DQN For Market Making

May 12, 2021

1 Problem Statement

In this project we're working on building Market Maker strategy using SHIFT API. For a Objective function:

$$\text{Max} \quad \mathbb{E}\left[e^{-\gamma W}\right]$$

Where,

$$W = x + qS$$

We use W to represent investor's wealth, and it consists of two elements. One is cash x, the other is stock holding qS. q is the number of sharing holding and S is the stock price. Also, the investor needs to decide their own risk aversion.

For this optimazation problem we're facing some constraints. Frist, we're focusing on single trading day. So the optimization period is only 6.5 hour data. And we have limit budget which is \$1,000,000. Also we are constrained by the technical problems. We're using Stevens High Frequency Trading(SHIFT) platform to implement our strategies. So we can't do anything beyond the platform. The details are covered in the next section.

2 Data Description

SHIFT(Stevens High Frequency Trading) is an platform providing users an Python API. It simulation Dow Jones 30 stocks price data for about 20 years. And these simulation data are in ticker level. SHIFT platform updates the data every 0.5 seconds. So this is also the upper bond for our strategies.

Also, They provide three different market types: high volatility, low volatility and financial crisis which involves a huge drop within a single trading day.

This platform utilizes historical stock price data and map from them to their simulation data. So our strategies would not has any impact on the stock market. That means even if we put a large amount of orders in the market, it won't change the prices in the next few minutes. And there're limited observation listed in the limit order book. So we have to truncate our strategies to incorporate these constraints.

3 Methodology

3.1 Benchmark

We consider a naive Market Maker strategy as benchmark. For each time, we put a binding quote(bid and ask) for a fixed spread. Also, we are trading fixed shares of stock for each time. To implement this static approach, we need to calibrate three parameters, bid price, ask price and number of trading shares.

We first calibrate the spread of bid/ask price over trading price. Optimal bid price

$$S_0 - B = \frac{1}{2}\gamma\sigma^2 \approx 0.01$$

Optimal ask price

$$A - S_0 = \frac{1}{2}\gamma\sigma^2 \approx 0.01$$

Where A is ask price and B is bid price. Although we are looking at a static model, we will use the last trading price for S_0 for calibration. The σ is the volatility of the stock price and γ as before is investor's risk aversion.

Note that this calibration depends on trading stock. Different stock has different liquidity thus different trading volume and bid-ask spread.

3.2 MDP formulation

To better formulate our Markov Decision Process, we need to initiate our model objective: which is providing real-time guidance for how to manage the firm's portfolio of limit buy and sell orders on the LOB so as to maximize the expected net profit, also we need to reduce the mismatch between the amounts bought and sold; besides, our goal is to pursue sufficiently high Sharpe ratio which measures the return of an investment compared to its risk , it has 3 different levels, listed as follows: (acceptable: > 1; very good: > 2; excellent: > 3) thus we make the following assumptions:

1. The model is a finite-horizon discrete-time MDP
2. At most one buy and one sell order can rest on the best bid price and the best ask price, respectively.
3. Backtest using the simplest strategy

Timing of LOB events is as follows: At time t, We take an action (add limit order, cancel limit order or do nothing); within period t to t+1, market sell and buy orders arrive, which may be fulfilled or lost, also limit orders are added or canceled by other participants, after those two events within time t to t+1, we then arrive at a new state at time t+1.

In the next step, we define our (State Space, Acction Space, Post-decision state) as follows:

The first, State Space

1. Price levels: $\mathcal{P} := \{1, 2, \ldots, n\}$, doing liner scaling transform for SPY price
2. Our limit orders: $|R_{tp}| \in \{0, 1, 2...\}$ (conservatively assume resting at the back of the queue) Other participants' limit orders: $\left|R^2_{tp}\right| \in \{0, 1, 2, \ldots\}$ LOB state variable: $R_t = \left(R^1_{tp}, R^2_{tp}\right)_{p\in\mathcal{P}}$
3. Best bid and ask prices: $\beta_{R_t}, \alpha_{R_t}$

The second, allowable actions

1. Having or not having one buy (sell) order at the best bid (ask) price
2. Action space:
$$A_t = (A_{t1}, A_{t2}) \in \mathcal{A} := \{(0,0), (0,1), (1,0), (1,1)\}$$
3. Post-decision state:
$$R^{a2}_{tp} = R^2_{tp}, R^{a1}_{tp} = \begin{cases} 1, & \text{if } A_{t1} = 1, p = \beta_{R_t} \text{ or } A_{t2} = 1, p = \alpha_{R_t} \\ 0, & \text{otherwise} \end{cases}$$

After defining the above states, we need to consider information Influx from the market and other participants which influence our states within the period from time t to t+1, thus we define the following terminologies:

