AlphaTrader: DRL in Partially Observed Environment with Variable Action Space

Ziheng Chen (zc2068@nyu.edu)
Finance and Risk Engineering Department, New York University

Gordon Ritter (gordon.ritter@gmail.com)
Courant Institute of Mathematical Sciences, New York University

Abstract

Deep reinforcement learning algorithms have been shown to be effective in a variety of challenging domains, including GO, robot control and Computer games. However, DRL algorithms have rarely reached successful results in trading. One of the reasons is that DRL algorithms, especially Q-learning, typically assume a fully observed state and the process needs to be Markovian. However, the environment for trading is always only partially observed and violates the Markovian assumptions needed by lots of DRL algorithms. Hence, applying DRL in trading falls in the field of Partially Observed Markov Decision Process (POMDP).

Another essential problem is that we always have variable action spaces since we are restricted by capital and legal provisions. In this paper, we research on how to apply DRL algorithms in partially observed environment with a variable action space and apply these algorithms in a simulated trading environment to find out trading strategies given alphas.

The main contribution of this paper is that we created an algorithm, named AlphaTrader, a modified version of the Proximal Policy Gradient (Schulman, Wolski, Dhariwal, Radford, & Klimov, 2017), which can work efficiently in a simulated trading environment and find out the optimal trading strategy to maximize for the investor's utility given alphas.

Keywords

Partially Observed Markov Decision Process, Variable Action Space, Proximal Policy Optimization, Entropy, Alphas, Utility

Introduction

The dynamics of financial markets could be understood as a Hidden Markov Process which is not fully observable directly. The features coming from research on market micro-structure and financial economics which have predictive power or contain information describing the current market situations are called alphas in industry because they can give extra return if exploited with suitable trading strategies. The alphas can also be understood as the partially observations of the states hidden by some noise or short-term shocks. Hence, given alphas, designing a trading strategy falls into the field of Partially Observed Markov decision problem.

Model-free DRL could mainly be classified into two groups: Q-learning algorithms and Policy Gradient algorithms.Q-learning algorithms such as DQN generally assume a Markovian observation space. Policy gradient methods such as proximal policy optimization does not need to assume Markovian observations but are less sample efficient than Q-learning algorithms. Since simulating an environment and sampling from it is always quite cheap for trading, we could afford the price. Hence, Policy gradient methods are preferred in trading.

Reinforcement Learning

Reinforcement Learning algorithms were mainly developed to solve the optimal control problem aiming for choosing the optimal action a_t in state s_t, maximizing for the long time cumulative expected given reward:

$$E[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ... + \gamma^{T-t} r_T]$$

where r_t is the reward the agent get in time t, r_T is the terminal reward, and gamma is the discount rate. RL have already shown amazing abilities on solving these tasks including Go, robot control and computer games. Reinforcement Learning are very good at learning from trial and error when the environment is nearly the same. RL algorithms excel in solving problems where environment are stable and meaning of every element in state and reward is the same.

Basic Concepts

environment Environment is what reinforcement learning interact with and it tries to understand. Usually, we do not have directly access to the internal mechanism of the environment and we need to learn how the environment works by interacting, exploring. The environment has to be stable to let the reinforcement learning work. If the environment changes greatly, reinforcement learning algorithm could not work. Model-free RL algorithms generate policy without building a explicit model for the environment, while model-based RL try to build a model for the environment.

agent In reinforcement learning, agents performs action to interact with the environment, updating its policy, trying to maximize for the long-term accumulated discounted reward. Agents learns in the exploration period, and use its learnt policy in the exploitation period.

Q-learning

Among model-free deep reinforcement learning models, one main kind of algorithm is Q-learning, a value function fitting algorithm. Q-learning always does exploration through ε-greedy algorithms and Q-learning mainly learn the Q-value of state and action pair by bootstrapping:

$$Q^*(s,a) = r(s,a) + \gamma max_{a'} Q(s',a')$$

s' represents the state agent arrives after taking action a at current state t, and a' represents the action agent takes at state s'. When the environment is Markov, Q-learning could be very powerful in giving the optimal solution, like in Gordon's paper (Ritter, 2017). However, when the environment is partially observed, the *max* operator could cause severer problem and let the Q-learning fail completely. In a partially observed environment, we can only get observation o_t instead of real state s_t. If we just assume that observations are true states, the original equations becomes

$$Q^*(o,a) = r(o,a) + \gamma max_{a'} Q(o',a')$$

which let the $Q(s,a)$ can not be calculated directly. Just assuming $Q(s,a)$ is $Q(o,a)$ could lead to great bias and the max operator makes the problem even worse. It simply chooses the a' with the largest estimated value, which in many cases the largest value is caused by the largest estimate error. By bootstrapping value with very large error to another value, the $Q(s,a)$ values finally goes far larger than the real values and could hardly be used. Here we reached to the conclusion that Q-learning algorithms are not suitable to work in a partially observed environment.

Policy Gradient

Policy gradient performs gradient descent on the true objective:

$$\theta^* = agrmax_{\theta} E_{\theta}[\sum_t r(s_t, a_t)]$$

Policy gradient methods do not assume full observability. In essence, the learning of policy gradient algorithms could be understood as increasing the probability of choosing an action if the output is good, otherwise decrease the probability. TRPO and PPO further improves the efficiency and performance by changing the learning process from parameter space to policy space.

