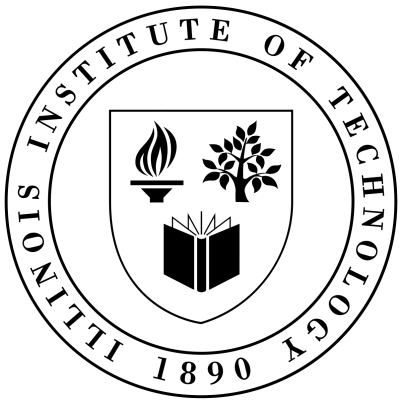

MATH527

Market Making Project

Authors:
Brandon Bennitt
Kemen Goïcoechea
Catherine Thomas
Akhil Srinath

Supervisor:
Matthew Dixon

Fall 2021

Contents

1 Introduction

In the financial markets, high frequency market making is a set of trading strategies that involve placing a limit order to offer or bid on a security to earn the bid-ask spread. For the sake of this project, we define the problem as when to optimally quote either a bid or ask, or simply wait, each time there is a limit order book update. In certain situations, a market maker may wish to quote an ask to close out a long position if it will most likely give an instant reward. On the other hand, it is possible that the price may continue to increase and it would be optimal to wait to capture a larger profit.

With technology becoming an ever increasing factor in how market makers operate, one may want to automate the process of optimally buying and selling with a machine learning algorithm or agent. This project seeks to use reinforcement learning agents to learn how to become a high frequency market maker. By defining a state space, action space, and rewards for each action, a reinforcement learning agent can be trained to learn an optimal policy such that it takes actions which are expected to maximize its reward function, and hence capture the bid-ask spread to make a profit.

2 Related Work

As this is not an uncommon problem that one may want to solve, there are different resources which have solved a variation of the problem we wish to solve. For example, the MATH 527 github repository contains an example of how to use reinforcement learning to learn an optimal policy to become a market maker. The authors were able to successfully implement the Q-learning and SARSA algorithms to trade as a market maker. In this notebook, the authors decided to simplify the problem by using a 2-dimensional discrete state space, with one component being the current position and the second component being the quantized fill probability. The authors were able to implement the Q-learning and SARSA algorithms themselves in the notebook to train the agents and generate an optimal policy that could be visualized as a table. The results from this work were then quantified to show how well the agents performed in this setting, with both agents being able to capture a significant profit in a short period of time.

3 Goal

In this project, we seek to extend the scope of the project that is in the MATH 527 github repository. Analyzing the work that was already completed and seeing how it can be extended to solve a slightly more realistic problem, this project decided to implement a continuous state space that did not discretize the fill probabilities. This

greatly increases the complexity of the problem as we can no longer use Q-learning or SARSA in their original form. To use either one of these approaches, some mechanism of approximation is required since we can not have an infinite number of elements in a lookup table. To solve this problem, we decided to implement two deep reinforcement learning algorithms: Deep Q-learning and Proximal Policy Optimization 2 (PPO2).

4 Mathematical Problem Formulation and Constraints

In this project, we defined the problem with constraints to simplify the real life problem of a market maker to show how reinforcement learning may be used to capture the bid-ask spread. The agent uses the liquidity imbalance at the top of the limit order book as a predictor for price movement, and therefore, fill probabilities. Another constraint put in place was the inventory had to stay between -1 and 1. This means that the agent may only be long one share, short one share, or flat at any given time.

In this project, we get a stream of data to simulate data coming in from a market exchange about one particular stock. At each non-uniform time update, t, the market feed provides best prices and depths $\{p_t^a, q_t^a, p_t^b, q_t^b\}$, where p is price, q is depth, a represents ask, and b represents bid. The state space is a 2-dimensional state space that is the product of the inventory, $X_t \in \{-1, 0, 1\}$, and liquidity ratio $\left\lfloor \frac{q_t^a}{q_t^a + q_t^b} \right\rfloor \in [0, 1]$, where q_t^a and q_t^b are the depths of the best ask and bid. As $\hat{R}_t \longrightarrow 0$, the mid-price will go up and an ask is filled. As $\hat{R}_t \longrightarrow 1$, the mid-price will go down and a bid is filled.

