Neural LOB Research Report

Kang Li
Mathematical Institute, Unversity of Oxford
kang.li@maths.ox.ac.uk

September 1, 2022

Table of contents

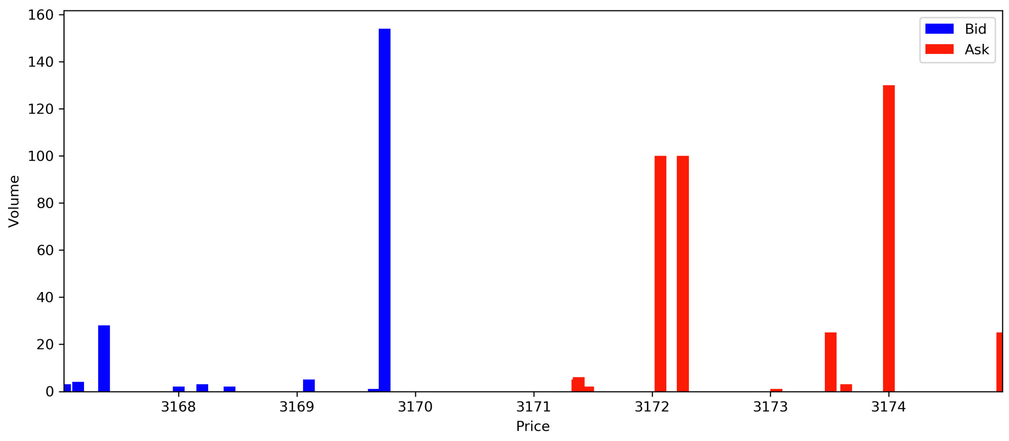

Figure 1: A snapshot of the LOB of AMZN with ten levels of bid/ask prices.

Table 1: An example of the LOB from the dataset

Bid Level	One	Two	Three	Four	Five
Bid Price	3169.74	3169.65	3169.1	3168.43	3168.2
Volume Available	154	1	5	2	3

1 Dataset Description

In our analysis, we use data from AMZN covering the ten price levels on both sides of the book from April 1st 2021, to April 30th 2021, with the interval between two price levels to be 1 cent. We want to mention that the data we use is between 09:30 and 16:00.

There are five types of orders in the order flow data, which are as follows. Submission is the submission of a new limit order. Cancellation is the partial deletion of a limit order. Deletion is the total deletion of a limit order. When the orders in the LOB are deleted or cancelled, the prices might move up or down to the next level of the LOB. Moreover, visible execution is the execution of a visible limit order. Finally, hidden execution is the execution of a hidden limit order. *Hidden* here refers to the properties of orders that do not display their quantity.

In Figure 2, we can find that over 40% of the order flows are of the cancellation type. Only about 10% of the limit order was finally filled. The reason for this phenomenon is that when an agent submits its own order, if a new order enters, it will lead to a change in the mid-price. At this time, in order to catch up with the best mid-price, the agent would tend to cancel the order that has been submitted before and then revise the order based on the new observation of the up-to-date order book.

