Project 8: Reinforcement Learner

Donald Ward – 7/18/2019

dward45@gatech.edu

Q-Learner Problem Framing:

In this project I adapted the Q-Learner that was created for the maze navigation problem to create and execute a stock trading strategy. I used the same technical indicators I used for manual trading and let the learner make decisions on what to do with those indicators. The learner was able to solve such a seemingly different type of problem because the learner made the decision to move from state-to-state based solely upon reward feedback, unaware of what problem it was actually solving.

I first had to train the learner with in-sample stock prices before testing against both in-sample and out-sample data. The pattern for establishing reinforcement convergence and for training the Q-Learner was similar between our robot navigation and stock trading problem. In both cases we cycled through our stock training data until we converged on a solution by repeatedly providing state and reward in each interaction. It is in this mapping between the stock data and our interactions with the learner where the differences from the maze navigation problem start to emerge, and by examining each input and output to our learner we see how to adapt the Q-Learner for the stock trading problem.

State:

The state provided to the learner is derived from the technical indicators for a given day in the training period. These technical indicators must be first discretized in order to convert them from continuous to discrete numbers in the Q-Learner table.

For each indicator I used the pandas qcut method to partition each indicator's dataframe into indices that evenly place data elements into the number of requested bins that I provided. I stored this bin indices array along with the indicator dataframe for later referencing. Whenever then a subsequent request was made to retrieve the state for a given day, the technical indicator values would also be retrieved for that day. Then by using the Numpy digitize function and the aforementioned bin indices array, I was able to convert the indicator values into their discretized values and combine them into a single state value. I created this combining by taking each discretized value, treating them as their own unique place value and converting them into a numeric base using the number of bins provided.

I used the indicators for Momentum, Bollinger Band Percentage (BBP), MACD and MACD signal on my manual strategy project. I left Momentum and BBP data alone and subject to the whims of the Q-Learner because the decisions I made in the manual strategy were based upon explicit threshold values. However, in manual trading I was checking to see if MACD was greater or less than the MACD signal and thus retrieving a Boolean result based solely on comparison. This True/False value did not map over readily to the learner and would not discretize readily. I decided to let the learner have the raw data for these indicators instead of trying to somehow pre-process the data and let the learner just make its own determination on how to interpret their values. In this way I could use the same indicators but also avoid the awkward True/False values that would not readily discretize.

Action:

There are three stock trading actions used in this trading problem, setting positions for Long, Short and Cash. The product of these 3 actions and the number of states provide us the size of our Q-table. Yet, the Q-Learner alone provides no understanding of what its actions mean in the context of the trading problem – only that it associates an action with a reward. Here then lies a crucial step: When adding evidence, you must arbitrarily decide what an action returning from the Q-Learner means in the context of the trading problem by mapping to our trading

actions. In my case, I decided that positions Short, Cash and Long would be represented by learner actions 0, 1 and 2 respectively. The learner would not know about this stock trading action because the learner is only capable of interpreting the consequence of providing that action by receiving a reward, when previously given a discretized state of technical indicators representing a given day. If we received an action from the learner that we interpreted as a short position, we would negate the days return in the reward. This would ensure that price declines would incur positive rewards, and price gains would incur negative rewards. The discretized state location in the Q-table would be modified with that reward, for that action and thereby improve the chances of choosing that action given similar circumstances in the future because we would be returning the action with the maximal award when not under the influence of a random action chance.

When we start to simulate actual purchase actions based on these reinforcements in our test policy code we must map actions again. The manual strategy used signals for Short, Cash, Long with the values -1, 0, 1 respectively. In our strategy learner code, we retained these purchase signals and mapped the values coming from the learner to these signal values. These signal values let us know when to buy and sell and by idempotently calling the learner querysetstate() method we can acquire predictive actions for all of our sample dates and use them to construct our trades and ultimately our portfolio used in our graphs.

Reward:

The reward given to the Q-Learner is based on the percentage return from the daily price of a stock compared to the previous trading day. This reward is passed to the learner along with the state (discretized technical indicators) for that day and as mentioned before is used as the basis for deciding if an action should be chosen because the learner will try to use the maximal award. If impact was provided, the reward is reduced by the amount of the impact for that day's price.

Convergence:

In my case, I decided to converge if a hard time limit was reach (22 seconds), the number of epochs were reached (200) or a daily return loss trend was detected for 10 days straight. In all cases, the maximum number of epochs was reached.

Experiment 1

In this experiment I compared the learner strategy with the manual strategy for training period 1/1/2008 to 12/31/2009 and tested against in-sample data. I used the same indicators BBP, MACD, MACD Signal and Momentum for both strategies. The table below lists several parameters I passed to my Strategy Learner and to the Q-Learner:

Parameters Used	**Value**
Symbol	JPM
numpy.random.seed	1
Training Date Range	1/1/2008 to 12/31/2009
Starting Value	100000
Commission	0.0
Impact	0.0
Bins	9
Alpha	0.5
Gamma	0.7

Rar	0.5
Radr	0.99
Indicator Rolling Days	20
Indicator Momentum Days	14
Epochs	150
Random Epochs	60
Dyna	Disabled

Table 1. Parameters used to construct Strategy Learner and Q-Table

The experiment showed far superior in-sample scores for the Q-Learner compared to the manual strategy and the benchmark. These relative results could be expected to hold after repeated runs because the number of epochs run was high enough to reinforce the decisions made based on the actual reward values. So of course, presenting these identical states would often retrieve the optimal action for that date.