A bid is filled with probability $\epsilon_t := \hat{R}_t$ and an ask is filled with probability $1 - \epsilon_t$. The rewards are chosen to be the expected total P&L. If a bid is filled to close out a short holding, then the expected reward $r_t = -\epsilon_t(\Delta p_t + c_b)$, where Δp_t is the difference between the exit and entry price and c_b is the transaction cost of a bid. Note that this project gives an option to set different transaction costs for bids c_b and asks c_a.

As an example, if the agent entered a short position at time $s < t$ with a filled ask at $p_s^a = 100$ and closed out the position with a filled probability at $p_t^b = 99$, then $\Delta p_t = 1$. In addition, the agent is penalized 5 units for quoting an ask or bid when the position is already short or long respectively.

To increase the complexity of this problem to model the environment that a real life market maker would encounter, market orders, knowledge of queue positions, cancellations, and limit order placement at different levels of the ladder could be added, but are ignored in this project.

5 Method

In order to create Deep Q-Learning and PPO2 agents, we decided to use the most popular open source reinforcement learning library to date, OpenAI Gym, to aid in reducing the amount of new code to be written. The purpose of creating an environment that is Gym compliant is there are many different open source libraries that define agents and RL algorithms that use the Gym environments. By creating a Gym environment for this version of market making, we are able to use multiple agents from different libraries to work in a plug and play fashion. After a market making environment was created using Gym, the project made use of the stable baselines open source library to implement Deep Q-Learning and PPO2.

5.1 OpenAI Gym Environment

The first part of this project that required programming was creating a market making OpenAI Gym environment. To begin, we referenced the market making notebook from the MATH 527 github repository and analyzed it to understand its inner workings. Our first goal was to implement a discrete state space environment to train Q-learning and SARSA agents to get comparable results to the previous work, which implemented the same algorithms without a Gym environment.

While reading the OpenAI Gym documentation, we found that there are certain requirements that each environment must fulfill in order to be Gym compliant. In summary, a Gym environment is a Python class with at least three required functions, $init()$, $step()$ and $reset()$, and two required instance variables, $action\_space$ and $observation\_space$. The $step()$ function, which is called at every iteration of the training, returns $\llbracket next\_state, reward, done, info \rrbracket$, which corresponds respectively to the state after the action is taken, the reward produced, a termination boolean, and a dictionary that can be added to include debugging information. The $init()$ function initializes the basic variables of the environment like the $action\_space$ and the $observation\_space$. In the $init()$ function, we also initialize variables related to a market making environment such as a buy_cost and a $sell\_cost$. The $reset()$ function returns the starting state the environment initializes to. The $action\_space$ and $observation\_space$ are required to be initialized to Gym spaces, which are either 'Discrete' or 'Box'.

After we had defined all of these functions and variables, we ran an automated check, provided by the stable baselines library, to ensure that our environment was Gym compliant and included all of the necessary functions and variables to interact with agents. After we defined a class that was Gym compliant, we knew we still

needed to implement some utility functions that would help us create a fully functional market making environment. A *DataFeed* object was defined that passes observations from the dataset to the *step*() function. The *DataFeed* object acts as an exchange that communicates changes in the limit order book to the environment. For the discrete state space, the *choose_action*() function, as its name implies, chooses an action based on the current state of the agent as well as the value for exploration and exploitation and the time it is called. If ϵ is the probability for exploration, a random action is chosen with probability and with probability $1 - \epsilon$ the agent chooses the action which maximizes its expected rewards. Lastly, empty arrays are added as instance variables and are updated at each step to keep track of every reward during the training. These are used later in the notebook for visualization purposes.

While we had no issue implementing our discrete state space environment, a problem we encountered while implementing the continuous state space environment was the lack of ability to restrict the actions for a given state. Thus, in every inventory state the agent can hold, buy, or sell even if it is long or short. To resolve this issue, we adjusted our rewards and gave a strong negative reward for invalid actions (such as sell in a short state or buy in a long state).

The *DataFeed* object was also adapted from the MATH527's notebook. In the original notebook, this function raised an error to stop the learning when the agent arrived at the end of the dataset. In our case, we want to train over multiple passes of the dataset. Thus, the function was changed to return the first observation when it arrived at the end of the dataset to create a loop of observations to train over.

