Neural LOB Research Report

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

August 22, 2022

Table of contents

1 Right Deceleration

2 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.

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

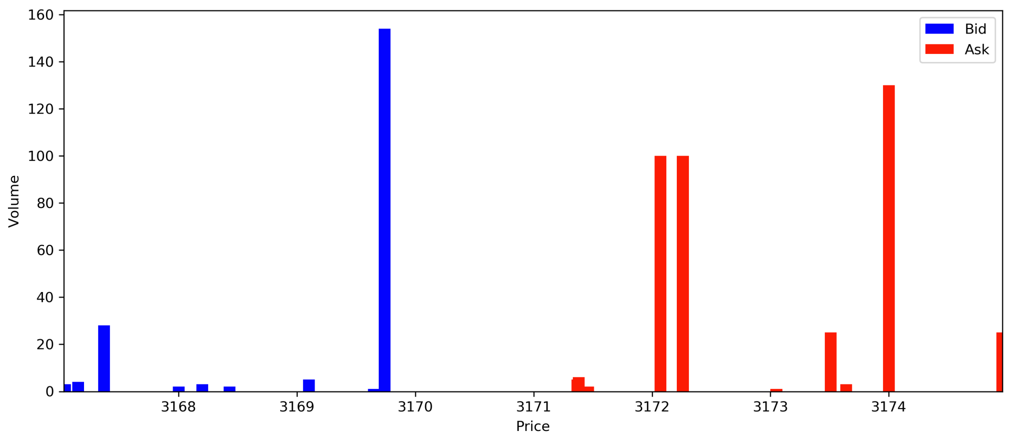


Figure 1: A snapshot of the LOB of AMZN with ten levels of bid/ask prices.

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.

3 Gym_trading

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

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 follows.
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(
```

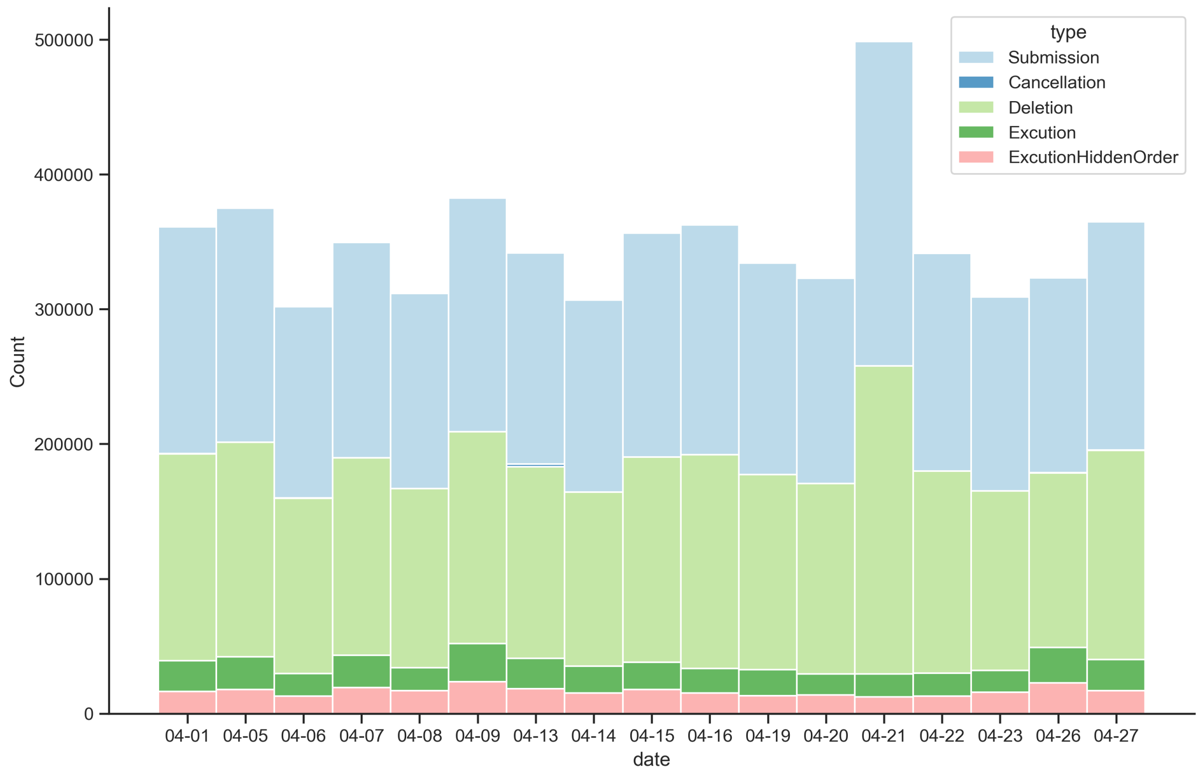
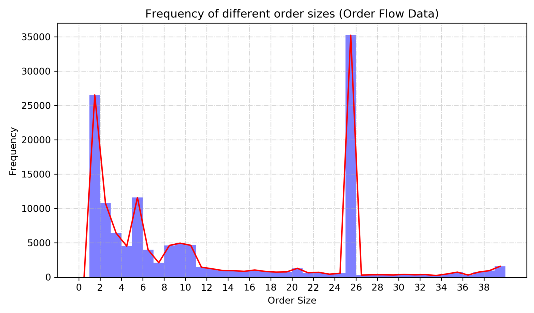


Figure 2: The Components of the Order Flows



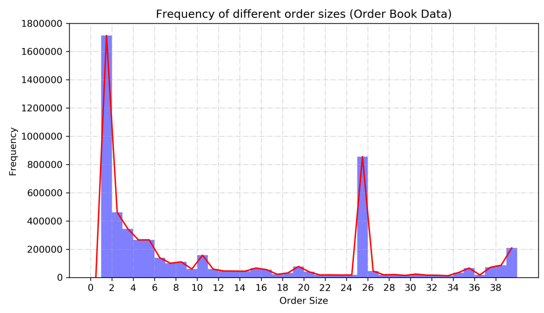


Figure 3: Order Size Frequency of the Order Flow   Figure 4: Order Size Frequency of the Limit Order Book

```
    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

```

```python
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.

```python
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")
```

4.1.2 Residual Policy Learning

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 timesteps 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 levarage the previous baselines.
The main idea of the algorithm is to design a neural network to learn the info that cannot be represented by the baselines algorithms.

4.2 Decision Transformer

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