1. Market buy and sell orders:
$$\widehat{D}^{MB}_t, \widehat{D}^{MS}_t \Rightarrow R^m_{tp} = \left(R^{m1}_{tp}, R^{m2}_{tp}\right)_{p\in\mathcal{P}}$$
2. Orders and cancellations from other participants:
$$\hat{O}_t = \left(\hat{O}_{tp}\right)_{p\in\mathcal{P}}, \hat{C}_t = \left(\hat{C}_{tp}\right)_{p\in\mathcal{P}} \Rightarrow R^o_{tp} = \left(R^{o1}_{tp}, R^{o2}_{tp}\right)_{p\in\mathcal{P}}$$
3. Pre-decision state for the next decision epoch:
$$R_{t+1} = \left(R^{o1}_{tp}, R^{o2}_{tp}\right)_{p\in\mathcal{P}}$$

The final goal is to maximize our objective function which is the sum of profit and loss (PnL) relative to the mid price and penalty of mid price movement.

$$V\left(R_t, A_t, \widehat{D}^{MB}_t, \widehat{D}^{MS}_t, inv_t\right) := \text{PnL}\left(R_t, A_t, \widehat{D}^{MB}_t, \widehat{D}^{MS}_t\right) + inv_t \cdot \Delta_{m_t}$$

in which Profit and loss (PnL) is defined by:

$$\text{PnL}\left(R_t, A_t, \widehat{D}_t^{MB}, \widehat{D}_t^{MS}\right) := E^{\beta} \cdot (m_{R_t} - \beta_{R_t}) + E^{\alpha} \cdot (\alpha_{R_t} - m_{R_t})$$

here E^{β} is 1 if we got our limit buy order executed at time t, otherwise 0; E^{α} is 1 if we got our limit sell order executed at time t, otherwise 0; $m_{R_t} - \beta_{R_t}$ represents the difference between middle price and best limit buy price at time t $\alpha_{Rt} - m_{R_t}$ represents the difference between best limit sell price and middle price at time t inv_t represents the difference between cumulative amount bought and cumulative amount sold at time t

3.3 State Aggregation and Stochastic Approximation:

In this step we use state aggregation to abstract the main features of the above defined states:

1. bidSpeed: $BS \in \{0, 1\}$, if the market sell orders exceed the book size at the best bid price;
2. askSpeed: $AS \in \{0, 1\}$, if the market buy orders exceed the book size at the best ask price;
3. avgmidChangeFrac: $MF \in \{0, \pm 1, \pm 2\}$, the relative change in the average mid price, (the threshold hyper-parameter $f \in [\mathbf{0}, \mathbf{1}]$ need to tuned)
4. invSign: $IS \in \{0, \pm 1, \pm 2\}$, the side and magnitude of inv_t (the threshold hyper-parameter $I \in [\mathbf{0}, \infty)$ needs to be tuned)
5. cumPnL: cumPnL $\in \{0, 1\}$, if the cumulative PnL is large or small (the threshold hyper-parameter $\boldsymbol{P} \in (-\infty, \infty)$ needs to be tuned)

here feature (1) and (2) represents the market volatility level; feature (3) decides which side of the order book we should place our liit order, and feature (4) represents our inventory level is high or low; feature (5) is our cumulative profit and loss over time, (4) and (5) are closely linked with past information and should be updated at each time step. Here we define our State aggregation function as the following:

$$G\left(R_t, \text{inv}_t, pnl_t\right) := (BS_t, AS_t, MF_t, IS_t, CP_t)$$

after the aggregation, we calculate the aggregated state space size: $2 \times 2 \times 5 \times 5 \times 2 = 200$, which is relatively small, it means our training will have good convergence speed.

3.4 Q-learning Algorithm:

To solve this RL problem, we describe the algorithm as following.

For each iteration $n \in [\bar{N}]$:Randomly select some "aggregated state-action" pairs to update.

For each selected "aggregated state-action" pair (s, a) :

1. Randomly select a sample path ω with initial aggregated state s, and associated full state by R_n^s
2. Update the full state and aggregate to $\bar{s} = G\left(R_{n+1}^S, inv_{n+1}^S, pnl_{n+1}^S\right)$;
3. Update the Q factor:

$$Q_{n+1}(s,a) = (1-\alpha_n(s,a)) \cdot Q_n(s,a) + \alpha_n(s,a) \cdot \left(V(\omega) + \gamma \max_{v \in \mathcal{A}_s} Q_n(\bar{s}, v)\right)$$

   where $V(\omega)$ is the "PnL + penalty term" obtained from sample path ω,

And $\alpha_n(s,a) := \frac{\alpha_0}{\text{updates of } (s,a)}$ is the learning rate
For any full state R_t with inventory and PnL at inv and pnl, the optimal action is:

$$\underset{a}{\text{argmax}} Q_{\bar{N}}\left(G\left(R_t,\ \text{inv},\ \text{pnl}\right), a\right)$$

4 Performance Analysis

We first test our static market maker strategy as benchmark for two different market types. As you can see from the picture below, the market maker PnL is highly sensitive to the market volatility. If the market volatility is high, then the strategy is keeping earning money. However, if the market volatility is extremely low, the strategy is keeping losing money. It because low volatility make trading price very hard to reach on or beyond the bid/ask quote prices.