5.2 Stable Baselines Deep Q-Learning and PPO2

The main goal of this project is to use deep reinforcement learning agents on continuous state space market making environments. There are some libraries that have implemented agents compatible with OpenAI Gym environments. The first one that we tried to use was *ElegantRL*, which is specialized for finance applications. However, the code was not very modular and using their agents in our environment was not straightforward. Thus, we looked at the *stable_baselines* library which fit our needs perfectly.

We focused on two types of agents: on-policy and off-policy algorithms. The most known off-policy learning algorithm is Deep Q-learning. Like Q-learning, it is a greedy algorithm but it approximates the Q-values with a deep neural network.

The other agent is implemented using the Proximal Policy Optimization 2 algorithm. This is a policy gradient algorithm, which means that it uses a stochastic

gradient descent to find the optimal policy.

5.3 Viewing Optimal Policy and Convergence

One objective that was fixed at the beginning of the project was to train agents and visualize their learned policies for discrete and continuous environments. For discrete environments, the procedure was straightforward as it corresponds to a table and was implemented in the original notebook. On the other hand, the thought process for visualizing a policy for a continuous environment was very challenging. Our idea was that it would correspond to a stepwise curve for each state, but its implementation was not successful. Therefore, we focused on the visualization of the learning of the agent. To do so, the rewards were saved in an array during the training and were plotted afterwards. Another method to display this information is through a module named TensorBoard. TensorBoard can be used if the deep reinforcement learning agent is created with TensorFlow. This module is a graphical interface that displays the rewards recovered from the learning environment and the evaluation metrics of training the deep neural network.

6 Results and Discussion

6.1 The Dataset

To test our implementations, we used the dataset AMZN-L1.csv given in the MATH 527 github repository. This dataset is composed of 57,515 continuous observations of the Amazon stock. Each observation contains the following information: $\imath ask, ask\_depth, bid, bid\_depth\imath$. Based on past and current observations, we wanted our agents to place bids, asks, or hold to maximize the return of trades.

6.2 Discrete Environment

For the first part of this project, we tried to convert the MATH 527 notebook to an OpenAI Gym environment. To verify that our approach was successful, we trained the same Q-learning and SARSA agents with the new environment and compared the learned policy with the original notebook's policy:

```
SARSA
ask fill prob: 0.00 0.11 0.22 0.33 0.44 0.56 0.67 0.78 0.89 1.00
         flat     b    b    b    s    s    s    s    s    s    s
        short     b    b    b    b    b    b    b    b    b    h
         long     h    s    s    s    s    s    s    s    s    s
Q-learning
ask fill prob: 0.00 0.11 0.22 0.33 0.44 0.56 0.67 0.78 0.89 1.00
         flat     b    b    b    b    b    s    s    s    s    s
        short     b    b    b    b    b    b    b    b    b    b
         long     h    s    s    s    s    s    s    s    s    s
```

Figure 1: The policy obtained in the original notebook

```
SARSA
ask fill prob: 0.00 0.11 0.22 0.33 0.44 0.56 0.67 0.78 0.89 1.00
         flat     b    b    b    b    s    s    s    s    s    s
        short     b    b    b    b    b    b    b    b    b    h
         long     h    s    s    s    s    s    s    s    s    s
Q-learning
ask fill prob: 0.00 0.11 0.22 0.33 0.44 0.56 0.67 0.78 0.89 1.00
         flat     b    b    b    b    b    b    s    s    s    s
        short     b    b    b    b    b    b    b    b    b    h
         long     h    s    s    s    s    s    s    s    s    s
```

Figure 2: The policy obtained with our OpenAI gym environment

We can see that the two policies are very similar. Thus, we concluded that we successfully ported the code from the previous notebook into a fully functional OpenAI environment.

6.3 Continuous Environment

For the continuous environment, we trained two agents from the *stable_baselines* library. Both agents were trained and evaluated through 5 episodes of the dataset (a training of this length is about 30 minutes).