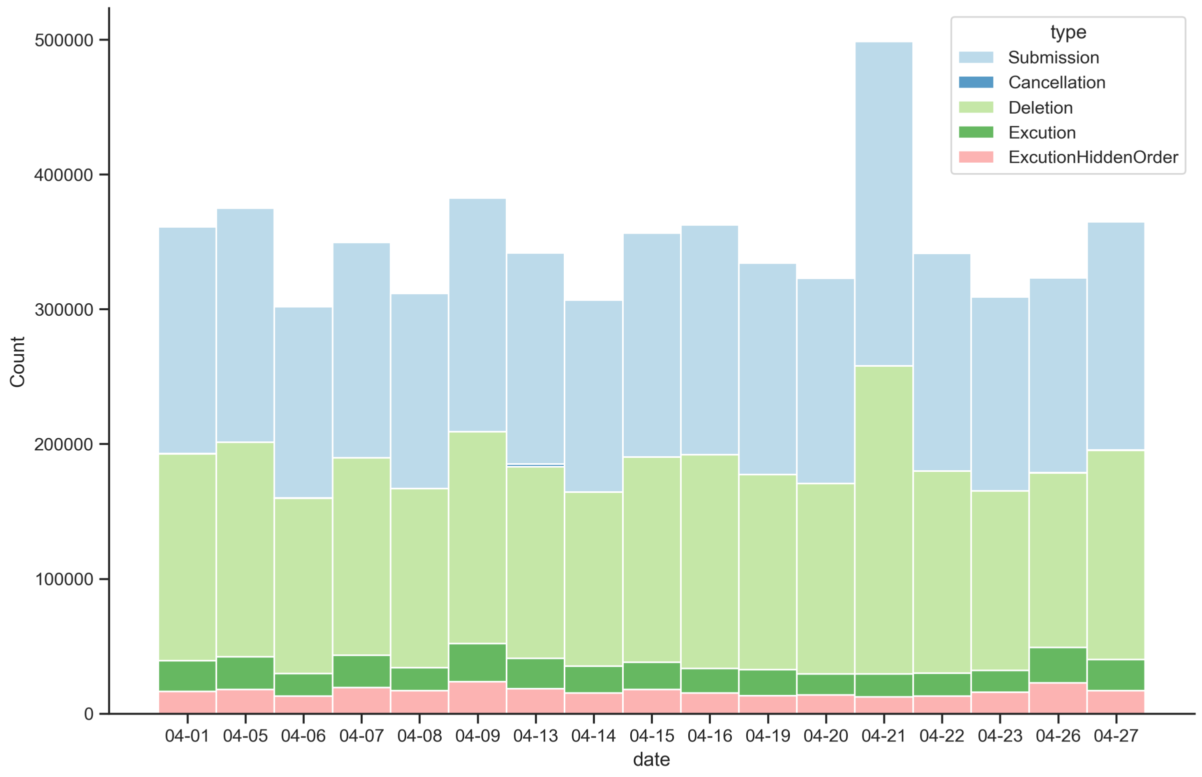

Figure 2: The Components of the Order Flows

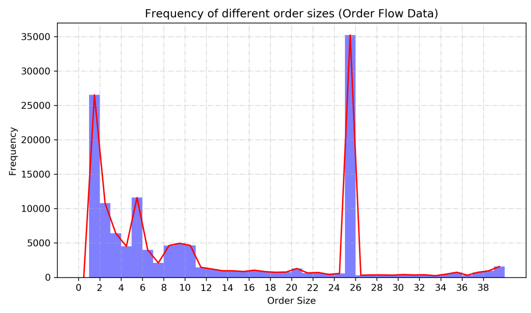

Figure 3: Order Size Frequency of the Order Flow

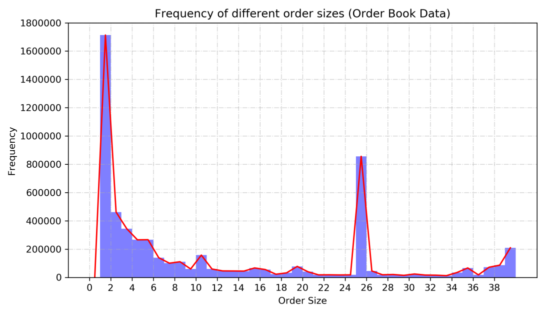

Figure 4: Order Size Frequency of the Limit Order Book

2 Gym_trading

we implemented a gym-interface trading environment named gym_trading for training the deep learning algorithms in it.

The average time interval between two timesteps is 50 miliseconds. As a result, an env with a fixed horizon of 512 means selling stocks within 25.6 seconds. And an env with a fixed horizon of 2048 means selling stocks within 102.4 seconds. And a 10 minutes contains 12000 steps. If we do not want to take so much steps into consideration, use the **Flag.skip** parameter to skip steps. For example, if we want to make a dicision every 1 seconds, the **Flag.skip** should be set to 20. And if we want to make a dicision every 1 minute, the **Flag.skip** should be set to 1200.

