RLQuant

Intro 2 AI Final Project Report

110652019 林楷傑
110705013 沈昱宏
110550041 蔡承翰
109652030 周子翔

Github: https://github.com/KJLdefeated/RL_for_Quatitative_Trading

Youtube : https://youtu.be/xPZTVYP1EEs

Introduction

Quantitative trading, also known as algorithmic trading, is gaining popularity in the financial industry.

Reinforcement learning (RL) techniques offer potential for developing adaptive trading strategies.

This project aims to evaluate the performance of RL algorithms in quantitative trading.

- REINFORCE with GAE
- Deep Q-Network (DQN)
- Double Deep Q-Network (DDQN)
- Trajectory Transformer

Related Works

There are many approach applying seqence to sequence ML algorithm to Quantitative market, but there is no one combine RL, seq2seq and Quatitative trading together.

We decide to apply these algorithm and evaluate how they performed. This is the report of our result and analysis why it perform good/bad.

Personae - RL & SL Methods and Envs For Quantitative Trading

Reinforcement Learning for Quantitative Trading

RLQuant

Environment / Platform : gym-anytrading

Base on this OpenAI Gym environment. Core part modified to suit our algorithm.

- **Action space** - Sell = 0 and Buy = 1. No action if action space same as previous.
- **Observation space**
  - Window size - number of previous days' data
  - Signal feature - opening price, closing price, highest price, lowest price.
- **Profit**

  Initial fund = 1, and everytime you sell, the fund becomes:

  $$fund(t) = fund(t-1) * (p_t / p_{buy})$$

  Final fund - 1 is our profit.
- **Trade fee**

  Due to high trading frequency, we disable the trade fee for simplicity in our environment.

Different Reward Definition

Reward

In various algorithms, we examined different approaches for calculating rewards. After testing, we identified two distinct reward calculation methods. One of these methods was selected for integration within the Reinforce algorithm, while the other was employed in DQN、DDQN、TT.

$$R1(t,a) = \begin{cases} p_t - p_{t+1}, & \text{if sell} \\ p_{t+1} - p_t, & \text{if buy} \end{cases}$$

$$R2(t,a) = \begin{cases} p_t - p_{buy}, & \text{if sell and long position} \\ 0, & \text{otherwise} \end{cases}$$

Dataset

Stock num : 2330 (TSMC)

Date : 2016/01/01 ~ 2022/12/31, one row per day

Source : web-crawled from 證券交易所 (in getStockData.py)

Training data : index 0~1000; Testing data : index 1000~1500

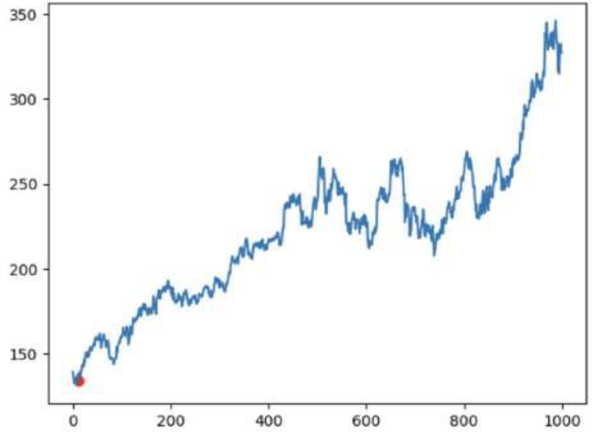
```python
def get_stock_month_data(year, month, stock_no):
    date = str(year+month+'01')
    print(date)
    html = requests.get('https://www.twse.com.tw/exchangeReport/STOCK_DAY?response=json&date=%s&stockNo=%s' % (date,stock_no), headers = headers) # disguised as browser
    content = json.loads(html.text)
    return content

def get_stock_by_list(year_list, month_list, stock_list):
    if len(year_list)==0 or len(month_list)==0 or len(stock_list)==0:
        return
    for stock_no in stock_list:
        content = get_stock_month_data(year_list[0], month_list[0], stock_no)
        first = True
        for year in year_list:
            for i in range(0,len(month_list)):
                if first:
                    first = False
                    continue
                time.sleep(3)                  # will be block if too fast
                content2 = get_stock_month_data(year, month_list[i], stock_no)
                if 'data' in content2:
                    content['data'].extend(content2['data'])
        df = pd.DataFrame(data=content['data'])
        df.head()
        df.to_csv('stock_data_'+stock_no+'_new.csv', index=False)
    return
```