6.4 The Deep Q-learning Agent

The first agent that we trained was a Deep Q-learning agent that works as an off-policy algorithm. When instantiating a Deep Q-learning agent, we found an interesting parameter that was crucial to modify. The $exploration\_fraction$ parameter is described as the "fraction of the entire training period over which the exploration rate is annealed". By default, this parameter is set to 0.1, which means that the epsilon decay occurs over only 10% of the training. The rewards obtained with this default setting are shown below:

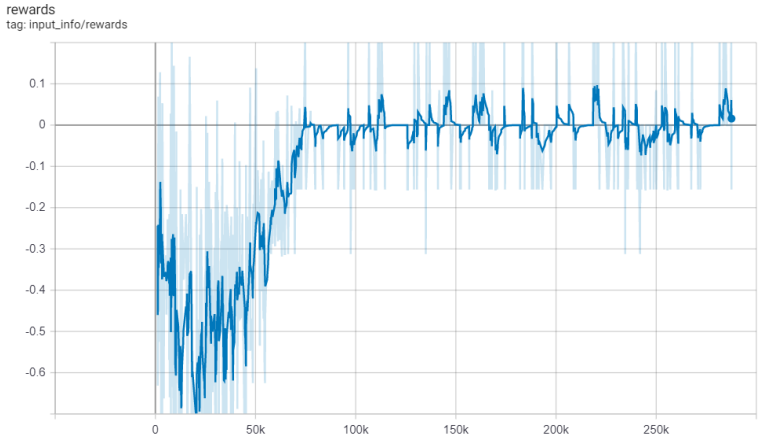

Figure 3: The rewards during a training of a DQN Agent with $epsilon\_fraction = 0.1$ (the default value)

Seeing that the rewards stop consistently increasing after about 75,000 steps, we realized the agent needed more time to explore the state-action space and update the weights in the network. Therefore, we changed this hyperparameter to 0.5 to explore for more steps through the dataset before converging to a learned policy and obtained this type of result:

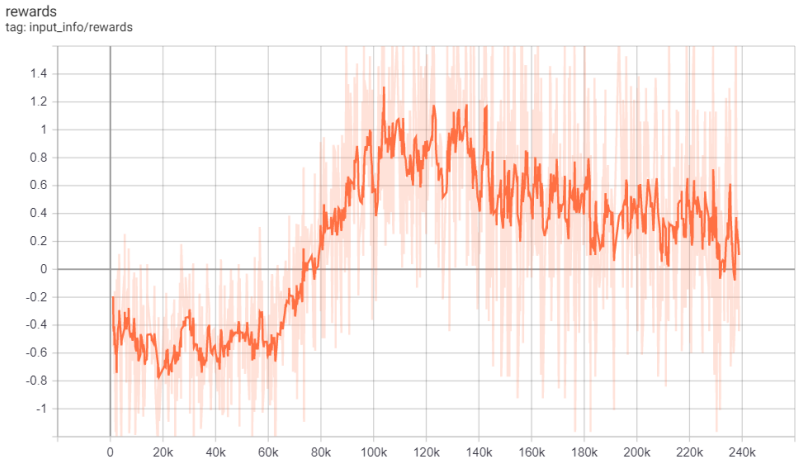

Figure 4: The rewards during a training of a DQN Agent with $epsilon\_fraction = 0.5$

By comparing these two plots, we can see that we obtain many more positive rewards when the $exploration\_fraction$ parameter is set to 0.5. To show our cumulative rewards obtained through the last pass of the dataset, we show the cumulative rewards as a function of steps below.

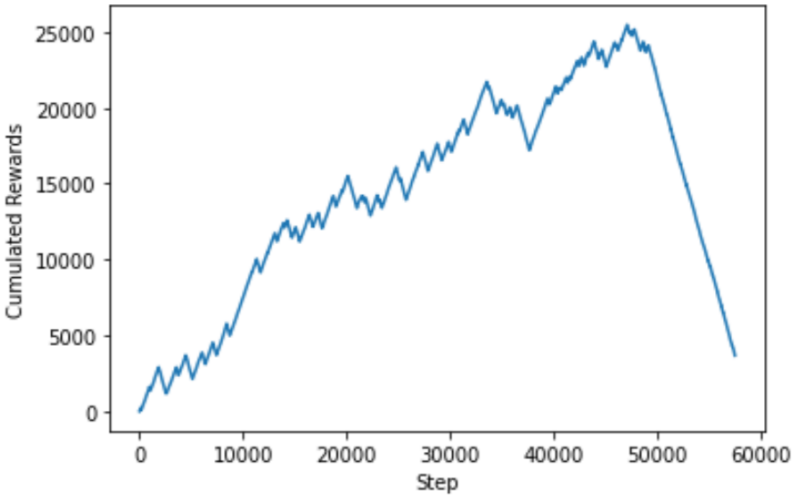

Figure 5: The cumulative rewards during the last pass on the dataset for the DQN agent