We utilize the orderbook info from the Lobster. And the incoming order can be calculated via the difference between to timesteps. And the **env.step(num)** means submitting the market order with

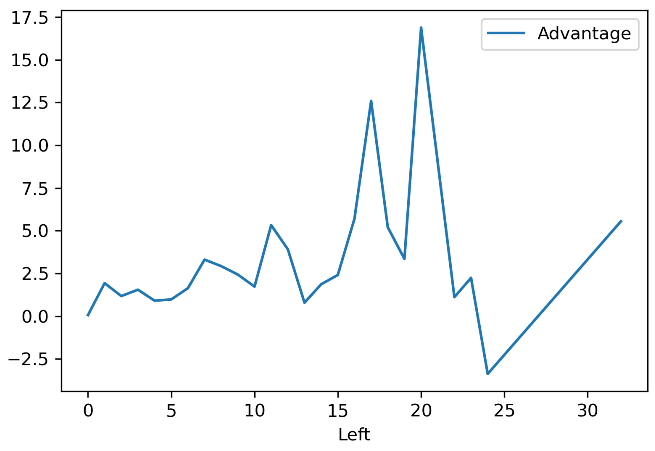

Figure 5: The Advantage over the Baseline(dollar), (512, 500, 1)

quantity **num**.
One potential problem of this setting is that the distribution of the orders varies according to different **env.reset()**. The phenomenon of the **epoch_reward** fluctuate too much may be due to the unstable distribution of the training data for each epoch, see Figure 12.

The architecture of the gym_trading looks like the follows:

```
gym_trading
|data
| |data_pipeline
|envs
| |broker
| |match_engine
| |base_environment
|train
| |train_gym_trading
```

More details can be found in the **Appendix**.

3 Experiment Results

3.1 Curves during Testing

3.2 Comparision between Training Based on Two Different Baselines

As has been shown in the Figure9, the first figure is based on the init_revenue, and the second is based on the twap.
Figure 10 shows other curves during the training. **(max_horizion, num_to_liquidate, skip)**

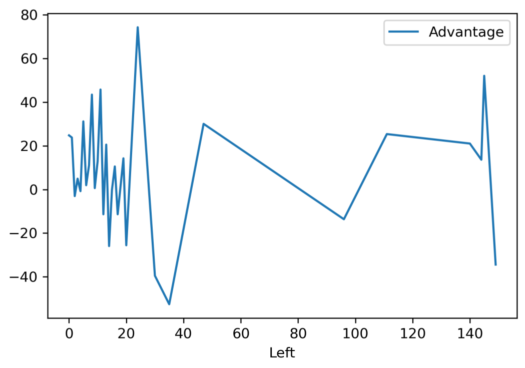

Figure 6: The Advantage of RL over the Init Revenue(dollar), (512, 500, 1)

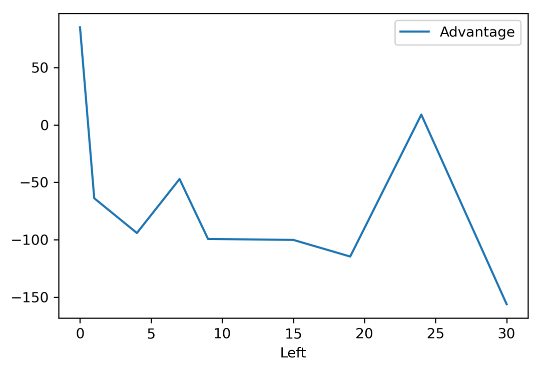

Figure 7: The Advantage of baseline over the Init Revenue(dollar), (512, 500, 1)

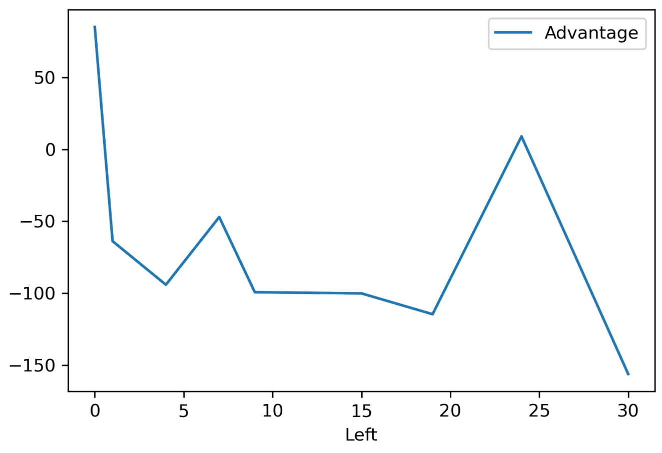

Figure 8: The Advantage over the Baseline based on the Init Revenue(Percentage), (512, 500, 1)

3.3 Tuning the RecurrentPPO

3.3.1 Learning Rate

As has been shown in the Figure11, I applied a decay learning curve to the training.

