Q-Learning the Market

Tanay Trivedi[1], Padraic McAtee[2] and Anita Raja[3]

***Abstract*— Cryptocurrencies and the trading of virtual assets represents a new front for both day traders and electronic trading firms. Machine learning as a tool in automated trading has matured and several past works have used a specific field called reinforcement learning to optimize trading actions in short horizon time spans. This work takes advantage of cryptocurrency exchanges open, free and extensive historical trading data to train a reinforcement learning agent to trade Bitcoin at medium frequencies, producing optimized trades.**

I. INTRODUCTION

Trading financial assets presents a difficult problem in numerical modeling: fat tailed distributions, noisy data and the resulting numerical issues, and dark market liquidity makes it difficult to apply the techniques developed in the fields of statistical modeling to predict future returns. As a result of these issues, many professionals in the industry have looked to alternative methods of modeling that use systematic learners rather than humans to interpret intra-day market data, understand patterns and provide the optimal trade action. Among these alternative methods is reinforcement learning, which provides an environment where a learner can interact with a problem by gaining reward as the result of taking an action so that it may optimize future actions [1].

Among the many different algorithms in the context of reinforcement learning is Q-Learning, which has been used in a number of studies in applying learning methods to finance. It has been shown in studies by Ritter [2] and Balch [3] that through repeated training of a Q-Learner over a fixed range of historical market data, the learners made considerable returns in out of sample testing. This paper presents a Q-Learning implementation that draws from work in both the previously mentioned papers.

II. TRADING SYSTEM AND PORTFOLIO SETUP

A. The Agent

Given that the desired trading system avoids complicated quantitative methods requiring extensive modeling and time, the agent would ideally operate in an environment where it has easy access to data as well as short term feedback on it's performance. With this, the was developed to emulate the behavior and intuitions of a day trader.

[1]T. Trivedi is a student in the Albert Nerken School of Engineering, Cooper Union for the Advancement of Science and Art, 41 Cooper Square, New York, NY 10009 trivedi@cooper.edu

[2]P. McAtee is a student in the Department of Mechanical Engineering Cooper Union for the Advancement of Science and Art, 41 Cooper Square, New York, NY 10009 mcatee@cooper.edu

[3]A. Raja is a Professor of Computer Science and Associate Dean of the Engineering Cooper Union for the Advancement of Science and Art, 41 Cooper Square, New York, NY 10009 araja@cooper.edu

Day traders will use regularly updated market data to base their actions on throughout a single trading day. This data is readily available and may include metrics such as price, price change and indicators. Our desired agent follows suit, making decisions based mostly on frequently updated market data that is easily accessible to all market participants. This data is usually updated in 1-minute, 5-minute, or 15-minute intervals. The data represents aggregations of the continuous forms of these metrics into 1, 5, or 15 minute buckets. As a result, the agent does not perform any historical aggregation of the data it sees during operation, limiting its look back to that of the data it observes. For example, if the agent was to use only 15 minute price change data to make decisions, it is "looking back" at data from no longer than 15 minutes in the past to do so. The agent will not, for instance, maintain a moving average of the most recent four 15 minute price changes, effectively making its look back an hour. During its operation, the agent should also address risk. This is achieved by also taking data from its portfolio over the course of the trading day.

B. Portfolio Structure

One of the pillars of the field of financial theory is the Capital Asset Pricing Model, which outlines a relationship between risk and expected return of an asset [4]. One of the major conclusions of CAPM is the holding strategy of both risky and riskless assets for an investor. The investment account used by the trading agent will follow the CAPM theory and consist of both risky and risk free assets. The portfolio will initially consist of a sum of cash that will be continually consumed and replenished by taking long or short positions of certain magnitudes in a single asset. Long positions are entered by purchasing an amount of the asset while short positions are by selling accordingly.

As a day trader, the agent will close out all of its positions at the end of day and take profit. The agent will begin the following trading day with a neutral position, owning zero shares of the asset at hand.

C. Measuring Performance

The agent's performance is measured by its profit or loss for each individual trade. These values are accumulated from the start of the day to the end, when all positions are closed, in order to calculate a net profit and loss (PnL) for a given trading day.

III. LEARNING METHODS

Reinforcement learning algorithms determine the optimal action to take of a system's possible actions given the

system's state. This is done through random exploration of the state-action space while storing the reward value corresponding to each state-action pair. The reinforcement is modeled as a Markov decision process, with the set of states S, the set of actions A that the agent can take, a policy $\pi : S \rightarrow A$ which recommends the action A to take given a state S, a function $P_a(x, y)$ giving the probability of the agent's transition from state x to y via action a, and a function $R_a(x, y)$ similarly denoting the reward of that transition [1]. Reinforcement learning algorithms are

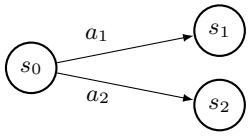

Fig. 1. A simple example of a Markov Decision Process with $S = \{s_0, s_1, s_2\}$ and $A = \{a_1, a_2\}$.

classified as either *value searching* or *policy searching*. In value searching algorithms, the reward value dictates whether the taken action in the particular state had good or bad results. This allows the learner to get feedback on optimization directly from the environment it operates in as opposed to supervised learning methods, requiring some human interpretation of the data.