Training Data & Testing Data

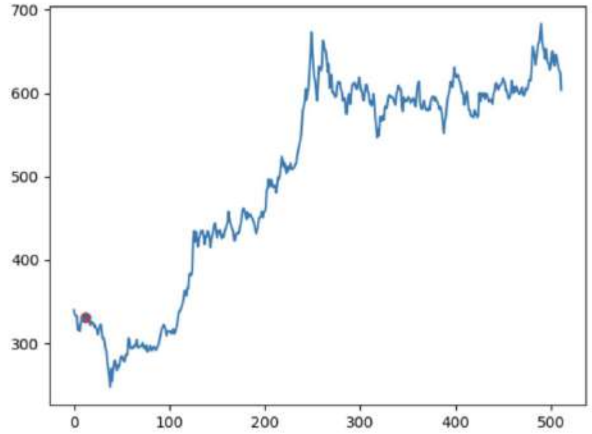
Training Data:

- Maximum Possible Profit: 251.61



Testing Data:

- Maximum Possible Profit: 31.38



Baseline - Moving average crossover method

Day trading method

Information used : last 200 day's Open, Close, High, Low price

Algorithm (in baseline.py) :

for each day :

calculate

1. the average price of last 50 days

2. the average price of last 200 days.

If 1 is greater than 2 -> indicates the stock is doing good recently -> buy

If 2 is greater than 1 -> indicates the stock is doing bad recently -> sell

Normalization

Reason for normalizing

1. Avoiding Scale Bias
2. Generalization

Method : calculate relative change for previous days

$$change_{relative} = (data_{now} - data_{previous}) / data_{now}$$

```python
for i in range(12):
    for j in range(4):
        tempstate1[i*4+j] = (state[44+j] - state[4*i+j])/state[44+j]
        # tempstate is fed into the NN
```

Main Approach

REINFORCE with GAE

The Reinforce algorithm includes 3 steps:

1. Collect Trajectories
2. Compute Advantages with GAE
3. Update Policy Network

   The objective is to increase the probabilities of actions that lead to higher rewards, weighted by the advantages computed with GAE. The policy gradient is calculated using the log probabilities of actions taken during the trajectories and the advantages.

$$\nabla_\theta V^{\pi_\theta}(\mu) = \mathbb{E}_{\tau \sim P_\mu^{\pi_\theta}}\left[\sum_{t=0}^{\infty} \gamma^t A^{\pi_\theta}(s_t, a_t) \nabla_\theta \log \pi_\theta(a_t \mid s_t)\right]$$

Generalized Advantage Estimation

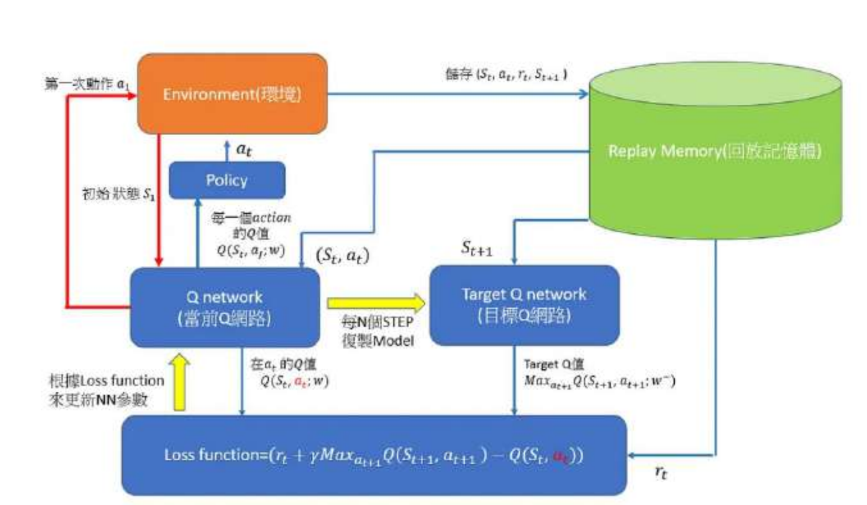
Generalized Advantage Estimation (GAE) estimates advantages by combines temporal difference (TD) errors over multiple time steps to capture the expected improvement of taking one action over another.

