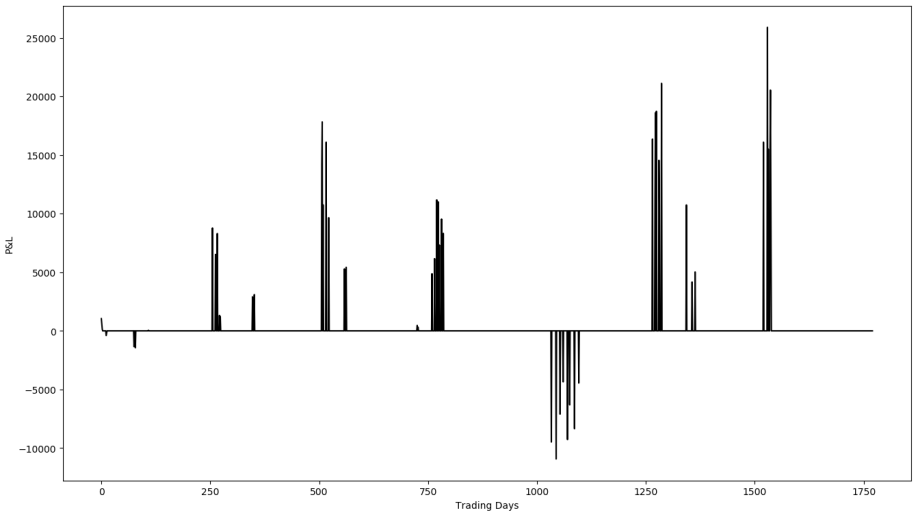

Figure 8: IVV Trading Profit, 2010 Jan to 2018 Dec

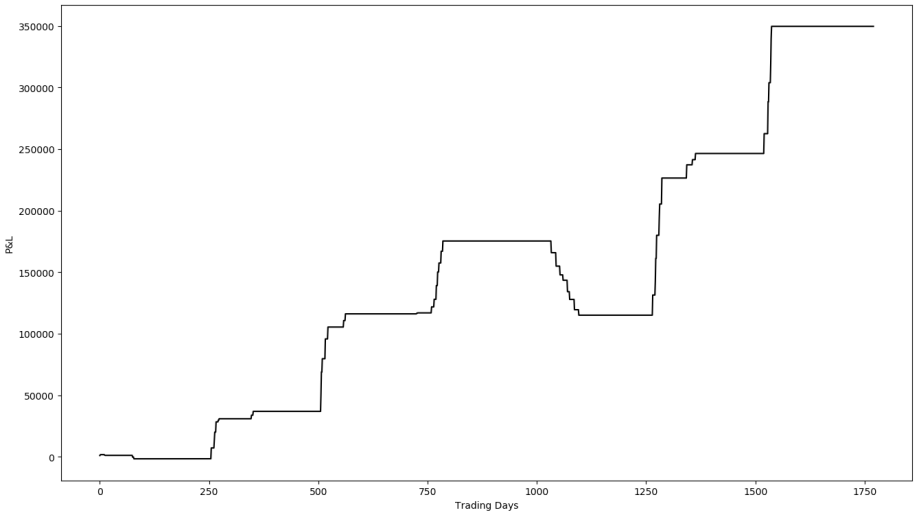

Figure 9: IVV Cumulative Trading Profit, 2010 Jan to 2018 Dec

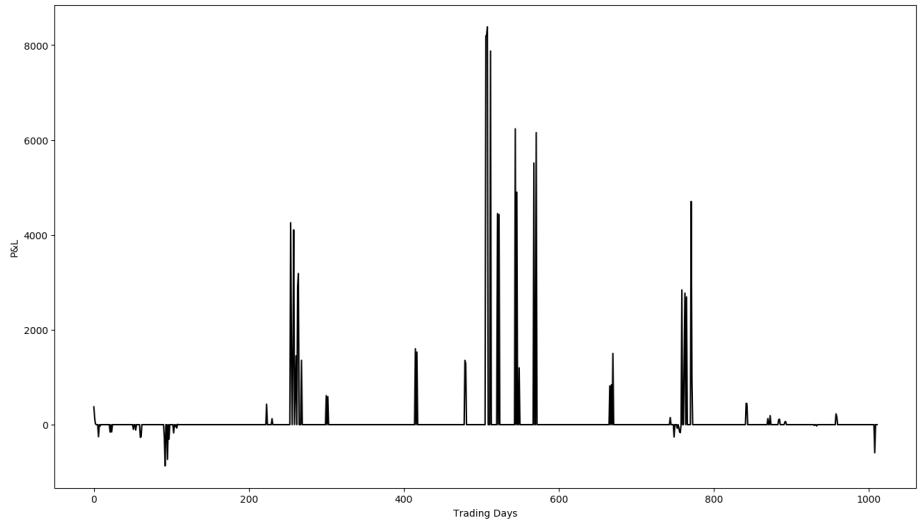

Figure 10: ITOT Trading Profit, 2010 Jan to 2015 Dec

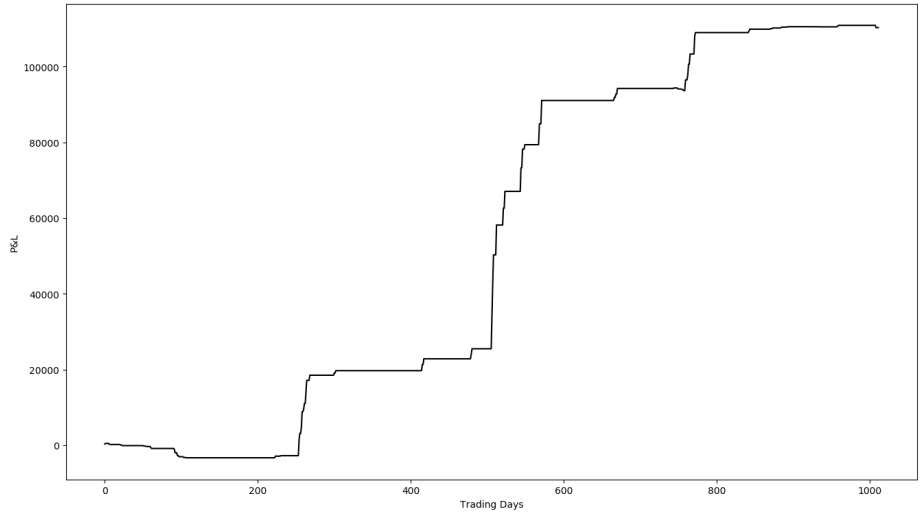

Figure 11: ITOT Cumulative Trading Profit, 2010 Jan to 2015 Dec

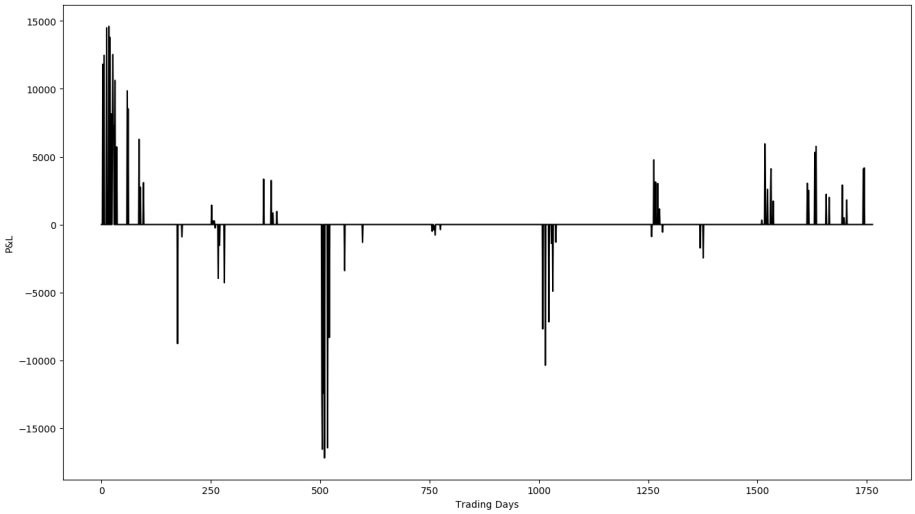

Figure 12: GLD Trading Profit, 2010 Jan to 2018 Dec