The value function $V^\pi(x)$ represents the future discounted reward that will be attained from an initial state x. For each iteration through the state-action space, we seek to maximize the value function show in Bellman's equation:

$$V^\pi(x) = \sum_a \pi(x, a) \sum_y p_{xy}(a)[D(x, y, a) + rV^\pi(y)]$$

where $\pi(x, a)$ is the probability of taking action a in state x, $p_{xy}(a)$ is the transitional probability from state x to state y under action a. $D(x, y, a)$ is the intermediate reward, and r is the discount factor weighing relative importance of future rewards to current rewards. The value iteration method is defined as

$$V^\pi_{t+1}(x) = \max_a \sum_y p_{xy}(a)[D(x, y, a) + rV^\pi_t(y)]$$

This iteration converges to the optimal value function defined as:

$$V^*(x) = \max_\pi V^\pi(x)$$

From this, the optimal action can be inversely determined as:

$$a^* = \arg\max_a \sum_y p_{xy}(a)[D(x, y, a) + rV^\pi(y)]$$

A. Q-Learning

Q-Learning is a value searching reinforcement learning method that utilizes a table, denoted $Q(x, a)$, to implement a value function in the state-action space. Q-Learning is a model-free learning method, meaning that it will learn to optimize reward directly from engaging with the environment as opposed to learning a model of the environment's structure. Bellman's optimality equation for Q-Learning is shown as:

$$Q^*(x, a) = \sum_y p_{xy}(a)[D(x, y, a) + r \max_b Q^*(y, b)]$$

The update rule is based on the gradient of error:

$$\frac{1}{2}[D(x, y, a) + r \max Q(y, b) - Q(x, a)]^2$$

, leading to the optimal action for a given state:

$$a^* = \arg\max_a [Q(x)]$$

Among the many reinforcement learning algorithms to choose from, Q-Learning provides the simplest means of training the agent for its desired purpose. The use of a table provides a clear mapping from the real time market data the agent observes to the optimal choice of action at that time.

The Q-Learning algorithm implements a learning policy during training in order to explore the state action space. As it conducts this exploration, the immediate rewards for each of the state action pairs are updated to the table. The algorithm then uses a behavior policy during testing that optimizes reward based on the table.

IV. Learner Configuration

After establishing the theoretical basis of using Q Learning to map input variables to optimal actions and establishing the expected value of these mappings, it becomes a choice of the user to find the appropriate definitions of the following variables:

- State Space- The choice of quantitative or qualitative factors that describe the situation presented to the learner at any time: $s(t)$
- Action Space- The choice of possible actions at any time
- Reward Function- The real valued function that rewards the learner on the optimality of any action
- Transition Probability- A parameter allowing the user to create an environment with some uncertainty of outcomes; usually used in a situation where even if an agent consciously chooses an action it may end up implementing another through some randomness in the environment

Much like other machine learning methods, the configuration of the learner is the non-trivial part of the implementation; most of the work in this research was done in modifying state spaces to achieve a better understanding of the trading problem. As such, the other parameters that were held constant throughout the experimentation will be detailed first, and then the different preliminary state definitions that were used along with the associated results.

A. Action Space

Many cues were taken on the recent work by Ritter, including his version of the action space. Ritter's work originated in trading equities and as such he described the typical actions of a firm in this space: his learner could buy

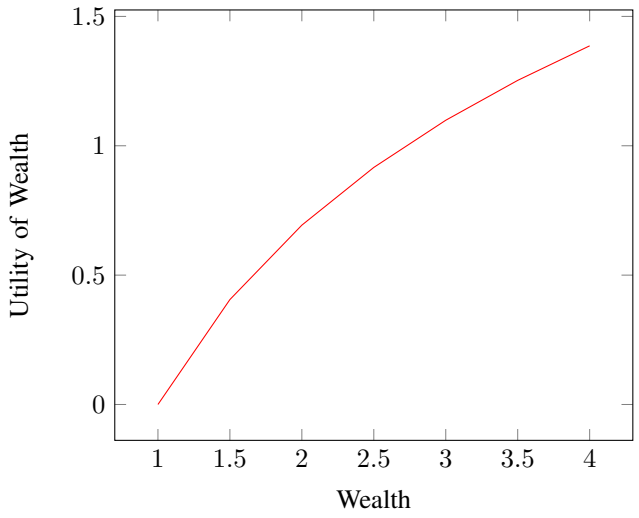

Fig. 2. Wealth Utility Curve of a Risk Averse Investor

or sell up to ten multiples of lots of stock, since firms usually buy in blocks. The action space was therefore:

$$\mathcal{A} = lotsize \cdot \{-K, -K+1, \ldots, K\}$$

where K is the maximum holding amount of the security. The maximum position for this application was also 10, with a lot size of 1 considering the high nominal price of in comparison to stocks.