$$
\begin{aligned}
\hat{A}_t^{\mathrm{GAE}(\gamma,\lambda)} &:= (1-\lambda)\left(\hat{A}_t^{(1)} + \lambda \hat{A}_t^{(2)} + \lambda^2 \hat{A}_t^{(3)} + \dots\right) \\
&= (1-\lambda)\left(\delta_t^V + \lambda(\delta_t^V + \gamma\delta_{t+1}^V) + \lambda^2(\delta_t^V + \gamma\delta_{t+1}^V + \gamma^2\delta_{t+2}^V) + \dots\right) \\
&= (1-\lambda)\left(\delta_t^V(1+\lambda+\lambda^2+\dots) + \gamma\delta_{t+1}^V(\lambda+\lambda^2+\lambda^3+\dots)\right. \\
&\qquad \left. + \gamma^2\delta_{t+2}^V(\lambda^2+\lambda^3+\lambda^4+\dots) + \dots\right) \\
&= (1-\lambda)\left(\delta_t^V\left(\frac{1}{1-\lambda}\right) + \gamma\delta_{t+1}^V\left(\frac{\lambda}{1-\lambda}\right) + \gamma^2\delta_{t+2}^V\left(\frac{\lambda^2}{1-\lambda}\right) + \dots\right) \\
&= \sum_{l=0}^{\infty}(\gamma\lambda)^l \delta_{t+l}^V
\end{aligned}
$$

DQN

DQNs work by using a neural network to approximate the action-value function, which maps states of the environment to the expected return for each possible action.

The goal of the DQN is to learn the optimal policy, which is the action that will maximize the expected return for each state.




```
q_eval = self.evaluate_net(states).gather(1, actions)
q_next = self.target_net(next_states).detach() * (1 - done).unsqueeze(-1)
q_target = rewards.unsqueeze(-1) + self.gamma * q_next.max(1)[0].view(self.batch_size, 1)
```

Double DQN

DDQN is an extension of the DQN algorithm.

In DQN, the Q-values are often overestimated, and DDQN is used to address this issue.

Separate target network and decouple the action selection and value estimation steps to reduce the overestimation bias observed in DQN and leads to more accurate Q-value estimates.

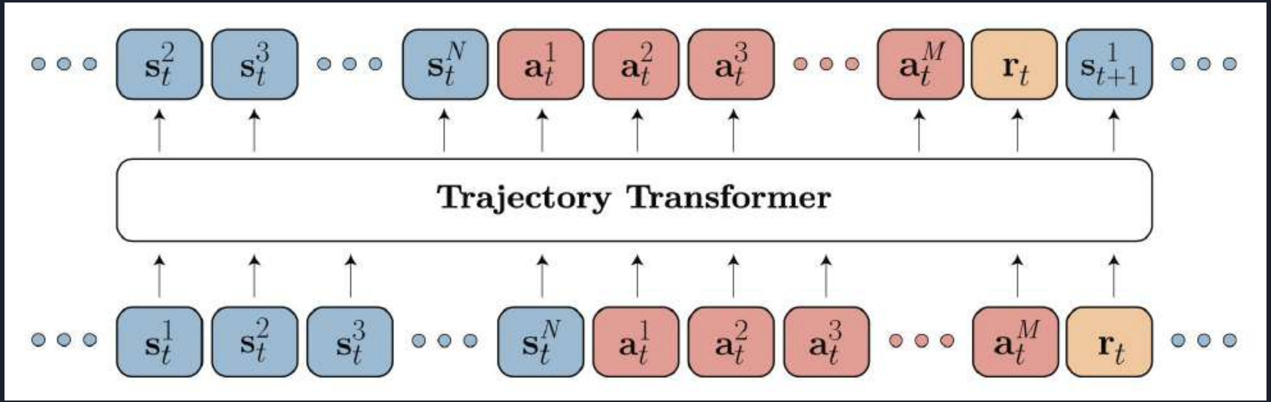
Result in improved performance and faster convergence.

