Optimal Market Making with Reinforcement Learning

Jaupi, Megi
jaupimeg@mit.edu

Laskowski, Michal
michlask@mit.edu

Tan, Shiyin
shiyin11@mit.edu

1 Introduction

During an era characterized by the prevalence of high frequency trading in financial markets, market making strategies play a crucial role. Market makers place multiple bid and ask offers to provide enough liquidity in the markets. The difference between the bid and ask price, namely the spread, represents the profit of this agent for the provided service. A market maker's goals are to maximize profits from capturing the spread and minimize the risk of holding a substantial inventory that could result in significant losses if the market experiences unfavorable fluctuations.

Our goal in this project is to use sensorimotor learning algorithms to generate optimal trading strategies for market making. This project can be broadly categorized as an application of sensorimotor learning in the realm of optimal trading, which falls under option B of the research scope.

To develop an algorithm that learns optimal strategies for market making, it is crucial to have a comprehensive model of the order book and its evolution. A common approach involves modeling the mid-price of the asset. In line with classical quantitative finance theory, we will employ a Brownian Motion process to model the mid-price. Additionally, we will model the Market Orders as Poisson Stochastic Processes and specify their probabilities of being filled. This market model will have only one agent interacting with it, namely the market maker. This implementation would follow the market model as outlined in this paper [4].

We treat this problem as a *finite time, sequential decision making framework*. We first implement a random strategy as a benchmark and then multi armed bandit algorithms as a first approach towards improving upon the benchmark disregarding the fact that the reward depends on the state and actions affect state transitions. We then tackle the problem with two different approaches that are suited for sequential decision making problems: we implement a policy gradient algorithm (PPO) and a reinforcement learning one (Q-Learning) and run several experiments to compare their performance.

2 Related Work

Market making has been extensively studied in the financial literature, commonly treated as a *stochastic optimal control problem*. One of the classic papers of the topic is [4] where the authors derive the optimal bid ad ask quotes in a two step procedure, computing a personal indifference valuation for the stock based on the agent's current inventory and then calibrating the bid and ask quotes to the limit order book by considering the probability of execution as a function of their distance from the mid-price. Similar papers that model price and order dynamics and develop optimal control algorithms are [2] and [3].

In recent years there has been increasing work on using newer algorithms in market making problems. [1] develops a market making agent via online learning whereas [5] use a Q-learning and R-learning agent to learn market making strategies by reconstructing an order book based on historical data.

6.484 Computational Sensorimotor Learning Project (Spring 2023).

3 Problem Formulation

3.1 Market Environment

In our market, there is only one asset, the price of which follows a Geometric Brownian Motion stochastic process, inherently assuming that the returns are log-normally distributed following a common framework in Quantitative Finance. The return dynamics are given by the following stochastic differential equation:

$$dS = \mu S dt + \sigma S dW \tag{1}$$

where S is the stock price, μ is the drift, σ the volatility and W is a standard Brownian Motion. The SDE has an closed form solution which provides the dynamics of the stock price:

$$S_t = S_0 e^{(\mu - \frac{\sigma^2}{2} t + \sigma W_t)} \tag{2}$$

Using this closed form expression, we simulate the stock market's mid price. Note that this is an indirect model of the market without considering the depth of the order book.

At any point in time, there is a number of market orders arriving following a Poisson Distribution for both incoming market buy orders and sell orders, respectively $n_{\text{buy}} \sim \text{Poi}(\lambda_{\text{buy}})$ and $n_{\text{sell}} \sim \text{Poi}(\lambda_{\text{sell}})$.

3.2 The Market Maker Agent

Our agent is a market maker, i.e. an agent that will place both bid (buy) and ask (sell) orders in the market. For simplicity purposes, we fix the volume of these orders as a constant. We also assume that at any point in time, the market maker will place orders in the market. With these assumptions, the market maker only has to make decisions regarding the price of the asset, or differently said, the spread. More formally, the market maker will decide how many ticks away from the mid-price will he/she quote.