B. Reward Function

The objective of a trader is to maximize the wealth of his portfolio. Wealth, and more importantly utility of wealth, has had a long tradition in the study of economics. Assuming a risk averse investor (as all rational investors are), the utility of wealth function is graphically depicted as such: Wealth is measured as the accumulation of PnL on top of the initial value of the portfolio. The reason for this concavity comes from the definition of diminishing marginal utility in economics. Ritter defines his reward with the differential wealth utility from action to action. To calculate differential utility of wealth, it is necessary to define differential wealth:

$$\delta v_t = h_{t-1} \cdot r_t - c_t$$

where δv represents the differential value of the portfolio at the current time step, h_{t-1} is the number of securities held at the previous time step, r_t is the current return of the price series of financial products and c_t is the transactional costs at the current step. This is the same as the differential Profit and Loss (PnL) of the agent, with total PnL across an epoch being a summation of differential PnL at every time step. A rigorous proof is given for the reward for any action at time t:

$$R_{t+1} = \delta v_t - \tfrac{\kappa}{2}(\delta v_{t+1})^2$$

Functionally, the differential value is the same as the differential PnL across trades.

C. Transition Probability

In some environments, it is a significant concern that even when consciously making a decision an agent may not be able to execute the action it intends on, thereby putting it into an unintended state. Reinforcement learning allows for this by incorporating transition probabilities into the Markov Decision Process. In the world of financial asset management, there is a situation may require a modeling of such an environment If the learner was required to maintain its own Order Management System (OMS), which handles posting orders to the exchange and recording relevant information: if orders aren't accepted by the exchange for a variety of reasons (such as Internet outages, exchange overloads, flash crashes), any position or exposure related state space variable may take on a form that is not the optimal action. One can model this in this way:

$$a(x{+}1) = \begin{cases} a^* = \arg\max_a[Q(x)] & \text{with probability } p \\ a^* = randomaction_a[Q(x)] & \text{with probability } 1-p \end{cases}$$

In this way, a system that for a small probability behaves irregularly or random can be learned from the ground up to account for such irregularly.

This sort of ground up robustness isn't always necessary since trade execution is handled separately and persistence in achieving target portfolios is built in at a very high frequency in state of the art OMS. Some incredibly high frequency learning systems may see it as necessary, because of the immense amount of feedback from competitors low latency trading systems in the the market at the same time. Our application wasn't operating in this domain, so we ignored the transition probability side of this configuration and simply kept it at $p = 1$, meaning that any action that our learner desired executed as designed.

D. Price as a State

Another initial inspiration the authors took from Ritter's publication was his state space definition: he discretized prices to the hundredth's place and made every cent its own entry in the price dimension. This allowed his learner to understand support and the lack of it at certain prices in the time series.

Ritter also added another dimension for cumulative position. It allowed the learner to take anywhere from five long to five short positions at any time, making the cumulative position dimension eleven items long:

$$\mathcal{H} = \{-M, -M+1, \ldots, M\}$$
$$\text{w/ cardinality } |\mathcal{H}| = 2M+1$$

Discretization of this price dimension became a significant issue, as the price of Bitcoin ranged from $600 to 4600 in the date range discussed, producing a dimension of ¿400000 indices. This price dimension is created for every value of cumulative position as well to generate the full state space, that is:

$$S = \mathcal{H} \cdot \mathcal{P} \approx 400000 \cdot 11 = 4400000$$

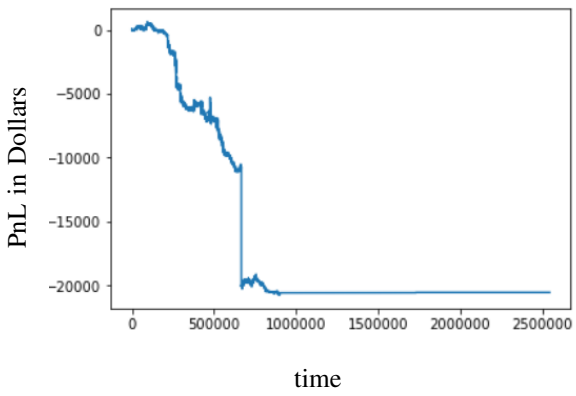

Fig. 3. In Sample PnL of Greedy Learner with Price as State, 600 Runs

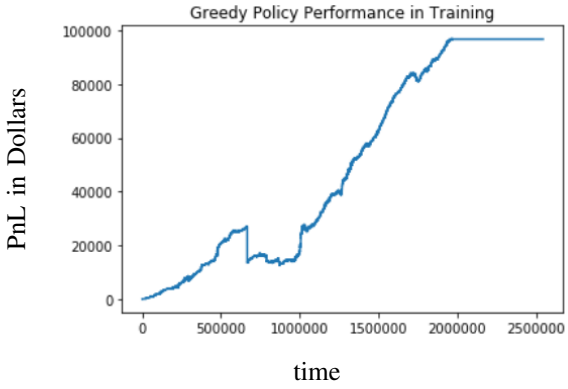

Fig. 4. In Sample PnL of Greedy Learner with Price as State, 2000 runs

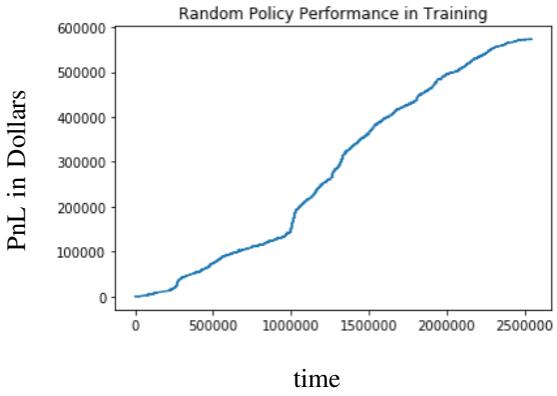

Fig. 5. In Sample PnL of Random Learner with Price as State, 600 runs

This illustrates the famous curse of dimensionality that Bellman encountered when implementing discrete learning via his equations of optimality, and led to the development of many continuous methods of reinforcement learning to realize the advantages of the scheme in the real world. The curse realizes itself when looking at Figure 3, which depicts the in-sample performance of the learner after 600 runs over the entire dataset. After investigating the state space, it was clear that the policy used to explore the space in training was too hasty in choosing the optimal action, thereby not filling necessary states quickly enough with training. It also easily observable in Fig. 3; when the PnL flat lines it indicates that the learner has no understanding of the state it is in and choosing to do nothing.

