Work Trail Task at BlockHouse

Author: Guoyao Hao ghao004@ucr.edu

Abstract

In this project I trained an reinforcement learning agent for trading on the sample stock data. And I fine-tuned a pre-trained transformer as the feature extractor. After solving different difficulties during the project, I finally got some good result. The agent are unwilling to trade in a declining market. And the agent can make profitable buy/sell actions in a bullish market.

Data preprocessing

The original code already provides my a few technical indicators. I added a few more moving average indicators. Although those indicators couldn't provide additional information, they can benefit the training by smooth volatility.
I used more features as the input of the model such as the OHLC data, timestamp, volume……
Only knowing the market information is not enough for making actions. Current number of shares held and the average cost of those shares are also included

All data are normalized to make the training easier. For each observation, a window of history data are visible to the model. By introducing intervals between each sample point, the model can achieve a larger receptive field while utilizing a smaller amount of data. A window size of 64 and window interval 32 give us reception field of 2048 time steps. Limited by my hardware, it will take me forever if use a larger window size.

How to use transformer

I realized several differences between this problem and those problems that I familiar with.
Firstly, transformer is a supervised model. Ground truth labels are needed to train the model. However, the raw data is just market data, there's no data for how to trading.
Secondly, after buying an order, only after a while, we can know whether it's a good action or not.
Thirdly, transformer need the data samples to be independent, however, in this project, different time steps are not independent to each other. For example, we need to buy stock before selling it.

It's a typical problem for deep reinforcement learning. So I am going to use transformer with PPO.
The Stable Baselines3 (SB3) networks are separated into feature extractor and networks that maps the features to action/value. In this project, I use transformer as the feature extractor, since this structure can be shared by the action net and the value net.

The use of a pre-trained Transformer

The provided data is not enough for training a transformer from the very beginning, so using a pre-trained model and fine-tune on it is very important. And I used pre-trained transformer from this repo https://github.com/hemangjoshi37a/TrendMaster#
I choose this model for the following reasons
1. It has some prediction ability on stock market, the domain is closed to my project
2. The model is small enough to be handled by my Apple M1 and 8GB RAM.
3. It has a complete transformer structure

I added an input transformation layer before the pre-trained model, and removed the decoder part, since a encoder-only transformer structure fits my project well.

Ways to calculate reward

The reward consists of two part: the cost transactions and the profit/loss of holding stocks.

Strategy 1: When buying or selling, calculate the transaction cost, minus the transaction cost from the reward. And on each time step, calculate the profit/loss of the current share by comparing the current price and the price of the last time step.

This causes problems:
1. When holding a lot of stocks, the reward is mostly decided by the noisy market, instead of actions of the agent. It makes the value net difficult to learn.
2. The transaction cost is immediately reward, but the profit of holding stocks is always delayed reward. Because of the using of Gamma, the reward will be unfair for the profit.

Strategy 2: When buying, the reward is the negative of all the cost. When selling, the reward is the income. And when holding, the reward is 0.
The problem of strategy 2 is that it prevent us from buying stocks because buying gives us a big negative reward.

Strategy 3: When buying and selling, calculate the transaction cost. Additionally, when selling, use *(current_price - average_price_of_holding)* shared_sold* as the reward for holding stocks. It solves the problem of strategy2 and the model is willing to have more buying actions. It also has a problem, if current holding shares are losing money, it's reluctant to sell it.

Although feeling that the strategy 1 is the most accurate one among those strategies, I am not able to converge the training because of the noisy market. I was using the first strategy one in the beginning, and moved to the third version.

Parameter selection

This parameter depends a lot on the nature of the task, instead of the training itself.

It has been proved by many papers that the stock price movement in very short time period is pure random walk. So, for trading strategy we will look for trading on trends on longer period. The parameter gamma will affect the willing of holding shares. A smaller gamma encourages a fast profit. And a large gamma are more like like to hold shares longer. In the end, I choose gamma = 0.9999 or gamma=1

Experiment result and Conclusion

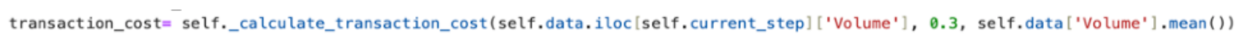
The result experiment shows that the the agent won't do any trade.
Explained variance is fluctuating between positive and negative values, implying the value net unable to predict well.

Several feature of the given data increases the difficulty of training a good agent.

1. The trend of the stock is mainly going down, the opportunity of making profit trading is very little.
2. There's a significant change in the environment between the beginning and the end

I also tried to reversed the given data, create a bullish market and train, then the agent is willing to buy a lot of stocks and hold. The result is saved in trades_ppo_reversed.csv

Potential mistakes I made
1. My transformer is too small to predict future market price.
2. The reward function is not well designed.
3. After calculating the transaction cost, I multiply it by the share sold/bought. I am not sure what is this cost, and whether I am doing it correctly.



```
transaction_cost= self._calculate_transaction_cost(self.data.iloc[self.current_step]['Volume'], 0.3, self.data['Volume'].mean())
```

More to do

1. Reduce the learning rate of the pre-trained transformer.
2. Use larger transformer and train on larger window size.
3. Maybe a better reward function can be developed.