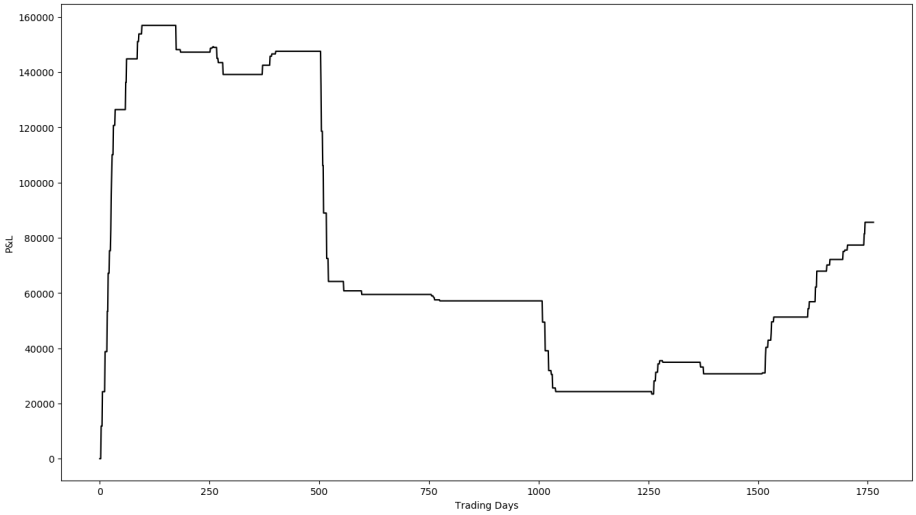

Figure 13: GLD Cumulative Trading Profit, 2010 Jan to 2018 Dec

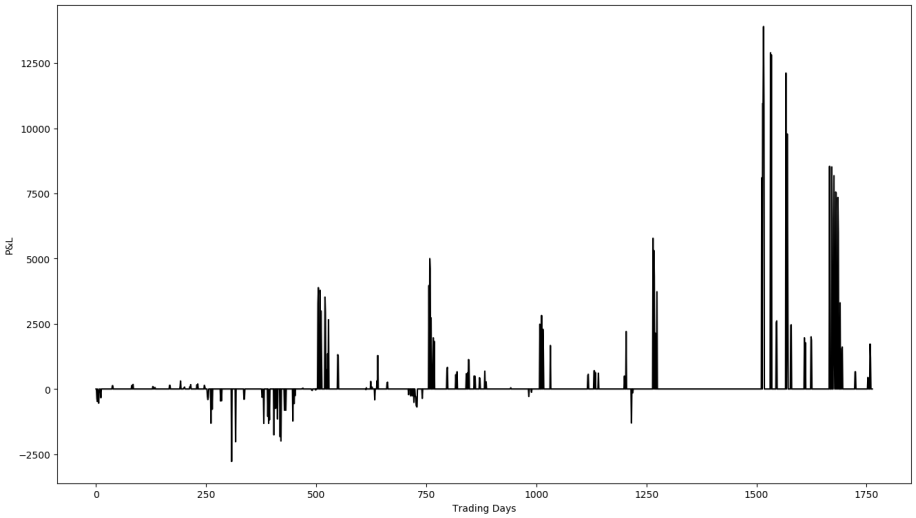

Figure 14: MSFT Trading Profit, 2010 Jan to 2018 Dec

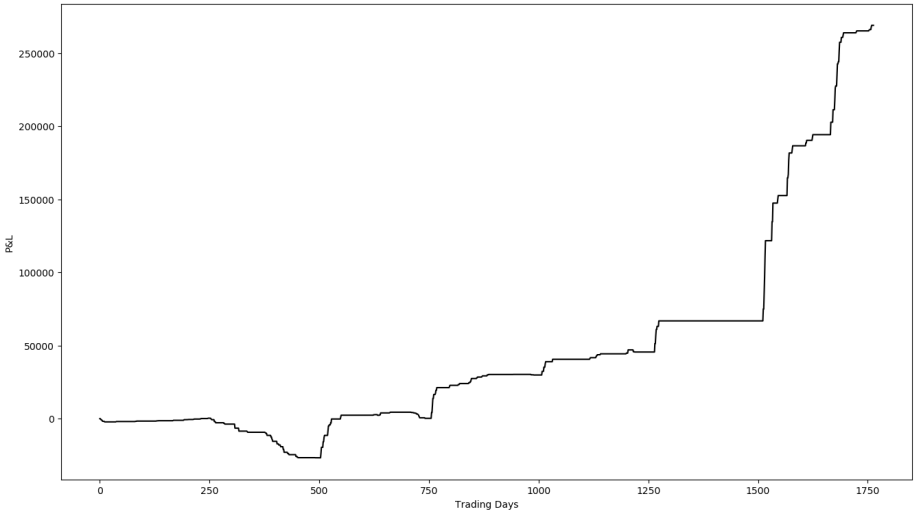

Figure 15: MSFT Cumulative Trading Profit, 2010 Jan to 2018 Dec

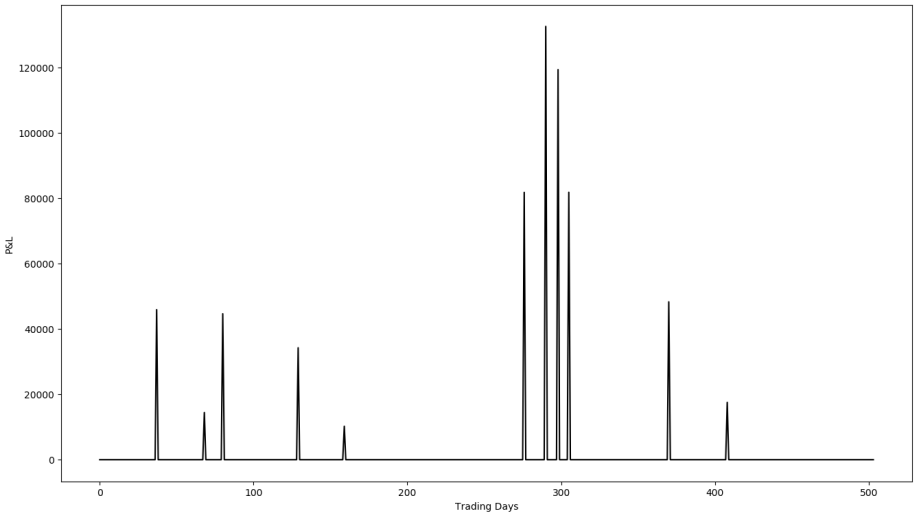

Figure 16: GOOGL Trading Profit, 2015 Jan to 2018 Dec

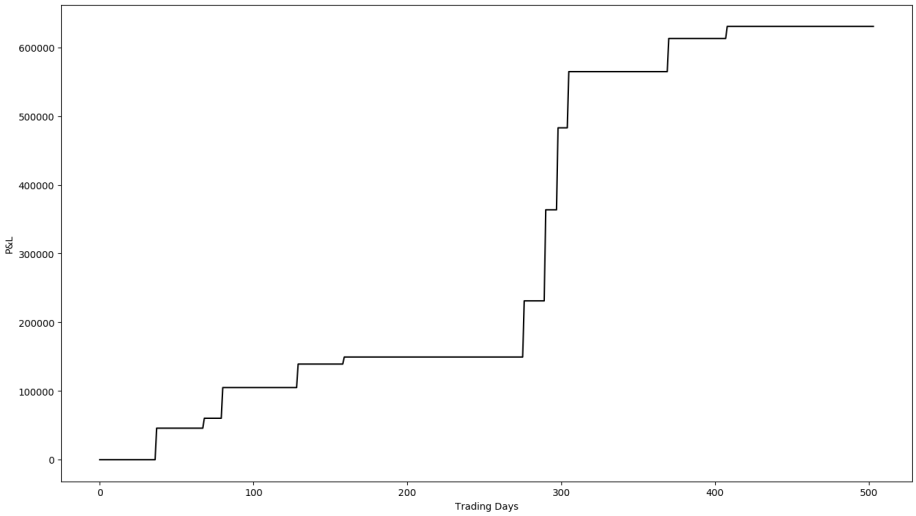

Figure 17: GOOGL Cumulative Trading Profit, 2015 Jan to 2018 Dec