It is probable that Ritter did not encounter this issue in his implementation due to two reasons: (1) his pricing data was manufactured to be have the interval [0.1 100], making his state space significantly smaller than ours and therefore easier to test till convergence and (2) his implementation in Java ran at almost 1 million iterations per second, which was orders of magnitude faster than an one in Python which was used for this project. This allowed his learner to more quickly run through the time series, so it trained with considerably faster speed. Indeed, once the implementation of a greedy policy was given more iterations to run in Python, it did create a vastly superior in-sample performance, as seen in Figure 4. Even in this implementation and with that many runs, the flatlining at the end indicates it did not reach the entire state space in training and left much to be desired.

Seeing as the intention was to implement the relatively simple Q-Learning method and instead optimize its parametrization, a change was made in the behavior policy to enhance the utility of Q-Learning: switching from a greedy behavior in training to a random behavior. This change makes the Q-Learner more akin to a Monte Carlo simulation, which moves randomly from state to state, learning the immediate repercussions of its actions and coherently understanding his state. This produces a superior in sample performance, as seen in Figure 5. It neither flatlines nor does it jump like the Greedy Learner. Based upon this, greedy training was never used again for the Q-Learner, and from here on all results shown are for random learning.

The out-of-sample (OOS) performance became a little difficult to evaluate because of the near constant monotonicity of the long term price movement of Bitcoin in the in-sample period chosen. Figure 6 is a plot of price in the in-sample period:

Because the price is a state now, with every price down to a hundredth's place defined in the Q-table, we must necessarily require that down to a cent every price seen in the OOS be one seen in the In Sample. This is not the case: the price series used for this analysis was drawn from the exchange at the second frequency for approximately 14 months, but price and price changes are discontinuous, quantized quantities, making it unlikely that in the 5000 second out of sample every single price observed is one that has been observed exactly before in the previous year. Furthermore, it does not help that the OOS period is directly after an upwards rise in prices, so the series has visited that region only once before. To combat this, the authors used two different strategies:

1) Passive: In this methodology, if the learner observes a state it is unfamiliar with, it will simply take no action until it falls into one it is familiar with. This is less illogical that it would seem: analysis of the Q-table shows that for prices that the learner has observed in the in-sample period, the random learning policy did well with observing every cumulative position

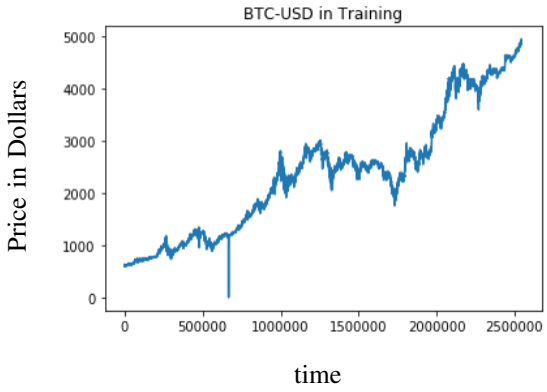

Fig. 6. In Sample Price of Bitcoin

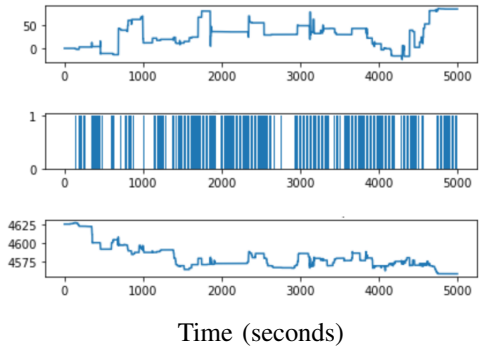

Fig. 7. OOS PnL of Random Learner with Price as State, Passive

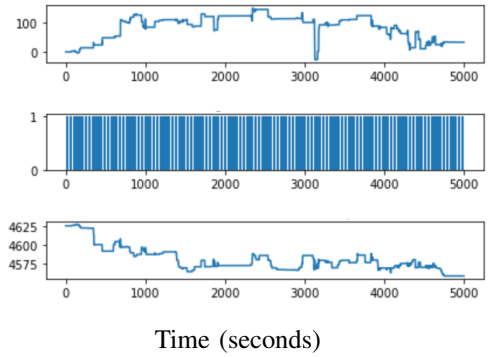

Fig. 8. OOS PnL of Random Learner with Price as State, Price Snapping

possible. This means that the only instances when the learner observes an undefined state is when the price has never been seen before, and it follows that the learner will not be static for that long before a price in the series is observed with some definition. The results of the strategy are seen in Figure 7. The top plot represents PnL in the Out of Sample period, the second is a plot showing when the learner did not recognize a state and when it did, and the final plot is showing the price in the Out-Of-Sample period. The second plot has a blue bar in parts of the time series where a state was recognized, and white where there was no state and the learner was sitting and waiting.