3.3.2 Approximate KL

Set an upper bound for the approximate KL to early stop during training. As I found the clip in the PPO hasn't really realised the **Trust Region**.

3.3.3 Problems during Training

The first problem is stuck in one epoch_reward, it is perhaps caused by the always choosing the same actions.

As shown in the Figure 12, the second problem is that it seems not converge during training.

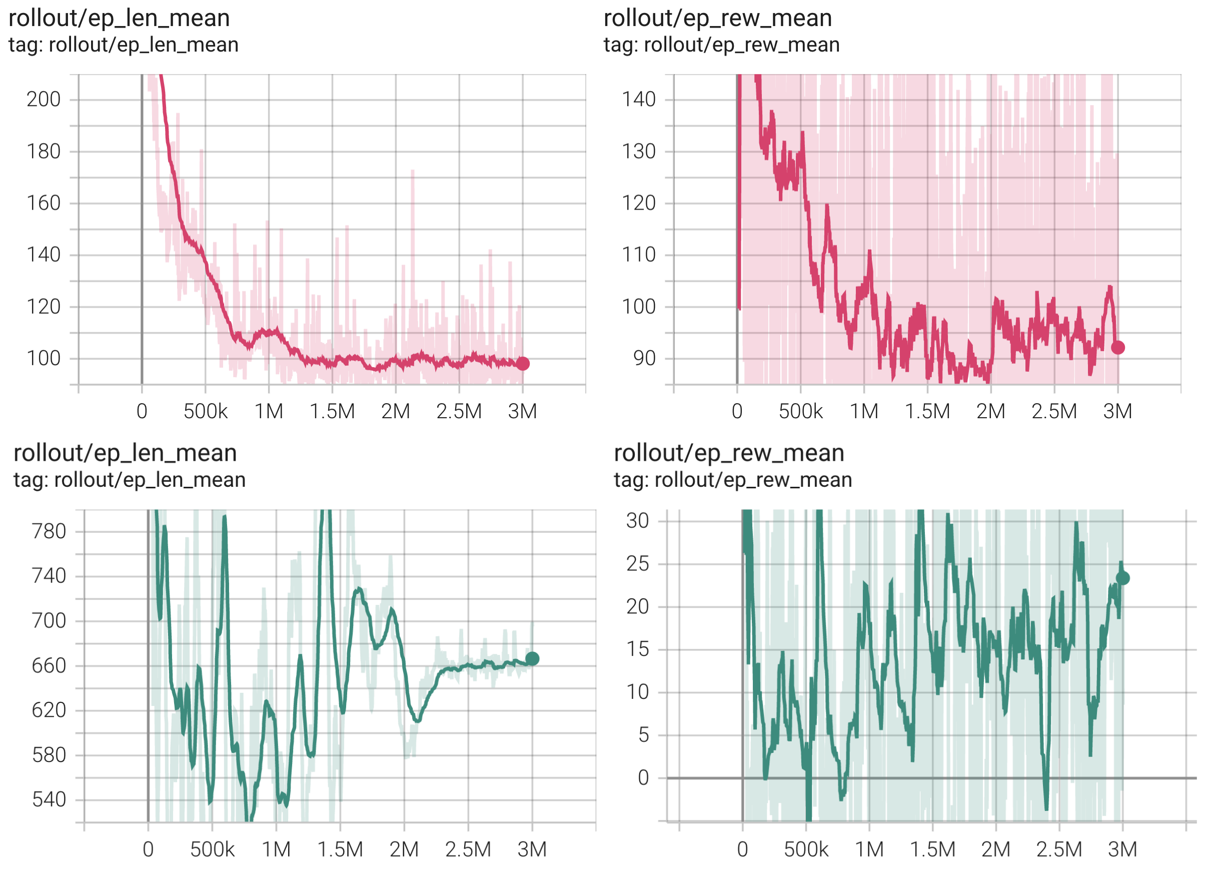

Figure 9: The Training Curves with Smoothing equals 0.96

4 Improving Learning

4.1 Leveraging the Baselines

4.1.1 Imitation Learning

The most advantage of applying imitation learning to this problem is that it can reduce the coupling between codes. As we can use the **stable-baselines** and **imitation** from the Human-compatible AI. The training interface would looks like the example in **Appendix**.