$$Q_{evaluate}(s_t,a) = Reward_t + \gamma Q_{target}(s_{t+1}, \max_a(Q_{evaluate}(s_{t+1},a)))$$

```
q_values = torch.gather(self.evaluate_net(states), 1, actions)
next_actions = self.evaluate_net(next_states).argmax(dim=1, keepdim=True)
next_q_values = self.target_net(next_states).gather(1, next_actions).reshape(32)
target_q_values = (rewards + self.gamma * (1 - dones) * next_q_values).unsqueeze(1)
```

Trajectory Transformer (TT)

- Chat GPT ? GPT-2
- Modeling RL to sequence to sequence problem.
- Train on dataset of trajectraries. Offline Algorithm. Use DDQN to collect data trajectory.
- Solve RL reward distribution problem through the self-attention in the transformer.



Original Github Repo: https://github.com/jannerm/trajectory-transformer

Trajectory Transformer Training

- Training is performed with the standard teacher-forcing procedure used to train sequence models. Denoting the parameters of the Trajectory Transformer as θ and induced conditional probabilities as Pθ, the objective maximized during training is:

$$\mathcal{L}(\bar{\tau}) = \sum_{t=0}^{T-1} \left(\sum_{i=0}^{N-1} log P_\theta(\bar{s}_t^i | \bar{s}_t^{<i}, \bar{\tau}_{<t}) + \sum_{i=0}^{M-1} log P_\theta(\bar{a}_t^i | \bar{a}_t^{<i}, \bar{s}_t, \bar{\tau}_{<t}) + log P_\theta(\bar{r}_t | \bar{a}_t^{<i}, \bar{s}_t, \bar{\tau}_{<t}) \right)$$

Trajectory Transformer Planning

- Beam search is a search algorithm that can be used to find the most likely sequence of tokens given a probability distribution over sequences.

Algorithm 1 Beam search

Require Input sequence $\mathbf{x}$, vocabulary $\mathcal{V}$, sequence length T, beam width B

Initialize $Y_0 = \{\,(\,)\,\}$

for $t = 1, \ldots, T$ **do**

  $\mathcal{C}_t \leftarrow \{\mathbf{y}_{t-1} \circ y \mid \mathbf{y}_{t-1} \in Y_{t-1} \text{ and } y \in \mathcal{V}\}$ `// candidate single-token extensions`

  $Y_t \leftarrow \underset{Y \subseteq \mathcal{C}_t,\ |Y|=B}{\operatorname{argmax}} \log P_\theta(Y \mid \mathbf{x})$ `//` B `most likely sequences from candidates`