2) Price Snapping: This method is very simple functional approximation tool that, after training the Q-table on the In Sample set, walks through a list of unique prices in sample and indexes into the table, checking the level at which the 2 dimensional $\mathcal{H} \times \mathcal{A}$ is well defined. For the purpose of this analysis, the requirement was that 50% of that space had non-zero values. The price levels that filled this requirement were collectively called the "snap prices", and if a price level in the OOS testing was undefined the strategy would find the near snap prices and take the optimal action according to that price level. This is somewhat similar to clustering, but in a one dimensional sense. Figure 8 details the results in the same format as Figure 7.

It was clear that using price as the state definition was suboptimal for performance, and it makes sense that it should: asking a learner to fit to price would be assuming that prices OOS move in the exact same order and way as prices in-sample. This is not the case with Bitcoin, or indeed any financial asset. Ritter acknowledges this in his publication, as his excellent OOS PnL is only due to the fact that his data is manufactured from a Ornstein-Uhlenbeck process that is added to a linearly increasing price series from [0.1 100]. The deviations from the linear price are driven by the following differential equation:

$$dx_t = -\lambda x_t + \sigma \xi_t$$

where x_t is the current price, λ is the mean reversion parameter, σ is the noise standard deviation, ξ_t is an independent observation sampled from a sample normal distribution. Real world financial prices, and especially Bitcoin's irrational exuberance, do not follow any such equation driven dynamics. His publication acknowledges that for real markets, learners need more nuanced information than that provided by just price.

In addition, having unnormalized indicators like price have another significant disadvantage other than bad OOS performance: a trainer trained on an interval of prices $[x\ y]$ can only be used on an interval $[a\ b]$ if and only if $a \geq x$ and $y \geq b$. If this condition is not met, obviously the learner will have no idea as to what future price movements will occur because it has never seen those prices before. This obviously leads to the conclusion that only indicators and quantities that are normalized or bounded in some way should be used in a state definition for a direct reinforcement learning application with time series data.

E. Indicators as States

With this in mind, the state definition was changed to accomodate two new indicators. As the goal of the project was to emulate a day trader, the two new indicators are common in the industry: Relative Strength Index (RSI) and Stochastic Oscillator. More common in the industry are multiple moving averages with different look back periods, but unfortunately these do not meet the requirement created

at the end of the previous section. Instead, RSI and the Stochastic Oscillator have ratios built on the Open-High-Low-Close (OHLC) structure of sampled financial data. Say that we requested from the exchange the OHLC prices at the second frequency. We get a table with the following columns and rows:

Epoch	Open	High	Low	Close	Volume
1512519240	457.22	457.96	457.35	457.23	180.42
1512519180	457.21	457.98	457.47	457.34	206.61
⋮	⋮	⋮	⋮	⋮	⋮

- Open- The price at which the first trade occurred in the current interval.
- High- The highest price at which a trade occurred in the current interval.
- Low- The lowest price at which a trade occurred in the current interval.
- Close- The price at which the last trade occurred in the current interval.
- Volume- The total volume traded in the current interval.

The RSI equations look like this:

$$RSI = 100 - \frac{100}{RS} \quad G = \frac{\sum_{winningtrades} PnL}{\#winningtrades}$$
$$L = \frac{\sum_{losingtrades} PnL}{\#losingtrades} \quad RS = \frac{G}{L}$$

The Stochastic Oscillator equations look like this:

$$SO = \frac{100(Close - Low14)}{High14 - Low14}$$

where the $Low14$ represents the lowest price in the last 14 time steps, $High14$ is the highest price in the last 14 steps and C is the current close price. In this way, it is possible to obtain SO and RSI for the time series data we are considering. For the purposes of quick training and reducing over fitting, the SO and RSI indicators were discretized to the singles digit, making each axis length 100.

Now the state space definition looks this:

$$\mathcal{S} = \{CP, SO, RSI\} \text{ with length}$$
$$|\mathcal{S}| = 21 \cdot 100 \cdot 100 = 210000$$

with CP being the cumulative position of the learner at that time. The results of using this learning configuration in-sample are shown in Figure 9, while the prices in chosen OOS periods are shown in Figure 10 and performance out of sample are shown in Figure 11. Since the method was no longer bound by having OOS prices be in the same range as the IS prices, three different OOS periods were chosen: one that was an upwards trending market, another that was a downwards trending market, and finally one that had no clear trend and instead oscillated up and down.