4.1.2 Residual Policy Learning

The main idea of the algorithm [3] is to design a neural network to learn the info that cannot be represented by the baselines algorithms. In the process of training Neural-Liquidators, due to the long training horizon, it is very difficult to train an efficient reinforcement learning agent. For example, if we want to train an agent to liquidate within 10 mins, there are over 10k time steps to perform actions. However, in the field of optimal control, there are already many excellent optimal liquidation algorithms that can be used. Thus, it is helpful to leverage the previous baselines.
Actually, we should refine this question even more. For example, the agent should actually predict the rise and fall of the next best buy at each step. Based on this information, a judgment is made and a policy is formed. For example, if the next best buy goes up, the agent's action should be

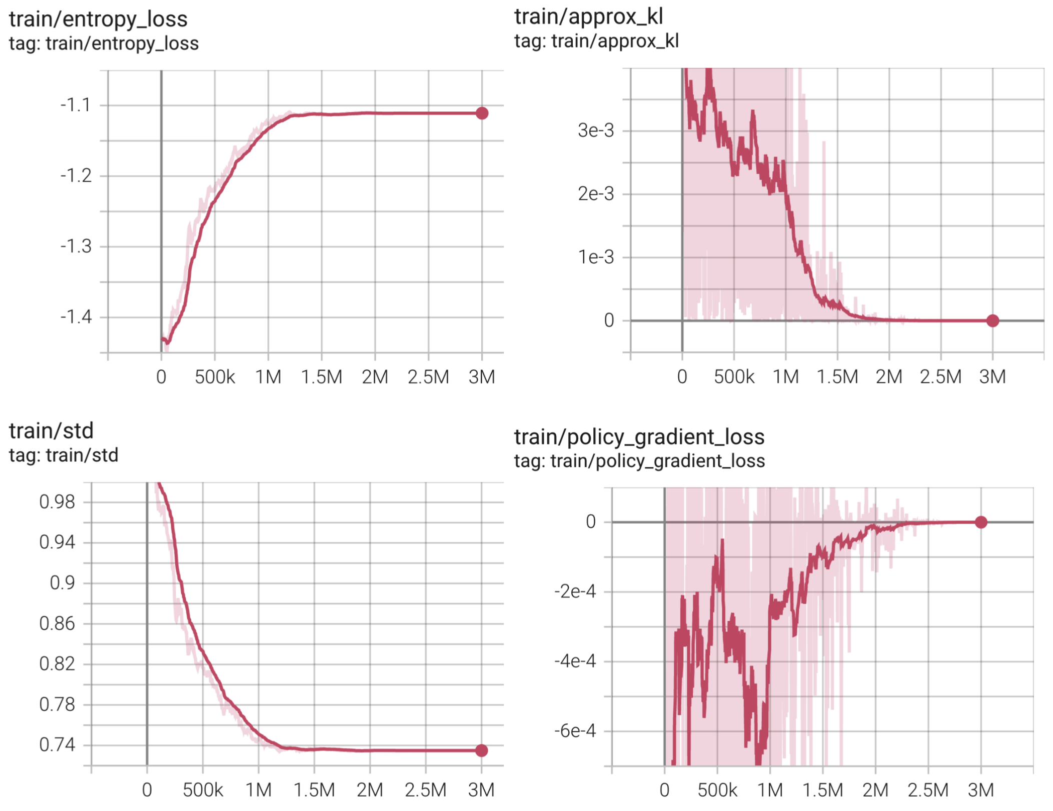

Figure 10: The Curves During Training

positive; if the next best buy goes down, the action of this step should be 0, which means that the agent needs to leave the order for the next time step. The agent's network should incorporate this policy before training. The direction we can try is residual policy learning.

4.2 Remember the Key Step in the Long Trajectories

4.2.1 Recurrent PPO

PPO Combined with LSTM.