The action space

Let $d \in \mathbb{N}$ represent the number of ticks away from the mid-price that the market maker can quote. Also, let d_p represent the tick size. Then the action space is defined as $[0, d-1]^2 \in \mathbb{N}^2$, where each action (i, j) means that we have quoted i ticks lower from the mid-price as bid, i.e. $\text{Bid}_t = S_t - i d_p$ and j ticks higher from the mid-price as ask, i.e. $\text{Ask}_t = S_t + j d_p$.

The state space

Let Q represent the maximum position allowed for the market maker to have. Then the state space for any point in time t is represented as $[-Q, Q]$ with increments of the fixed volume size of the orders. The inventory of a market maker can have both negative and positive values depending on the positional risk that the market maker is willing to take at each point in time. Positive values mean that the market maker is long the stock whereas negative values mean that the market maker is short the stock.

The transition function

In order to mimic the real market behaviours, once the agent places the orders, the probability of these orders being executed will depend on the spread. Certainly, the market maker has an incentive to provide a large spread in order to maximise its profits, but on the other hand, a larger spread will lower the probability of the orders being executed, $p_{\text{exec}} = e^{-\kappa(\text{Ask}-\text{Bid})}$, with κ being the sensitivity of the execution probability to the spread. Intuitively one can think of κ as a measure of market liquidity. Then, the number of orders executed follows $n_{\text{exec,buy}} = \text{Bin}(n_{\text{buy}}, p_{\text{exec}})$ for the bid offers and $n_{\text{exec,sell}} = \text{Bin}(n_{\text{sell}}, p_{\text{exec}})$ for the ask offers respectively.

For the purpose of the transition function and calculations, we can aggregate the number of buy and sell orders. Then it is clear that the transition function from one state to the next will depend on the number of orders being executed and the market price at the time of execution:

$$Q_t = \begin{cases} Q_{t-1} + \text{Ask}_t(n_{sell,t} - n_{buy,t}) & \text{if } \ n_{sell} > n_{buy} \\ Q_{t-1} + \text{Bid}_t(n_{sell,t} - n_{buy,t}) & \text{if } \ n_{sell} < n_{buy} \end{cases} \tag{3}$$

Note that for easiness of notation, in Equation 3 we did not consider the maximum allowed inventory Q. We do implement the logic of executing only the portion of the aggregated orders that is allowed per the maximum inventory in our experiments.

3.3 The Objective

The market maker is a rational agent striving to maximize its total expected profits. We define the value function as:

$$V_t = Q_t S_t + X_t \tag{4}$$

where Q_t represents the inventory, S_t the mid price, and X_t the cash process at time t. X_t is calculated as the difference in the dollar value between the executed ask orders and bid orders.

Then we can define a simple reward function as:

$$R_t = V_{t+1} - V_t \tag{5}$$

A more realistic reward function that we are working with is:

$$R_t = \gamma(V_{t+1} - V_t) - \xi Q_t^2 \tag{6}$$

where γ is a discounting factor for t and ξ is a penalty for holding inventory. After all, the market maker needs to be a market-neutral agent and avoid inventory risk. The objective of the market maker as such is to maximize the incremental rewards until market close at time T.

4 Algorithms

4.1 The Benchmark: Random Agent

We use a random strategy as the benchmark, which randomly selects bid price and ask price at time t:

$$\begin{cases} \text{Bid}_t = S_t - i_t * d_p \\ \text{Ask}_t = S_t + j_t * d_p \end{cases} \text{where } \begin{cases} i_t \sim Uniform\{0, d-1\} \\ j_t \sim Uniform\{0, d-1\} \end{cases} \tag{7}$$

4.2 Multi-Armed Bandit Agents

An intuitive way to model market-making is the Multi-Armed Bandit (MAB) problem, where at time t we pull the i-th arms from d bid arms and the j-th arm from d ask arms. The indices of arms indicate the number of ticks away from the mid-price that we ask and bid.

$$\begin{cases} \text{Bid}_t = S_t - i_t * d_p \\ \text{Ask}_t = S_t + j_t * d_p \end{cases} \text{where } \begin{cases} i_t \in \{0, 1, ..., d-1\} \text{ bid arm is pulled at time } t \\ j_t \in \{0, 1, ..., d-1\} \text{ ask arm is pulled at time } t \end{cases} \tag{8}$$

We simplify the model by assuming that the bidding and asking actions are independent to each other, therefore, the market-making problem can be modeled as 2 independent MAB. We also disregard here the dependence of the reward on the environment state and the fact that actions would inherently change the states as per our modelling framework since in MABs the reward only depend on the action. We are aware that this is not fitted for our modelling framework. However, we were curious to see weather MABs would beat the random benchmark.