There are several issues obviously apparent with the results: (1) the In-Sample PnL of the learner should be positive, considering that guarantee that Reinforcement Learning makes of convergence, (2) the PnL of the learner OOS is only positive for sideways markets. It should be noted that this learner was not run until convergence, because the learner failed to converge in any reasonable time frame. The results

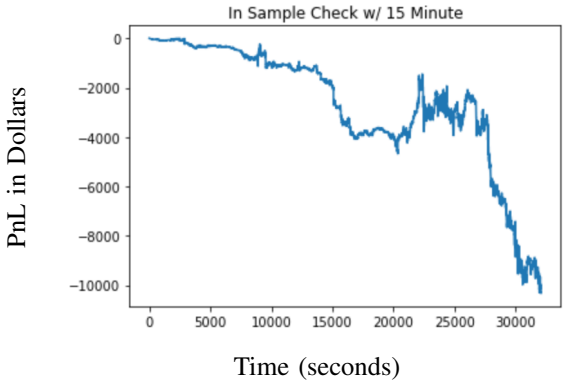

Fig. 9. OOS Price in Varying Market Conditions

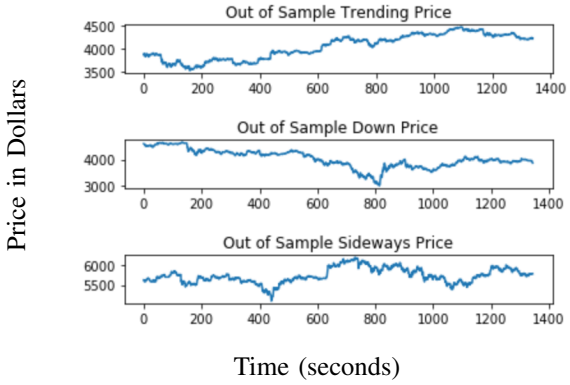

Fig. 10. OOS Price in Varying Market Conditions

were simply taken after an extended period of training. It was suspected that by using a normalized oscillator, the expected reward at each state was being overwritten on each epoch. For example, Figure 12 is a chart of the RSI indicator in one of the OOS periods. Whats clear is that for the majority of the time period, the indicator remains in the same range making the decision process muddled. If both indicators remain at normal levels, the learner only has the cumulative position to differentiate its situation, with no clear indicator of the trend in the market: knowing whether the learner is in a cumulative position of 10 or 5 does not give enough information about

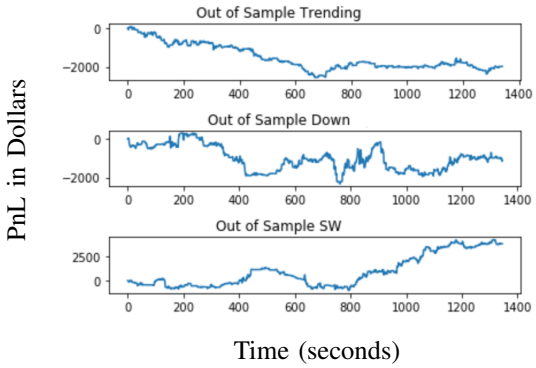

Fig. 11. OOS PnL of Random Learner with Indicator States

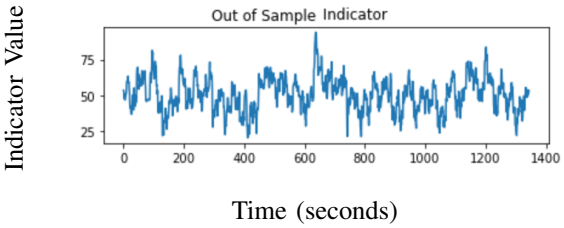

Fig. 12. OOS RSI Indicator Value

the market state. Because the data is overwritten with every epoch, the Q table with the associated average levels of the indicators are constantly oscillating and producing no convergence. When the market trend was sideways, the performance was appropriate because the indicators (which were originally created to capture oscillatory behavior) were outside their normal range, the Q-table unmuddled and the optimal decision was clear. It was clear that a new indicator was necessary to differentiate market regimes, whether that be trending, sideways or downward, so as to make the oscillators more useful in producing optimal trades.

F. Price Difference vs Prices

The primary problem with using price as a state as discussed in Section D is the lack of utility in the OOS period because of the uniqueness of prices and the requirement that the OOS prices be completely inside the range of IS prices. In addition, if we tried to use prices as a proxy for market regime, the deduction necessary is non-obvious and would be akin to predicting future price with only the current price and a few indicators. Price history, then, is what we need to use to quantify the market trend.

A common day trading strategy is to use moving averages of varying length of past prices to evaluate the current price in terms of its history. Unfortunately moving averages of price suffer the same issues as price itself, as they have no maximum and the OOS statistics must be inside the range of the IS training. Another approach of capturing price history is to look at historical price changes, or returns. A well documented characteristic of asset returns is their near normal distribution, which will allow us to set reasonable bounds on the price changes that should capture most price movements. After setting these caps, the price changes observed in one year will include most of the future price changes in our OOS and those that exceed the caps will simply concentrate their informations in the ends of the distribution. Figure 13 depicts the histogram of the price differences in the In Sample period.

The ideal discretization would be one that scales the size of the states with larger ones at the tails of the distributions, where roughly every price change looks the same and happen with less frequency, and smaller ranges for the states near the center so that the high frequency of the price changes in this region are differentiated. To simplify the system, a simple uniform size was chosen for each of 100 states. The distribution was capped at

The new state definition is therefore:

$$\mathcal{S} = \{CP, DIFF, SO, RSI\}$$

This is the final configuration of the learner, the results are discussed in the following section.

V. Results

The data presented in this section shows the agent's performance in out of sample testing after extensive iterations over a fixed in sample training period.