4.2.2 Several Observations Combined as One

Like Atari, we can treat several observations from multiple time steps as one observation.

4.2.3 Attention Based Algorithms

There are two examples of RL-Transformer Algorithms: Adaptive Transformer (Online) [1] and Decision Transformer (Offline).

In addition, due to the long horizon, we cannot learn the info earlier before in the trajectories. The sota algorithm on it is the decision transformer. We casts the problem of RL as conditional sequence modeling. The comparison between these two are as follows. The traditional RL fits value functions or compute policy gradients. And the new transformer method outputs the optimal actions by a transformer. By conditioning an autoregressive model on the desired return (reward), past states,

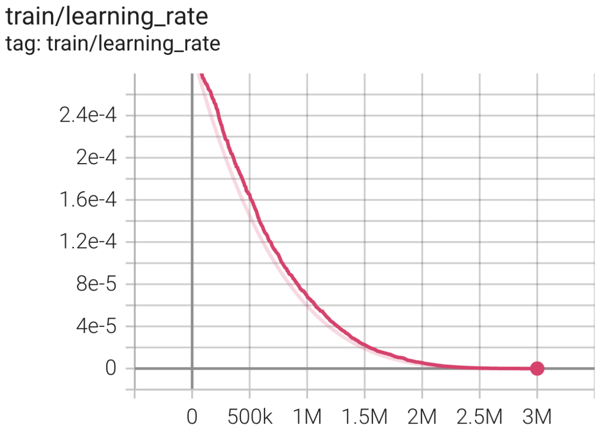

Figure 11: The Learning Rate used in the experiments

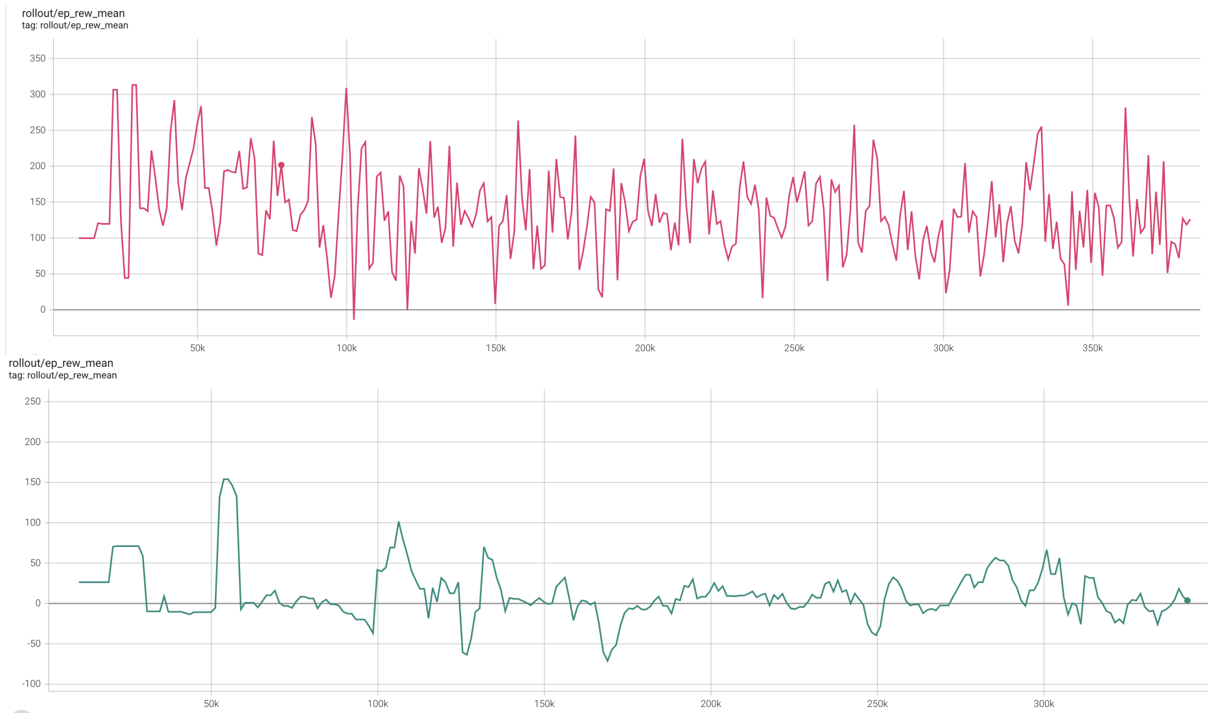

Figure 12: The Reward of Epochs During Training

and actions, the Decision Transformer model can generate future actions that achieve the desired return.