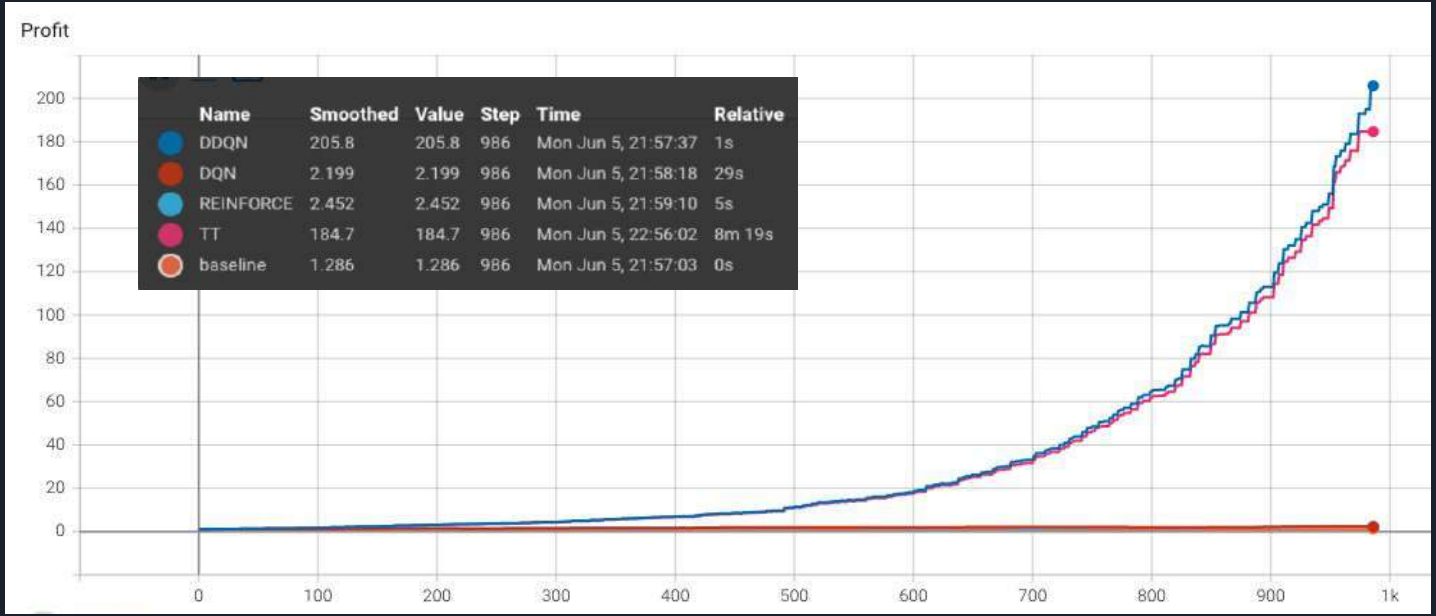
end for

Return $\underset{\mathbf{y} \in Y_T}{\operatorname{argmax}} \log P_\theta(\mathbf{y} \mid \mathbf{x})$

Experiment & Result

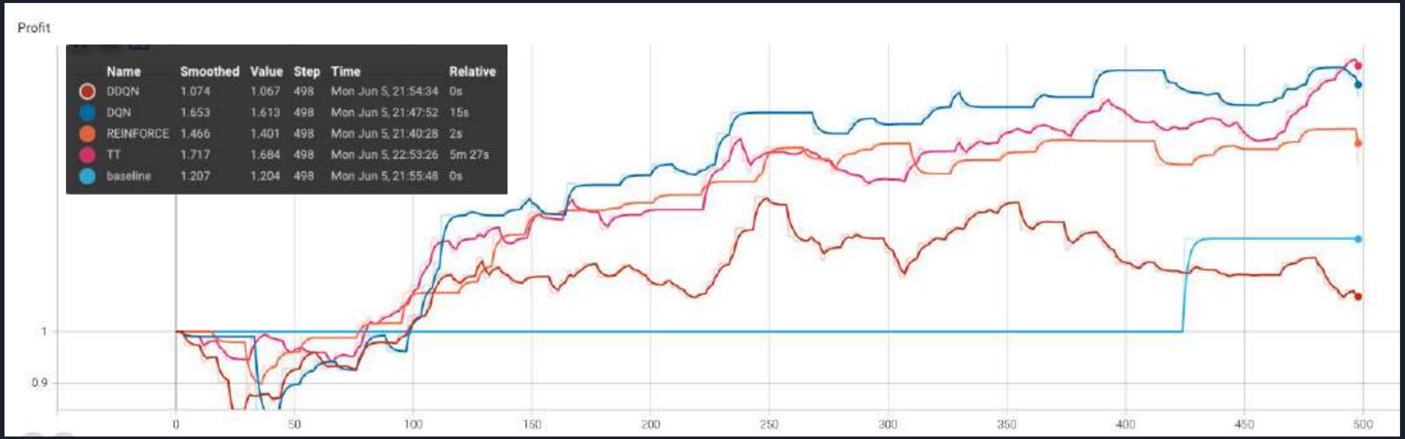
Result & Analysis
Total Profit of four models on training data