A. In Sample Training

The training period consists of approximately 11 months of Bitcoin data acquired freely through the GDAX exchange. The raw data specifies OHLC and volume amounts for the BTC-USD currency pair at irregular time intervals. To uniformly fill the time dimension, each data entry was filled forward for every second in the time interval until the immediate next data entry. This was done to subject the learner to periods of time without price movement, giving it insight into the time dependency of price change. Forward filling the data also necessary for calculating the indicators which all aggregate data across evenly spaced time intervals of varying length.

For the purposes of this study, the price of Bitcoin was taken as the closing price of BTC-USD for each entry in the forward filled data, giving a continuous time series of the price of Bitcoin for every second in the training set. These prices were then used to calculate corresponding time series for each of the indicators, defining a state S for every second in the training set.

In training the learner on the set, a random action learning policy was implemented in order to explore the rewards corresponding to each state action pair. Starting with a cumulative position of 0, the learning took a random action which adjusted its cumulative position, observed the resulting change in PnL as reward, updated the reward to a Q-table of state-action pairs and proceeded to the next state for every second in the training set. The learner performed repeated iterations through the training set until the state-action space was sufficiently explored. Due to the size of the defined state space, the training iterations were ended before the learner could attain complete exploration of the rewards for each state-action pair. These unexplored states could be identified by locating values in the Q-table that had not been updated in training. An issue arises when the agent finds itself in an unexplored state as there is no mapping to the optimal action for that state. The next section discusses this issue more and how it was resolved.

B. Out of Sample Testing

The testing period started immediately after the training period and consisted of approximately 66 days of Bitcoin from the GDAX exchange. The data was manipulated to match the format of the training data, giving a time series of states in the form of S. The behavior policy used in testing took the optimal action for a given state, which corresponded

to the maximum reward stored in the Q-table given a state. The policy is shown as:

$$\pi : S \rightarrow A$$

such that

$$A = \arg\max Q(S)$$

For every day in the testing set, the agent started trading with a cumulative position of 0 and took the optimal action according to the behavior policy for every state it visited through the day. At the end of the day, the time series of the agent's performance that day measured in PnL was stored in memory and reset for the following day's testing. Before moving on, the agent retrained on the day of data by performing a fixed number of training iterations under the learning policy. Each day of testing effectively incremented the size of the IS by a day to ensure temporal continuity between IS and OOS periods.

If during testing the agent found itself in an insufficiently explored state, the optimal action was estimated to be that of the 'closest' sufficiently explored state. Closeness in this sense was determined using a three dimensional nearest neighbors search in the space of explored states. Explored states are considered with the same criterion as was discussed in Subsection D of Section IV: we must create the set of all unique sets of price differences, RSIs and Stochastic Oscillator values visited in the training set. This will allow us to reference any point in the testing set that isn't directly recognized as a visited set to a familiar point in the training set. If s_1 is the state definition of an unrecognized observation, s_2 is that of recognized observation from the training set:

$$d(s_1, s_2)$$

which can also be shown as:

$$\|s_1 - s_2\| =$$

$$\|diff_1 - diff_2\| + \|RSI_1 - RSI_2\| + \|stoch_1 - stoch_2\|$$

Evaluating this at every recognized point from the training set and finding the point that minimizes the distance will result in a point from the training set that will approximate the unrecognized state. This 3 dimensional clustering is done for all unrecognized states, allowing the learner to take the approximately optimal action even if the state presented in the time series OOS is not directly inside the Q-table.

The daily performance trends were averaged across all days of testing, giving an average daily return trend shown below: Another point of interest was to study the agent's performance based on the daily trends in the price of Bitcoin. For this, each day of testing was classified as being one of three different types of markets: (1) upwards trending, (2) downwards trending, or (3) sideways. Days with upwards trending markets saw a generally increasing time series of price while downwards trending markets saw a decreasing time series. Days with a sideways market saw neither a significant increase or decrease in price. The classification of each day's market type was done by analyzing the trends