We assume that each arm corresponds to a distribution of profit. The idea of MAB agents is that with more explorations, we can better estimate the profit distribution associated with that arm. We implemented 4 basic MAB agents, which are discussed below.

4.2.1 Explore-First Agent

Explore-First agent uses the first T_0 steps to randomly pull arms, and the rest steps to pull the best arm with the highest return. We have

$$i_t = \begin{cases} Random\{0, d-1\}, & t \le T_0 \text{ (pull a random bid arm)} \\ arg\max_i\{\frac{\sum_{t'<t} \mathbf{1}(i_{t'}=i) R_t}{\sum_{t'<t} \mathbf{1}(i_{t'}=i)}\}, & t > T_0 \text{ (pull the best bid arm)} \end{cases} \tag{9}$$

$$j_t = \begin{cases} Random\{0, d-1\}, & t \le T_0 \text{ (pull a random ask arm)} \\ arg\max_j\{\frac{\sum_{t'<t} \mathbf{1}(j_{t'}=j) R_t}{\sum_{t'<t} \mathbf{1}(j_{t'}=j)}\}, & t > T_0 \text{ (pull the best ask arm)} \end{cases} \tag{10}$$

We used $T_0 = 0.1 \times T$ so as to use the first 10% as exploration stage and the rest 90% as exploitation stage.

4.2.2 UCB Agent

Upper-Confidence Bound (UCB) agent improves upon explore-first agent by taking into account the confidence of our estimation in the average returns - the more we pull an arm, the more confident we are on this arm. At time t, the upper bound of average return of arm i is given by

$$\text{Estimated return of arm } i \text{ at time } t = \frac{\sum_{t'<t} \mathbf{1}(i_{t'} = i) R_t}{\sum_{t'<t} \mathbf{1}(i_{t'} = i)} + \sqrt{\frac{4log(t)}{\sum_{t'<t} \mathbf{1}(i_{t'} = i) + \Delta}}$$

where $\Delta = 1e - 5$ is a small number to avoid dividing by 0.

The second component of the formulation can be viewed as the confidence interval of sample mean, or it can be viewed as a bonus to encourage exploration.

4.2.3 ϵ-Greedy Agent

The ϵ-greedy agent is another variation of explore-first agent, which enable explorations at anytime. Instead of only exploring for $t \leq T_0$, ϵ-greedy agent explore random arms for probability ϵ and exploit the best arm for probability $1 - \epsilon$.

$$i_t = \begin{cases} Random\{0, d-1\}, & \text{with probability } \epsilon \\ arg\max_i \{\frac{\sum_{t'<t} \mathbf{1}(i_{t'}=i) R_t}{\sum_{t'<t} \mathbf{1}(i_{t'}=i)}\}, & \text{with probability } 1-\epsilon \end{cases} \tag{11}$$

where we used $\epsilon = 0.1$ to be consistent with the $T_0 = 0.1 \times T$ of explore-first agent.

4.2.4 Decay-ϵ-Greedy Agent

Decay-ϵ-greedy agent is a natural extension to the ϵ-greedy agent, where the exploration probability ϵ decays with time, because we want to explore more at first and exploit more towards the end. The decay ϵ is given by:

$$\epsilon_t = \max\{\epsilon_{t-1} - \epsilon_{decay}, \epsilon_{min}\} \tag{12}$$

where we specified $\epsilon_1 = 0.5$, $\epsilon_{decay} = 0.01$ and $\epsilon_{min} = 0.01$.