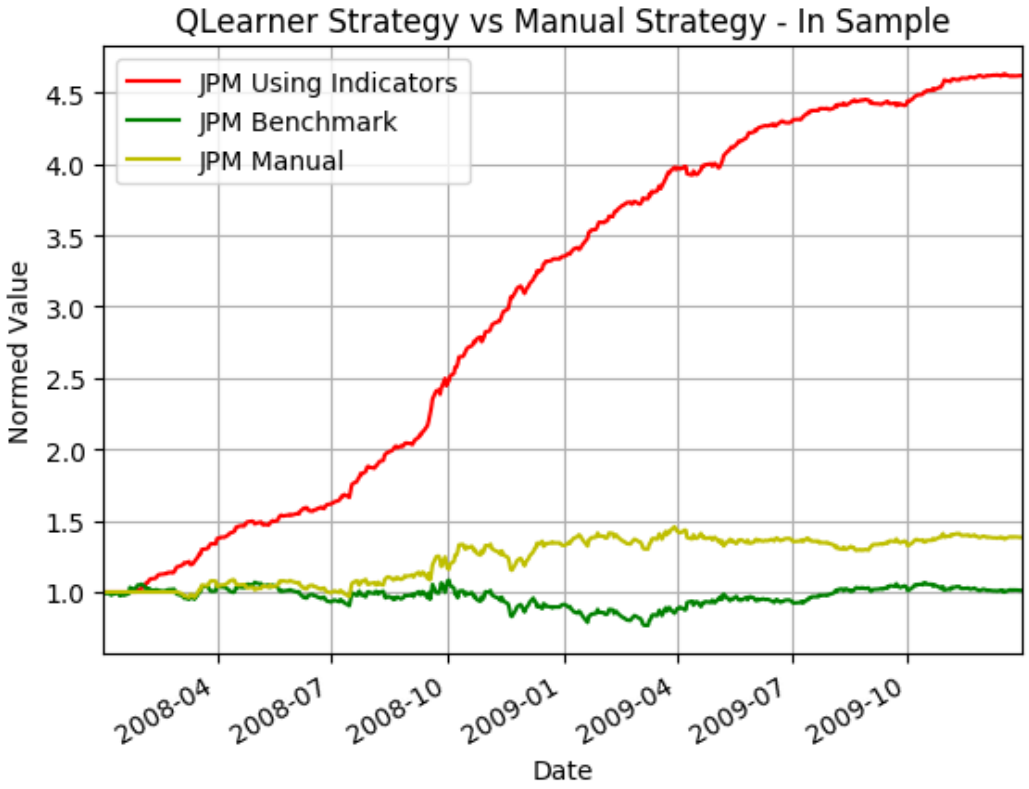

Figure 1. Graph showing learner strategy vs manual strategy. Q-Learner provides superior results.

	Q-Learner Strategy	**Benchmark**	**Manual Strategy**
Sharpe Ratio	8.02312245469	0.156918406424	0.897343586176

Cumulative Return of Fund	3.6187	0.0123249333401	0.384
Standard Deviation of Fund	0.00605191703901	0.0170412470682	0.0128650772222
Average Daily Return of Fund	0.00305869460007	0.000168759162	0.000727228499188
Final Portfolio Value	461870.0	101027.7	138400.0

Table 2. Experiment 1 portfolio statistics. Q-Learner obliterates the competition yet again.

Experiment 2

Experiment 2 used the same initialization and configuration for the Strategy Learner that was used in experiment 1.

Hypothesis:

Increasing impact will cut into profits and I believe will negatively affect the frequency of trading actions. As impact increases, I would expect to see a declining price compared to the benchmark as well as less frequent trades. I will create a chart that overlays in-sample JPM results for impact values using 0.000, 0.001 and 0.005. I will also show another chart with the number of trades illustrated using vertical mark for these same impact numbers.