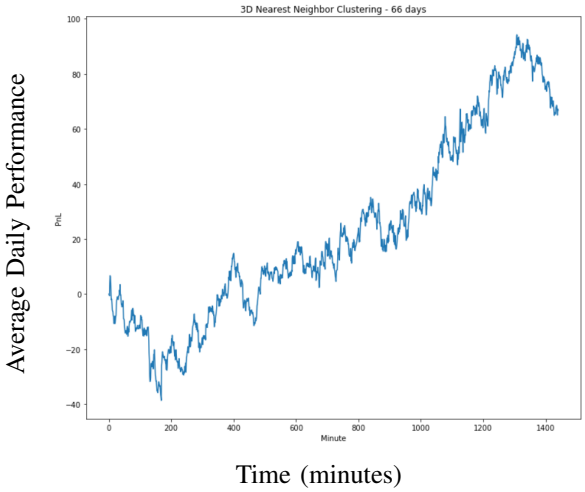

Fig. 13. Average Daily Performance OOS

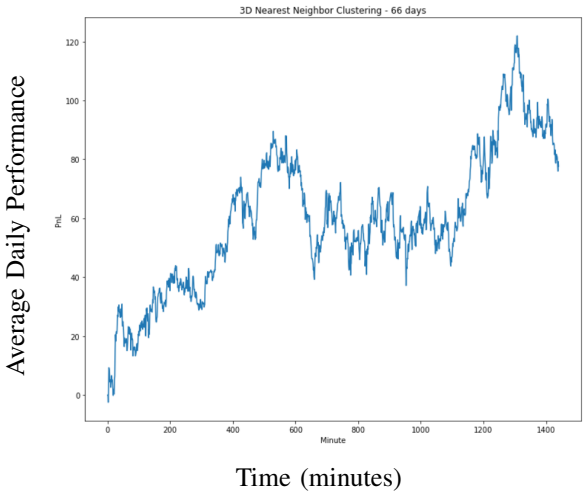

Fig. 14. Average Upwards Market Performance

of the peaks and troughs in the time series of price. If the trends of peaks and troughs on a given day were of the same direction (either both upwards or both downwards), then the market type for that day was classified as such. If the directions of the trends did not agree or had magnitudes of slope below a threshold amount, the market type for that day was classified as sideways. The daily performance trends were grouped by market type and averaged accordingly, giving average daily returns trends for each of the three market types.

These figures demonstrate the effects of market type on performance, particularly when the market is not upwards trending. In sideways and downwards trending markets, it can be seen that the agent takes losses for the first few hours of trading before it starts to profit. This *burn-in period* occurs in sideways and downwards market as a result of the market conditions during the first seconds of the training set. Since the agent's cumulative position is fixed to zero only once at the beginning of the training set, the agent has learned to take actions optimized for markets similar to the beginning

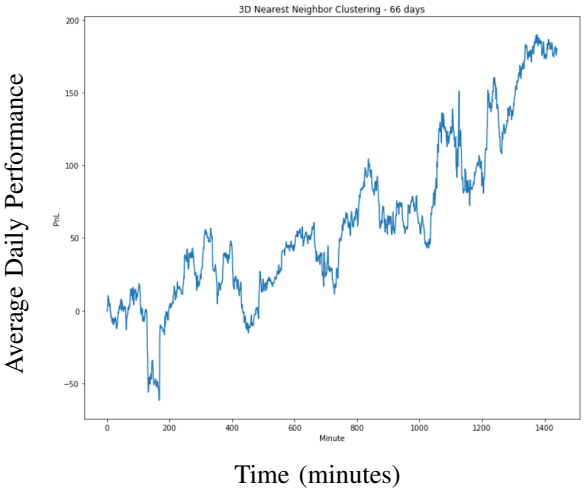

Fig. 15. Average Sideways Market Performance

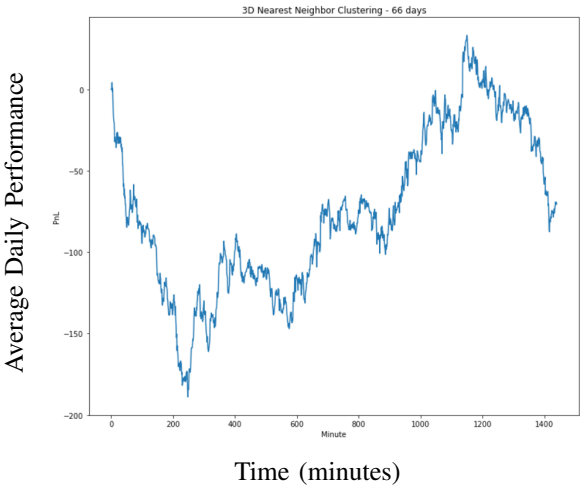

Fig. 16. Average Downwards Market Performance

of the training set when it finds its the cumulative position to be zero. In testing when cumulative position is reset to zero at the start of the day, the agent expects an upwards trending market and buys, causing it to lose when the price is not necessarily increasing as it would in an upwards market.

VI. Conclusion

This paper demonstrates an application of reinforcement learning methods in an algorithmic trader based on historical data. Drawing from studies conducted by Balch and Ritter, the trader made considerable daily returns after being trained using Q-learning.

Over the course of this study, the drawbacks to using Q-learning became apparent. Like other learning methods that rely on the use of a discrete state space, Q-learning is subject to the Bellman's curse of dimensionality. A more ideal learning method for this application would be able to handle many more than four variables defining state. Additionally, the mapping of state to optimal action would be a continuous function as opposed to a table in memory, solving the issue of clustering discrete states. In future replications of this study, Q-learning would preferably be replaced with learning methods such as Recurrent Reinforcement Learning or Recurrent Neural Networks which can handle larger state spaces.

Another aspect of this study that would varied in future replications was the definition of reward. Where this study used differential wealth measured in PnL as the agent's observed award in training, other works have found success using differential Sharpe Ratio [5]. This metric provides the agent with a better indication of its risk during the trading day which may in testing improve performance.

VII. Acknowledgements

References

[1] R. Sutton and A. G. Barto, *Reinforcement Learning*. Cambridge, Massachusetts: The MIT Press, 2012.
[2] G. Ritter, Machine Learning for Trading (August 8, 2017).
[3] T. Balch, Deep Q-Learning (April, 2017)
[4] F. Sharpe, Capital Asset Prices: a Theory of Market Equilibrium Under Conditions of Risk (September 1964).
[5] Du, Xin, Jinjian Zhai, and Koupin Lv. Algorithm Trading Using Q-Learning and Recurrent Reinforcement Learning. CS229, n.d. Web. 15 Dec. 2016.