As seen in the plot, we have a very stable increase in rewards over time, showing that our agent is profitable. Through 50,000 steps, we have a cumulative P&L of approximately $25,000. After 50,000 steps, we see a sharp, linear decrease that results in the agent losing almost all of its rewards that it made up to that point. This is extremely strange behavior as it does not make much sense for the agent to linearly lose the rewards that it has gained. Since we know we have not rolled over the data yet and we don't see an instantaneous decrease in rewards, it seems that the agent is continuously choosing actions with the same negative reward. After discussion amongst team members, we realized that the only explanation for this would be that the agent continuously decides to buy when it is already long or sell when it is already short for the entirety of the linear decrease. The only way the agent would continue to take the negative penalty is if it calculated that performing a different action would result in a more negative reward. As the agent nears the end of the dataset, it is possible it is expecting a large price drop and has somehow determined it is more optimal to take a smaller penalty continuously rather than to get hit with a one time massive negative penalty. More research and investigation is needed to determine if this is an accurate assumption.

6.5 The PPO2 Agent

The PPO2 agent is a policy gradient agent. The results are obtained with a MLP Policy, which is the default policy in the *stable_baselines* library.

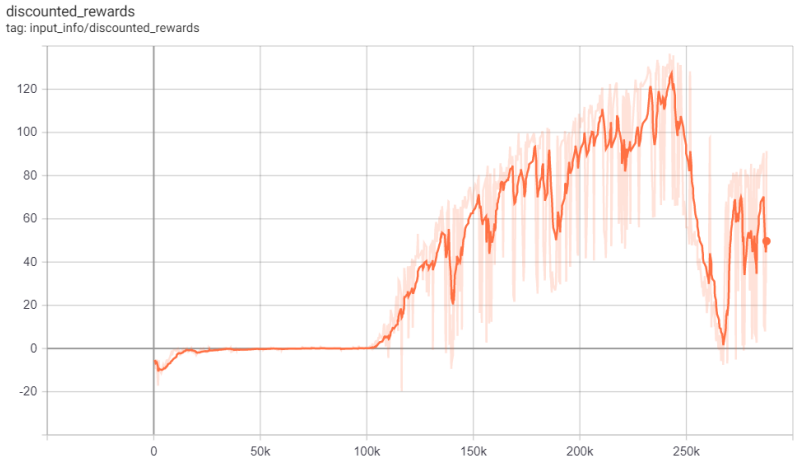

Figure 6: The rewards during a training of a PPO2 Agent

As seen in the plot, the results for the positive discounted rewards are very good starting from approximately the beginning of the third episode of learning (since the dataset length is about 55,000 observations, the second episode ends at 110,000 steps). We believe that the big decrease at around 250,000 steps is due to the loop over the dataset and the sudden, drastic change of the stock's value. Ideally, we would have a dataset that was long enough to successfully train the agents to the best of their ability without having to loop through it several times.

Below we show a plot of the cumulative P&L for the PPO2 agent through the last pass of the dataset.