4.3 Q-Learning Agent

Afterward, we implement a dynamic programming approach to this Sequential Decision-Making process - Q-learning. Mathematically, Q-learning can be represented using the following equation:

$$Q(s, a) \leftarrow Q(s, a) + \sigma \left[r + \gamma \max_{a'} Q(s', a') - Q(s, a)\right] \tag{13}$$

$$\text{where } \begin{cases} Q(s, a) \text{ is the estimated value of taking action } a \text{ in state } s \\ \sigma \text{ is the learning rate} \\ r \text{ is the reward obtained by taking action } a \text{ in state } s \\ \gamma \text{ is the discount factor for future rewards} \\ s' \text{ is the state that the agent transitions to after taking action } a \text{ in state } s \\ \max(Q(s', a')) \text{ is the maximum estimated value of any action } a' \text{ in the next state } s' \end{cases}$$

In the training stage, we use ϵ-greedy strategy, which is to randomly explore the action space with the probability ϵ and exploit the best-estimated action with probability $1 - \epsilon$, i.e.,

$$a_t = (i_t, j_t) = \begin{cases} \sim Uniform\{[0, d-1]^2\} & \text{with probability } \epsilon_t \\ argmax_a\{Q(s_t, a)\} & \text{with probability } 1 - \epsilon_t \end{cases} \tag{14}$$

We decay the probability of exploration ϵ_t by ϵ_{decay} in every step, and we also set a lower bound for ϵ_t as ϵ_{min}. We update ϵ_t as below:

$$\epsilon_t = \max\{\epsilon_{t-1} - \epsilon_{decay}, \epsilon_{min}\} \tag{15}$$

4.4 Proximal Policy Optimization (PPO) Agent

Last, but not least, we decide to examine a PPO Agent in market-making setting which belongs to the family of actor-critic methods and utilizes a trust region optimization approach to update the policy.

The basic idea behind PPO is to constrain the policy update step to ensure that the new policy remains close to the old policy. This is achieved by defining a clipping function that limits the size of the policy update to a specified range. The objective function that PPO aims to optimize is the following:

$$L(\theta) = \hat{\mathbb{E}}_t\Big[\min\big(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\big)\Big] \tag{16}$$

where θ is the parameter vector of the policy, $\hat{\mathbb{E}}_t$ denotes the empirical expectation over a batch of trajectories, $\hat{A}_t$ is an estimator of the advantage function at time step t, and $r_t(\theta)$ is the ratio of the probability of the new policy to the old policy, evaluated at the current state and action. The clipping function ensures that the ratio $r_t(\theta)$ remains within the range $[1-\epsilon, 1+\epsilon]$, which corresponds to a trust region around the old policy.

5 Experiments and Results

5.1 The Simulator

For the purpose of simulation, the probabilistic environment described in Section 1 was used which is the extension of OpenAI gym.

The main goal of the agent is to maximize incremental rewards until market closure at time T, which is at least a non-negative value (minimum success criterion).

5.2 Experiment Setup

In the experiments, we specify the following parameters for the environment and agents:

$$\begin{cases} T = 420 \text{ is the minutes in the entire one trading day} \\ Q = 4 \text{ is the maximum position allowed for the market maker} \\ d = 5 \text{ is the number of ticks away from the mid price market maker can quote} \\ T_0 = 0.1 \times T = 42 \text{ is the exploration period for explore-first agent} \\ \epsilon = 0.1 \text{ is the exploration probablity for } \epsilon-greedyagent \\ \epsilon_1 = 0.5, \epsilon_{decay} = 0.01, \epsilon_{min} = 0.01 \text{ are the decay-}\epsilon\text{-greedy agent params} \\ \sigma = 0.1 \text{ is the learning rate of Q-learning agent} \\ \gamma = 0.99 \text{ is the discount factor for future rewards of Q-learning agent} \\ \epsilon_1 = 1.0, \epsilon_{decay} = 0.3, \epsilon_{min} = 0.01 \text{ are the exploration params for Q-learning agent} \end{cases}$$

In each experiment, we implement all agents for $T = 420$ steps and repeat the experiment for 100 times. In the next section, we will present the average final reward for 100 experiments.

5.3 Experiment Results

The statistics of cumulative rewards (averaged over 100 experiments) are shown in table 1. After fine-tuning the model, Q-learning and PPO agents have a higher average reward than the multi-armed bandits. They as well substantially outperform our random benchmark strategy.

To demonstrate how the algorithms perform, we plot the cumulative rewards over time in figure 1.

Additionally, in order to understand better the behavior of best-performing agents - Q-Learning and PPO - we plot their spread over time in their last episodes, respectively figure 2 and 3. As we can observe they have similar behavior with a relatively small, conservative spread when compared to the baseline (4).