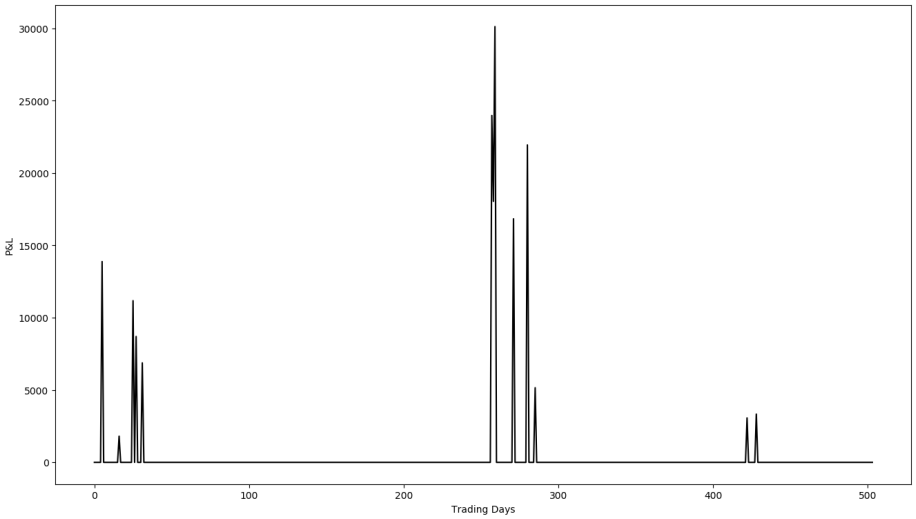

Figure 18: FB Trading Profit, 2015 Jan to 2018 Dec

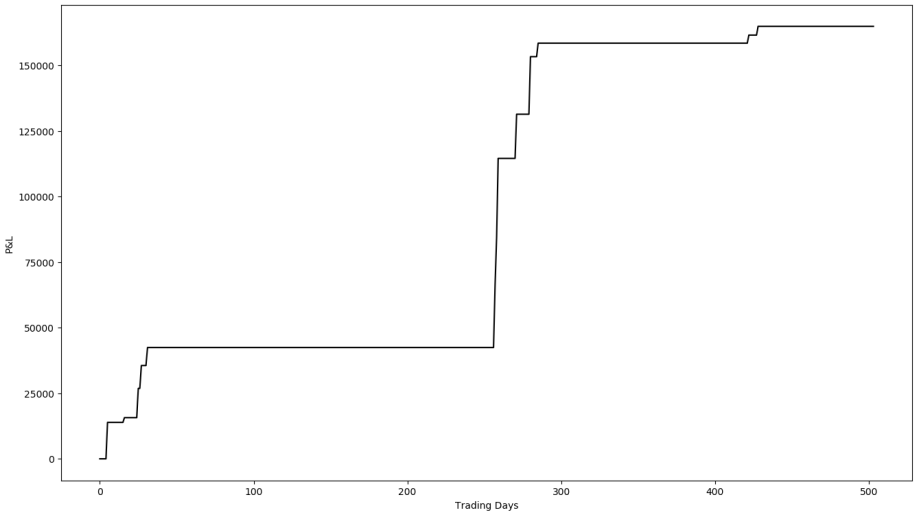

Figure 19: FB Cumulative Trading Profit, 2015 Jan to 2018 Dec

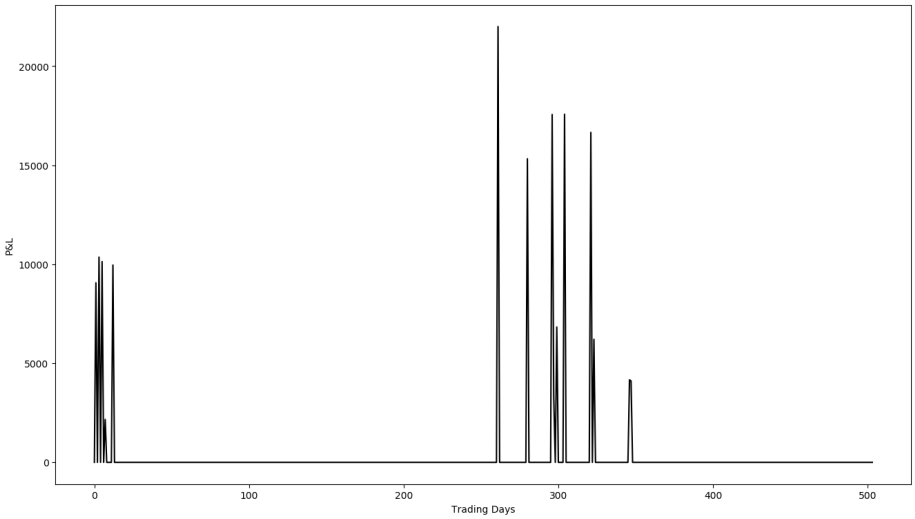

Figure 20: AAPL Trading Profit, 2015 Jan to 2018 Dec

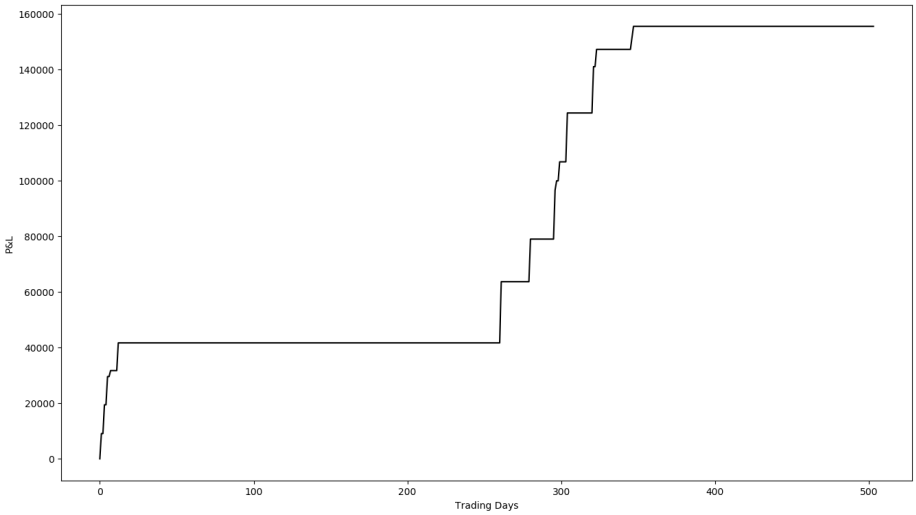

Figure 21: AAPL Cumulative Trading Profit, 2015 Jan to 2018 Dec

References