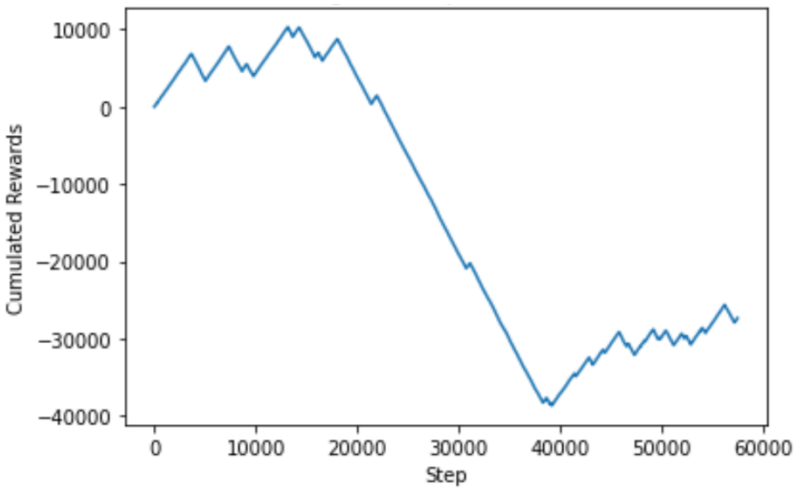

Figure 7: The cumulative rewards during the last pass on the dataset for the PPO2 agent

In this plot, we see very unstable results. The agent is able to get to an impressive P&L extremely quickly, but then has a series of linear decreases. This may be due to the same reasons as discussed for the linear decrease in the Deep Q-learning

agent. More investigation would have to be done to explore why this decrease is occuring, as it seems it is highly profitable outside of the long span of linear decreases.

7 Future Work

Even though we are proud of the work accomplished, we have just touched the tip of the iceberg of market making using reinforcement learning agents. We found many ways of continuing or improving this project.

Since we successfully implemented an OpenAI compliant market making environment for a continuous state space, visualizing the optimal policy for a deep reinforcement learning agent would be a future goal. There are different ways that this may be done, but we believe one way to visualize the optimal policy would be to plot a 3-dimensional space with two components being the state space and the third component being the optimal action given that state.

Another way we can extend this project to get more realistic results is to increase our dataset size. We used a dataset that contains about 57,000 records. If we had more data we would get better results as we would not have the limitation of working with data that had to be rolled over multiple times to get fully trained. This may solve some of the problems that we have seen in the cumulative rewards plots for each of the agents.

In addition, we did not include this part in the notebook, but we tried to split the dataset into a training and testing set to train the model using the training data and evaluate its performance by using the testing data. However, the results were pretty disappointing since in testing conditions the agents preferred to hold instead of placing market orders. We believe this has to do with the rewards given for invalid actions. The rewards may be too negative and discourage the agent from doing anything but holding. Thus, one could study in-depth optimal rewards to give to a market making agent, or implement a way of restricting actions in certain states.

Moreover, the problem can be enhanced with the use of market orders along with the knowledge of queue positions, cancellations and limit order placement at different levels of the ladder. To help in this task, one can use other open source frameworks such as FinRL and ElegantRL. These libraries would be useful as they implement deep reinforcement learning agents specifically for finance.

Finally, to be able to supervise the agent's work on a testing set, an animation of consecutive actions taken plotted over the stock price would be a great addition. This would allow us to see when and where the agent decides to take certain actions.

8 Conclusion

To summarize, through this project we were able to extend the work of the MATH 527 market making notebook by handling a continuous state space. This was a challenging but rewarding step because we were able to create an OpenAI Gym environment. Since the OpenAI Gym library is the most popular library in the reinforcement learning area, there are many other open source libraries that implement different algorithms that use Gym environments. Creating the environment took time, but once it was completed, we were rewarded with a variety of other tools we could use to help us solve the problem.

The open source library that we used to implement Deep Q-learning and PPO2 is the *stable_baselines* library. By trial and error, we were able to find the correct parameters that led to optimal performance for the data that we trained the agents on. We trained the agents on an Amazon stock dataset and evaluated their performance by plotting the rewards for each time step as well as cumulative rewards.

Overall, we were able to successfully train multiple reinforcement learning agents to act as high frequency market makers. As mentioned before, if future developments are made to this project to mimic a problem that a real world market maker faces, it is possible that this approach would prove to be extremely profitable, and therefore, successful.

References

[1] Matthew Dixon and Igor Halperin, *ML_in_Finance_MarketMaking.ipynb*

[2] Stable Baselines library, *https://stable-baselines.readthedocs.io/en/master/*

[3] Mate Pocs, *Beginner's Guide to Custom Environments in OpenAI's Gym*

[4] Amrani Amine, *Deep Q-Networks: from theory to implementation*