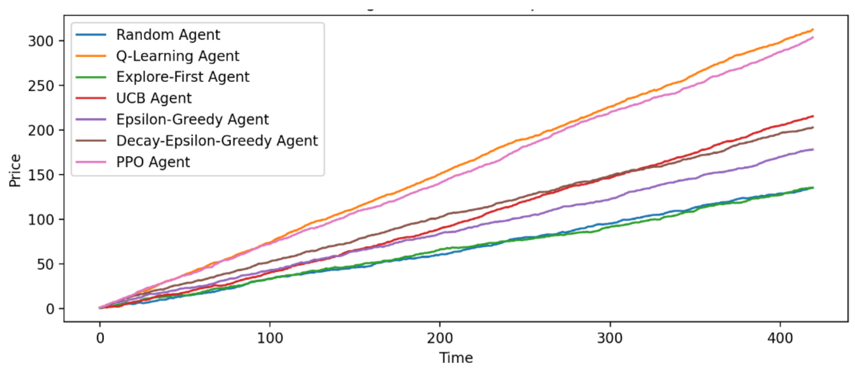

Figure 1: Cumulative Reward Over Time of various algorithms

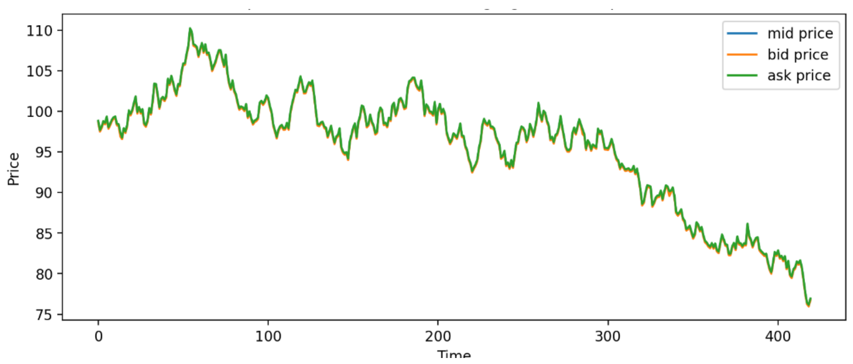

Figure 2: Spread over time for Q-Learning Agent in the last episode

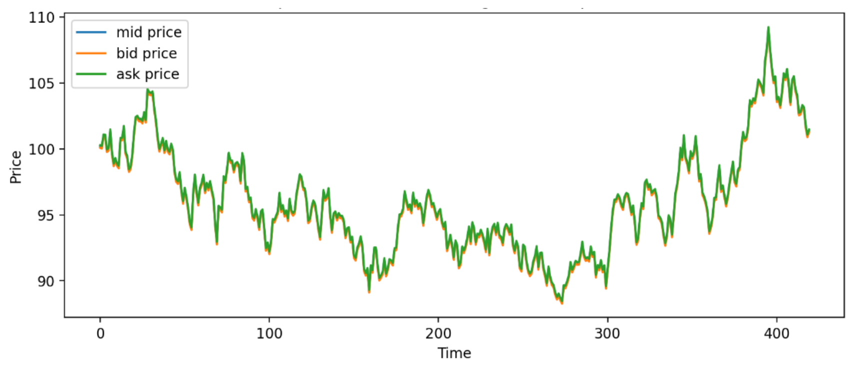

Figure 3: Spread over time for PPO in the last episode

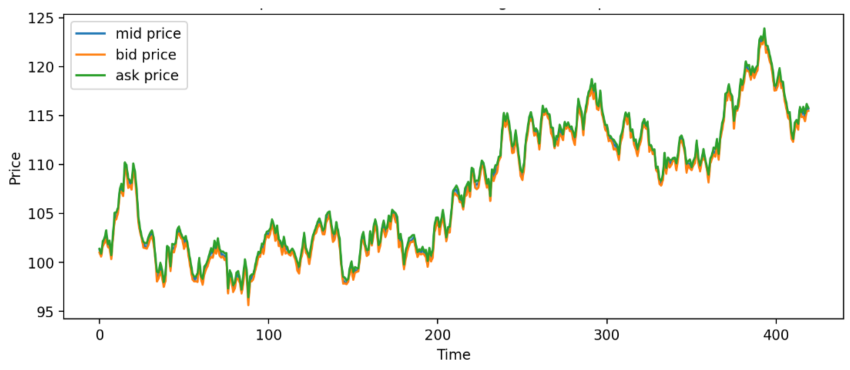

Figure 4: Spread over time for Random Agent in the last episode

Rewards Metrics	Random Agent (baseline)	Explore- -First Agent	UCB Agent	Epsilon- -Greedy Agent	Decay-Epsilon- Greedy Agent	Q-Learning Agent	PPO Agent
Mean	135.21	135.22	215.20	178.00	202.71	312.48	303.71
Std.	66.72	121.47	92.25	90.74	127.73	73.60	71.45
Min	-154.63	-183.73	-17.90	-63.19	-51.22	118.05	110.24
25%	95.81	61.20	149.07	121.52	101.77	263.66	260.64
50%	137.07	122.22	230.35	183.30	197.33	322.53	299.41
75%	169.95	220.16	284.83	243.27	310.88	355.58	355.31
Max	265.86	412.82	411.44	421.49	479.79	484.43	443.72

Table 1: Cumulative Rewards of 10 Experiments

6 Conclusions

Based on the cumulative rewards of the different market-making agents, we can see that the Q-learning and PPO agents achieved the highest mean rewards, with Q-learning having a slightly higher mean than PPO. UCB also performed well with a mean reward of 215.20, while the other agents had lower mean rewards. However, it is important to note that the results had high variance and are not very stable depending on the simulation. Based on the market-making case, we can observe that the choice of algorithm plays a crucial role in having an optimal strategy, especially in an industry where small margins play a critical role.

The limitations of the proposed approaches are also worth mentioning. In real-world multiple agents act in the same environment on multiple assets which makes the problem even more complex and dynamic (in comparison to our approach of one agent and one asset). Additionally, from an algorithmic point of view, all of the proposed methods rely on a fixed set of parameters, such as learning rates and exploration probabilities. In practice, these parameters may need to be tuned over time to ensure optimal performance, which can be time-consuming and difficult. Furthermore, the methods assume a stationary environment, which may not be true in real-world setting where market conditions can change rapidly.