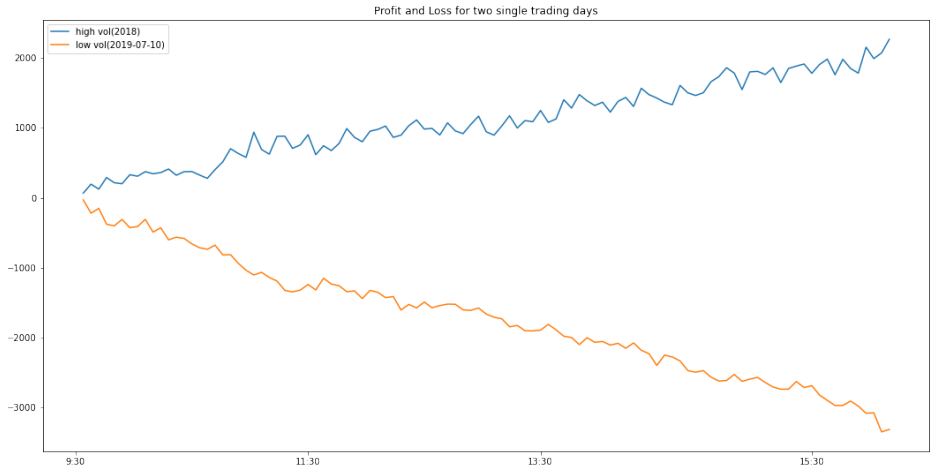

Figure 1: PnL comparison between low vol and high vol

This static model is not good enough due to the sensitivity of market volatility. Also it can not achieve the optimal performance given the high market volatility. The test of our advanced market maker model using Reinforcement Learning algorithm would mitigate this flaw and optimize the profit given any market situation.

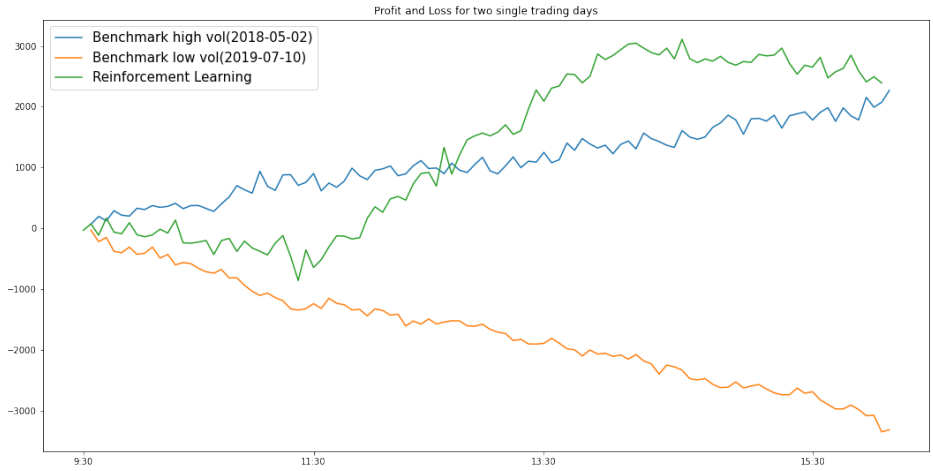

Figure 2: PnL comparison between RL MM and benchmark

As shown above, we test our RL market maker strategy on a high volatility trading day. And RL case gives us a much better performance than static one. Although it lose some money at the beginning, it catches up quickly and beat the benchmark at the end. This is because, we are using an online learning algorithm. So that it may not achieve the best at the very beginning. It still learning from the market. But it learns quickly and apply what it learned to beat the benchmark.

We trained six Q tables for each hour from $9:00$ a.m. $-14:30$ p.m. $(9:00-10:00, 10:00-11:00,$, 11:00-12:00, 12:00-13:00, 13:00-14:00, 14:00-14:30) in the Shift API system, the result Q table is as follows

Aggregated book state					Suggested action	
bidSpeed	askSpeed	avgmidChangefrac	avgSign	CumPnL	Action-bid	Action-ask
0	0	-2	-2	0	0	0
0	0	-2	-2	1	1	0
0	0	-2	-1	0	0	1
0	0	-2	-1	1	0	1
...	...	...	...	...	...	...