5 Application: Market Making in FX or Bonds Market

Actually, there are two important problems in high-frequency trading. One is optimal execution, including optimal liquidation and optimal acquisition. The other one is market making, which is the hardest and the most profitable, especially in the foreign exchange market. Foreign Exchange (forex or FX) is the trading of one currency for another. For example, one can swap the U.S. dollar for the euro. And FX is the blood vessel through which various macro assets flow.

The optimal executing problem is an easy start and meaningful try to get prepared for the hard problem: market making. In contrast to submitting market orders on one side in the optimal execution, namely orders only with the parameter of how much to buy or sell, the market-making problem is more complex. It requires us to put sell and buy limit orders on both sides to serve as the trading opponent for all the incoming security brokers. The limit orders here are orders with three parameters: **direction**(buy or sell), **price**(waiting for what price to be executed), and **quantity**(how many stocks to be executed). Here is an example showing how market making makes profits.

For bond A, some people want to sell it for 8 pounds, and some people want to buy bond A for 10 pounds. If a market maker offers to buy A for 8.5 pounds from investors who want to sell A, and then resell it to investors who want to buy it for 9.5 pounds. If both sides get matched successfully, the market maker can make a profit of 1 pound spread.

The market maker should put orders near best buy and best sell prices to keep track of the mid-price of one security. Then it will frequently submit orders and cancel the remaining unexecuted parts to make sure the price is tightly around the real mid-price. So it is one reason why called high-frequency trading. As in very short period, the price moving of the limit order book has its trend, it is possible to predict the up or down of the best buy price or best sell based on the multi-factor models. As a result, many high frequency trading hedge fund can have a profit over 100% to 200% per year.

In this problem, **Multi Asset Trading** and **Multi Agents Trading** are also tough problems which are hard for mathematical modeling. But we can apply deep rl to try to solve it. For example, we can apply multi agent rl to solve this kind of problem. In this setting, two or more agents cooperate with each other. E.g. The macro agent optimizes the decision to buy, sell or hold the asset. Micro-agent optimizes limit orders within limit orders. To be more specific [2]:

1. Macro-agent: The macro-agent is given a minute tick data (data at a macro-level) and makes the decision to buy, sell, or hold the asset.
2. Micro-agent: The micro-agent is given order book data (data at a micro-level) and makes the decision of where to place the order within the limit order book.

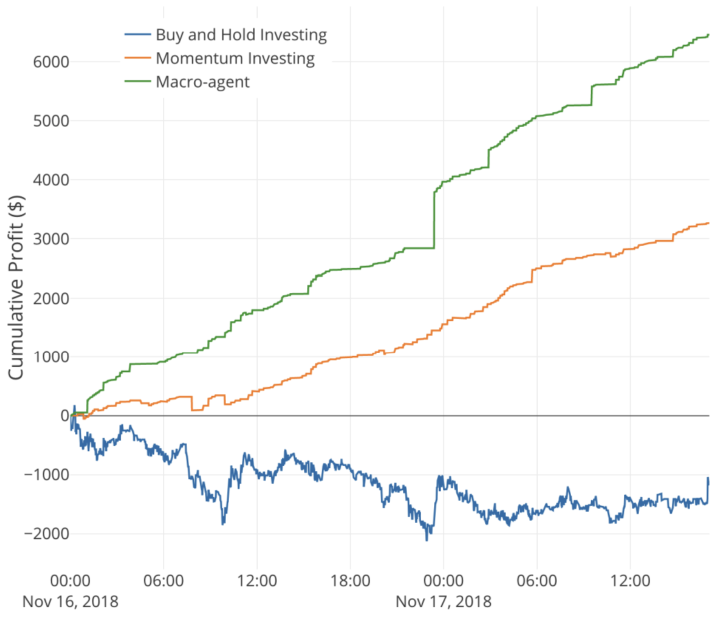

Figure 13: Performances of Various Investment Strategies Realized-PNL Graph(PNL:Profit and Loss)

References

[1] S. Kumar, J. Parker, and P. Naderian, *Adaptive transformers in RL*, CoRR, abs/2004.03761 (2020).