[1] Fama, Eugene F., and Kenneth R. French. "Common Risk Factors in the Returns on Stocks and Bonds." *Journal of Financial Economics* 33, no. 1 (1993): 3-56. doi:10.1016/0304-405x(93)90023-5.

[2] Kaundinya, Varun, Shubham Jain, Sumanth Saligram, C. K. Vanamala, and Avinash B. "Game Playing Agent for 2048 Using Deep Reinforcement Learning." *Nciccnda*, 2018. doi:10.21467/proceedings.1.57.

[3] Ritter, Gordon. "Machine Learning for Trading", 2017.

[4] Markowitz, Harry. "Portfolio Selection." *The Journal of Finance* 7, no. 1 (1952): 77-91. doi:10.2307/2975974.

[5] Mnih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Rusu, Andrei A, Veness, Joel, Bellemare, Marc G, Graves, Alex, Riedmiller, Martin, Fidjeland, Andreas K, Ostrovski, Georg, Petersen, Stig, Beattie, Charles, Sadik, Amir, Antonoglou, Ioannis, King, Helen, Kumaran, Dharshan, Wierstra, Daan, Legg, Shane, and Hassabis, Demis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[6] Nevmyvaka, Yuriy, Yi Feng, and Michael Kearns. "Reinforcement Learning for Optimized Trade Execution." *Proceedings of the 23rd International Conference on Machine Learning - ICML 06*, 2006. doi:10.1145/1143844.1143929.

[7] Richard Sutton and Andrew Barto. *Reinforcement Learning: An Introduction*, "Complete Draft", MIT Press, 2017

[8] Poterba, James, and Lawrence Summers. "Mean Reversion in Stock Prices: Evidence and Implications." 1987. doi:10.3386/w2343.

[9] Watkins, Christopher J. C. H., and Peter Dayan. "Q-learning." *Machine Learning* 8, no. 3-4 (1992): 279-92. doi:10.1007/bf00992698.