Results:

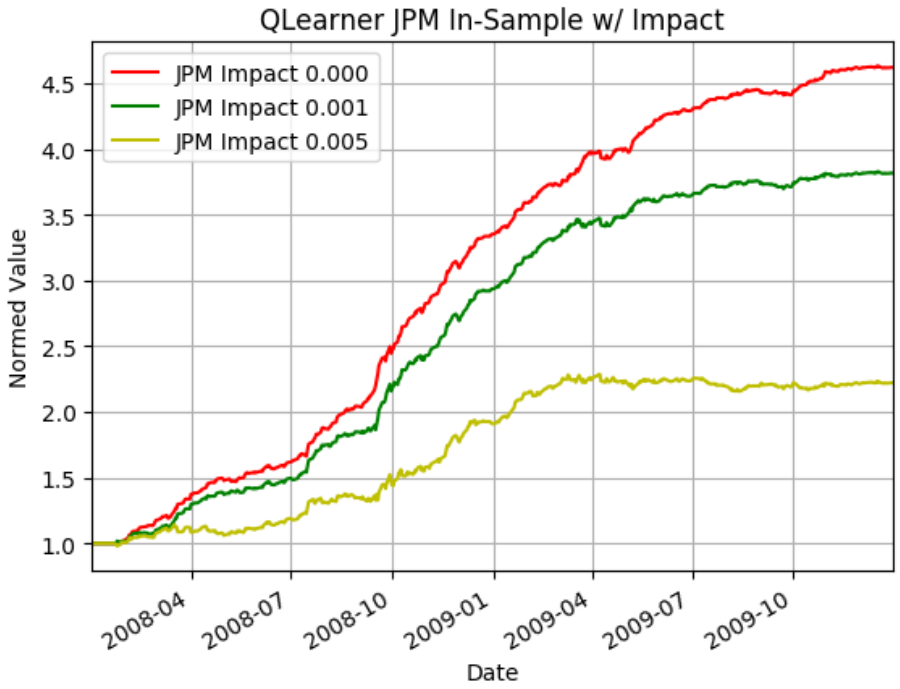

Figure 2. Increasing impact reduces profits.

The effect of the impact is visible in Figure 2. We can see a reduction in the portfolio values as impact is increased. The portfolio goes from a normed value of around ~4.6 to ~3.8 with impact of 0.001 and to an abysmal ~2.3 when impact reaches 0.005. Impact affects the bottom-line and For the next example I plotted the same values used in Figure 2. into separate sub-plots showing trading activity.

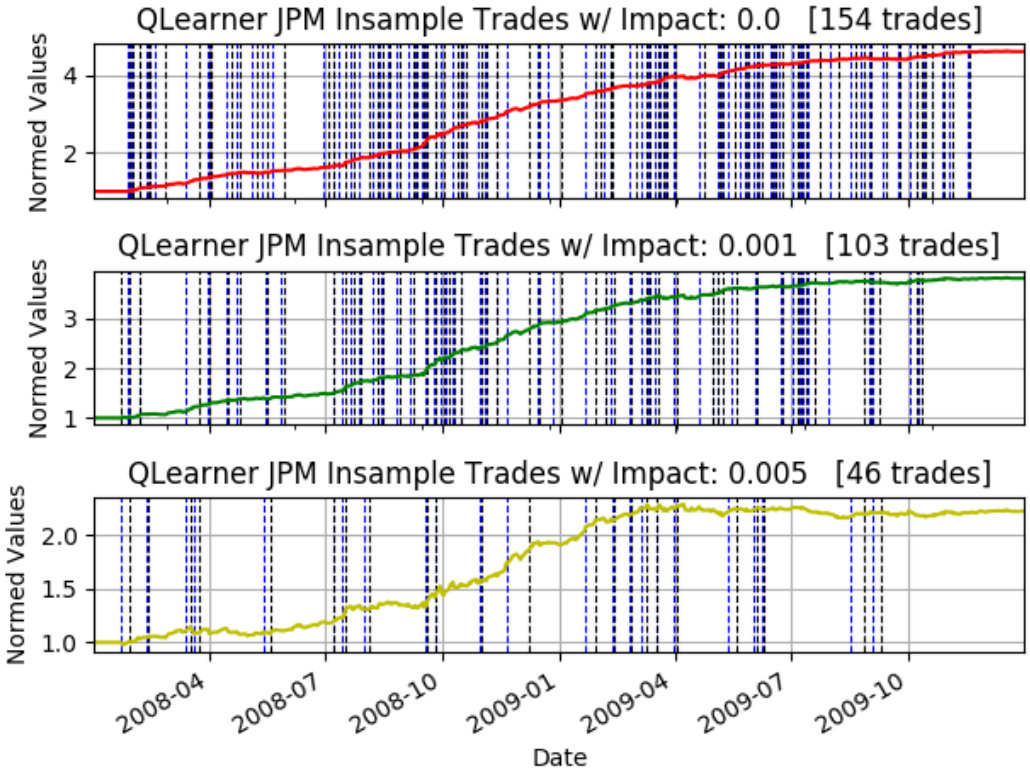

Figure 3. The effect of impact on trade frequency. Vertical lines represent buy and sell events (cash position not shown). Trades decrease as impact increases.

The effects of impact on trading frequency are clear to see in Figure 3. Without any impact in we can see there are 154 buy/sell trades that rapidly go down to 103 for 0.001 impact and then to 46 trades for 0.005 impact. This would make sense because frequent trading would constantly be penalized in the form or reward decrease. The buy/sell decision would only happen when the momentum was more extreme, triggering a change in position.