Table 1: Q table after training

After train our model using tabular Q-learning, we then have the trading policies from the resulting Q table:

Our agent will make best limit buy and best limit sell order when the volatility is high(reflected

on the BidSpeed and AskSpeed), and when the moving speed of the middle price is high and our PnL level is high, we tend to put only limit order on one side of the order book which is more active; e.g., add limit buy orders on a sell-heavy market;

Market-making is not directional, the trading policy will not do trend trading.Also, the optimal strategy keeps inventory near zero.

5 Future Work

- Incorporate correlated pairs of stocks to do market making
- Increasing the size of limit buy or limit sell orders and refit the model
- Using partially observed information deep reinforcement learning algorithm to improve our model
- Incorporate our market participants' states and formulate a multi-agent deep reinforcement learning model and do the mean field game approximation to train our models

```python
import asyncio
import pandas as pd
import numpy as np
import shift
import time
trader = shift.Trader('market_maker_101')
trader.connect('initiator.cfg', 'u24G58hcgg')
# trader.connect('initiator.cfg', 'llc101')

trader.sub_all_order_book()


# time.sleep(300) data has latency, always get limit sell at price 55 filled
def clearAll(ticker='SPY'):
    trader.cancel_all_pending_orders(timeout=3)
    try:
      a=trader.get_best_price(ticker).get_ask_price()
      b=trader.get_best_price(ticker).get_bid_price()
      x=trader.get_last_price(ticker)
    except:
      x = trader.get_last_price(ticker)
      a = max(180,a + 0.02)
      b = x - 0.01
    for item in trader.get_portfolio_items().values():
        trade_share = int(item.get_shares() / 100)
        if item.get_shares() > 0 and b > 180:
            trader.submit_order(
                shift.Order(shift.Order.LIMIT_SELL, item.get_symbol(),
                            trade_share, max(180,a - 10)))
        if item.get_shares() < 0 and a < 300.01:
            trader.submit_order(
                shift.Order(shift.Order.LIMIT_BUY, item.get_symbol(),
                            abs(trade_share), b + 10))
        else:
            pass
    # clear all orders in the limit order book

# when the buying bower is below 40000, do clearALL()
def triggerClearCheck(threshold=37):
  pending_size=np.sum([order.size for order in trader.get_waiting_list()])
  -np.sum([order.executed_size for order in trader.get_waiting_list()])
  try:
    item = list(trader.get_portfolio_items().values())[0]
    trade_share = int(abs(item.get_shares()) / 100)
  except:
    trade_share = 0
  if trade_share+pending_size >= threshold or pending_size >= 30 or trade_share >= 24:
    clearAll()
    time.sleep(1)
  pass
```

```python
def makerTrade(ticker='SPY'):
    try:
      b=trader.get_best_price(ticker).get_bid_price()
      a=trader.get_best_price(ticker).get_ask_price()
      x=trader.get_last_price(ticker)
    except:
      x = trader.get_last_price(ticker)
      b = x - 0.01
      a = b + 0.02
    if a < 180.00:
        trader.cancel_all_pending_orders(timeout=3)
        trader.submit_order(shift.Order(shift.Order.LIMIT_BUY, ticker, 3, 181.00))
    elif b > 300.00:
        trader.cancel_all_pending_orders(timeout=3)
        trader.submit_order(shift.Order(shift.Order.LIMIT_SELL, ticker, 3, 299.00))
    else :
        # Sometimes with the above conditions limit sell price like 50 still get filled
        trader.submit_order(shift.Order(shift.Order.LIMIT_BUY, ticker, 3, min(b,x)))
        time.sleep(0.2)
        trader.submit_order(shift.Order(shift.Order.LIMIT_SELL, ticker, 3, max(180,a,b+0.02,x+0.02)))
        time.sleep(1.3)
        pass

if __name__ == "__main__":
    for k in range(115):
        for i in range(120):
            triggerClearCheck(44)
            makerTrade()
        time.sleep(0.2)
        clearAll()


    print("Buying Power\tTotal Shares\tTotal P&L\tTimestamp")
    print("%12.2f\t%12d\t%9.2f\t%26s" % (
        trader.get_portfolio_summary().get_total_bp(),
        trader.get_portfolio_summary().get_total_shares(),
        trader.get_portfolio_summary().get_total_realized_pl(),
        trader.get_portfolio_summary().get_timestamp(),
    ))

    print(
        "Symbol\t\t\t\tType\t  Price\t\tSize\tExecuted\tID\t\t\t\t\t\t\t\t\t\t\t\t\t Status\t\tTimestamp"
    )

    trader.disconnect()
```