[2] Y. Patel, *Optimizing market making using multi-agent reinforcement learning*, 2018.

[3] T. Silver, K. R. Allen, J. Tenenbaum, and L. P. Kaelbling, *Residual policy learning*, CoRR, abs/1812.06298 (2018).

Appendices

A Example Codes for Imitation Learning

We firstly introduce the expert which utilize the baselines algorithms, such as TWAP or VWAP.

```
import gym
env = gym.make("Gym_trading-v1")
expert = TWAP()
```

Secondly, we apply the expert to sample some trajectories.

```
from imitation.data import rollout
from imitation.data.wrappers import RolloutInfoWrapper
from stable_baselines3.common.vec_env import DummyVecEnv
num_cpu = 8 #check it by nproc --all in bash
rollouts = rollout.rollout(
    expert,
    DummyVecEnv([lambda: RolloutInfoWrapper(gym.make("Gym_trading-v1"))] * num_cpu)
    ,
    rollout.make_sample_until(min_timesteps=None, min_episodes=60),
)
```

After we collected our transitions, then we set up our GAIL(Generative Adversarial Imitation Learning) algorithm. And the reward_net is actually the network of the discriminator.

```
from imitation.algorithms.adversarial.gail import GAIL
from imitation.rewards.reward_nets import BasicRewardNet
from imitation.util.networks import RunningNorm
from stable_baselines3 import PPO
from stable_baselines3.common.evaluation import evaluate_policy
from stable_baselines3.common.vec_env import DummyVecEnv, SubprocVecEnv
import gym

venv = DummyVecEnv([lambda: gym.make("Gym_trading-v1")] * num_cpu)
learner = PPO(
    env=venv,
    policy=MlpPolicy,
    batch_size=64,
    ent_coef=0.0,
    learning_rate=0.0003,
    n_epochs=10,
)
reward_net = BasicRewardNet(
    venv.observation_space, venv.action_space, normalize_input_layer=RunningNorm
)
gail_trainer = GAIL(
    demonstrations=rollouts,
    demo_batch_size=1024,
    gen_replay_buffer_capacity=2048,
    n_disc_updates_per_round=4,
    venv=venv,
    gen_algo=learner,
    reward_net=reward_net,
)

learner_rewards_before_training, _ = evaluate_policy(
    learner, venv, 100, return_episode_rewards=True
)
gail_trainer.train(int(1e8))  # Note: set to 300000 for better results
learner_rewards_after_training, _ = evaluate_policy(
    learner, venv, 100, return_episode_rewards=True
)
```

We can keep on training based on the PPO agent trained via GAIL.

```
learner.learn(total_timesteps=int(2e8), learning_rate = 0.0003, tb_log_name="
    first_run")
#a relatively bigger learning rate for faster converging in the first-stage
    training
learner.learn(total_timesteps=int(2e8), learning_rate = 0.000003, tb_log_name="
    second_run", reset_num_timesteps=False)
# turn down the learning rate after the first-stage training
learner.save("gym_trading-v1")
```

B Interface of the Gym_trading Package

The class **Flag** stores all the parameters for an environment.

```
class Flag():
    lobster_scaling = 10000 # Dollar price times 10000 (i.e., A stock price of $91.14 is g
    max_episode_steps= 2048 # max_episode_steps = 10240 # to test in 10 min, long horizon #
    max_action = 300
    max_quantity = 6000
    max_price = 31620700
    min_price = 31120200
    min_quantity = 0
    scaling = 30000000
    num2liquidate = 500
    cost_parameter = 5e-6 # from paper.p29 : https://epubs.siam.org/doi/epdf/10.1137/20M138
    skip = 1 # default = 1 from step No.n to step No.n+1


class Broker():
    @classmethod
    def _level_market_order_liquidating(cls, num, obs):
    @classmethod
    def pairs_market_order_liquidating(cls, num, obs):
```