DDQN > TT > DQN > REINFORCE > baseline



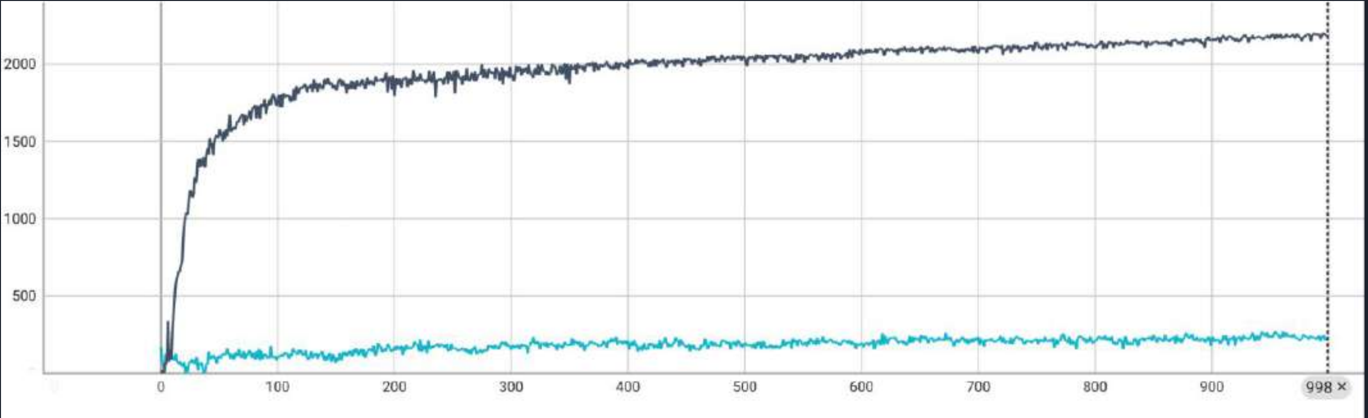
Result & Analysis
Total Profit of four models on testing data

TT > DQN > REINFORCE > baseline>DDQN



Different Reward Definition

- Use two different reward descriptive above to see which one is better.
- Train DDQN for 1000 epochs.
- Clearly first reward definition is better than the second one.



Different training epoch get different testing result – DDQN

- We can find that when the training epoch is large, although the result on training data is good, the testing become worse.
- This is an overfitting issue.

- Training bound

Training epoch	Total Reward	Total Profit
200	1931.5	109.31
400	2063.5	150.94
600	2162.5	193.63
800	2220.5	217.85
1000	2254.5	232.90

- Testing bound

Training epoch	Total Reward	Total Profit
200	102.0	1.63
400	-14.0	1.54
600	-26.0	1.39
800	-95.0	1.27
1000	-250.0	1.05

Trajectory Transformer(TT) perform on different training epoch

- Although TT does not perform well on training as DDQN, TT would not suffer from overfitting. TT can outperform DDQN on testing environment.

- Training bound

Training epoch	Total Reward	Total Profit
100	2202.5	212.68
200	2202.5	212.68
250	2202.5	212.68
300	2202.5	212.68
400	2202.5	212.68
450	2202.5	212.68
500	2202.5	212.68

- Testing bound

Training epoch	Total Reward	Total Profit
100	-175.0	1.15
200	-181.0	1.104
250	-230.0	1.014
300	-126.0	1.16
400	150.0	1.587
450	58.0	1.564
500	236.0	1.615

Conclusion and Future work

- In online algorithm, DDQN performs best in three RL algorithms.
- Trajectory Transformer perform best in four algorithm
- Add more algorithm in the future (like PPO, DDPG, Decision Transformer)
- Test on more stock data in the fure

Contributionn of each member

- 林楷傑 : Trajectory Transformer, Environment (30%)
- 沈昱宏 : DDQN, Data collection, Baseline (30%)
- 蔡承翰 : REINFORCE with GAE, experiment (20%)
- 周子翔 : DQN, report, README, reward (20%)
- Github contribution:

Reference

Playing Atari with Deep Reinforcement Learning

High-Dimensional Continuous Control Using Generalized Advantage Estimation

Deep Reinforcement Learning with Double Q-learning

Offline Reinforcement Learning as One Big Sequence Modeling Problem

Thank you