To address these limitations, one potential approach would be to use a multi-agent setting which unfortunately requires much more computation power. Moreover, in the future adaptive parameter tuning techniques, such as grid search or Bayesian optimization, could be used to automatically search for optimal parameter settings based on the performance of the agents in the market. This could reduce the need for manual parameter tuning, which can be very time consuming and difficult in the long run. Last, but not least, with higher computational power ensemble learning can be used to combine the predictions of multiple algorithms and improve the overall performance. This can help to mitigate the high variance observed in the results and provide more reliable performance.

7 Contributions

The optimal market-making project was a collaborative effort among Megi, Shiyin, and Michal. All team members contributed equally to problem formulation, report writing, and presentation. However, each member took the lead on certain aspects of the project. Megi was responsible for developing the Random Agent, Explore-First Agent, and Q-learning algorithms. Michal developed the UCB and PPO agents. Shiyin focused on the Epsilon-Greedy and Decay-Epsilon-Greedy Agents. It is important to note that despite these specific roles, the team worked closely together, regularly sharing ideas, providing feedback, and ensuring that all work was completed in a supportive and collaborative manner.

References

[1] Jacob Abernethy and Satyen Kale. “Adaptive Market Making via Online”. In: *Proc. of NIPS* (2013).

[2] Sebastian Jaimungal Álvaro Cartea and Jason Ricci. “Buy Low Sell High: A High Frequency Trading Perspective”. In: *Journal on Financial Mathematics* (2014).

[3] Tanmoy Chakraborty and Michael Kearns. “Market Making and Mean Reversion”. In: *Proc. of EC.* (2011).

[4] Avellaneda Marco and Sasha Stoikov. “High Frequency Trading in a Limit Order Book”. In: *Quantitative Finance* (2008).

[5] Thomas Spooner et al. “Market Making via Reinforcement Learning”. In: *CoRR* abs/1804.04216 (2018).