RL Environment and Agent Setting

Simulated Trading Environment and Alphas

We simulated assets with geometric Brownian motion:

$$dS_t = \mu S_t dt + \sigma S_t dW_t$$

The prices are given every minute. We set μ to zero, so there is no drift and the agent can only try to learn how to make profit out of volatility.

The allowable position is [-100, 100], so the action space is variable according to current position. No trading fee or spread are considered in this research.

In industry, so called alphas are standardized signals, having mean of 0 and variance of 1, with a certain correlation coefficient with future return in a given time horizon. The correlation coefficient is called information coefficient, or IC in short:

$$IC_i = corr(\alpha_{i,t}, \frac{S_{t+i}}{S_t} - 1)$$

In this paper, we have four alphas, each having predictive power in a 1-min, 5-min, 10-min, and 30-minute time horizon. The signals having a constant correlation with future return, and we change the IC multiple times to see its effect on agent learning.

State

In our experiment, state is consist of useful alphas and current position. Considering 5-min alphas at time t, it could at most be useful at time t+4. We can also do this for all other alphas. So, finally s_t should contain alphas:

$$[\alpha_{1,t}, \alpha_{5,t\cdot}, \alpha_{5,t-4}, \alpha_{10,t\cdot}, \alpha_{10,t-9}, \alpha_{30,t\cdot}, \alpha_{30,t-29}]$$

If these alphas earlier than t do not exist, alphas are set as zero. and $position_t$ is also needed in states to let the agent know the current position. Hence, in our experiment, there are 47 elements in each state, 46 alphas and 1 current position.

Action

The trade a_t the agent take at time t. The minimum unit of a trade is a position of 5, so there are 41 possible actions at most:

$$[-100, -95, ..., -5, 0, 5, .., 95, 100]$$

These actions are indexed from 0 to 40.

Reward

The reward function is the risk-adjusted return:

$$r_t = PnL_t - \kappa PnL_t^2$$

Here, κ serves as the penalty coefficient for variance. According to Gordon's paper (Ritter, 2017), when the PnL_t is small, PnL_t^2 could serve as a simple abut effective institute for variance.By setting the reward function this way, we are also doing a mean-variance style optimization. According to utility theory, mainly developed by Pratt (Pratt, 1964) and Arrow (Arrow, 1971), an investor's utility could mostly be satisfied by doing such an optimization.

Proximal Policy Optimization

Here, we give a quick refresh about PPO: PPO is a simpler and more effective version of TRPO (Schulman, Levine, Moritz, Jordan, & Abbeelv, 2015). It alternates between sampling data through interaction with the environment, and optimizing a "surrogate" objective function using stochastic gradient ascent (Schulman et al., 2017). TRPO's surrogate objective is:

$$L(\theta) = E_t[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}A_t] = E^t[r_t(\theta)A_t)]$$

PPO applies a clipping function to ensure that the new policy π_θ does not deviate too much from the old policy $\pi_{\theta_{old}}$, and the main objective PPO needs to optimize is:

$$L^{clip}(\theta) = E_t[min(r_t(\theta)A_t, clip(r_t(\theta), 1-\varepsilon, 1+\varepsilon)A_t)]$$

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}A_t$$

The clipping function *clip* modifies the surrogate objective by clipping the probability ratio. If the ratio is larger than $1+\varepsilon$, it becomes $1+\varepsilon$ after clipping. If the ratio is smaller than $1-\varepsilon$, it becomes $1-\varepsilon$ after clipping.

Key Components of AlphaTrader

We use the Proximal Policy Optimization (Schulman et al., 2017) as our framework for AlphaTrader and there are four key modifications we made in order to let AlphaTrader to work in this partially observed environment with variable action space:

1. Advantage Function Estimation: Monte Carlo Method with Baselines

PPO belongs to the family of actor-critic algorithms, but AlphaTrader gives up the normal actor-critic style of estimating advantage function:

$$A(s_t, a_t) = r_t + \gamma V_{critic}(s_{t+1}) - V_{critic}(s_t)$$

and chose to use Monte Carlo method:

$$A(s_t, a_t) = r_t + \gamma r_{t+1} + ... + \gamma^{T-t} r_T - V_{critic}(s_t)$$

in exchange for less bias caused by the inaccuracies of the critic. In a partially observed environment, the estimates of critic could hardly be very accurate and could possibly harm the performance of the agent if using the original formula to estimate state value. By using a Monte Carlo method with a baseline could lead to a better performance under partially observed environment.

2. Mapping Illegal Actions into Legal Action Space

Since there are many illegal actions could be chose, what I need to do is to map them back the legal actions with a constant function. This does not harm the decision process, since the agent does not need to do what it is exactly doing. For example, the agent may choose an illegal action 40 and this illegal being mapped into legal action 20, and it receives the reward exactly the same of choosing action 20. As long as we keep the mapping constant, this could solve the problem of variable action space. In our experiment, we constantly map the possible illegal action to the nearest legal action.

3. Choice of Gamma

In a highly stochastic environment like this, the transitions between states cannot be predicted without having signals for that, so we are forced to focus on short-term profit that is predictable. Aiming for long-term profit cannot work and will only lead to the failure of learning. According to rule of thumb, the number of the following states we take into consideration could be estimated by:

$$num = \frac{1}{1-\gamma}$$

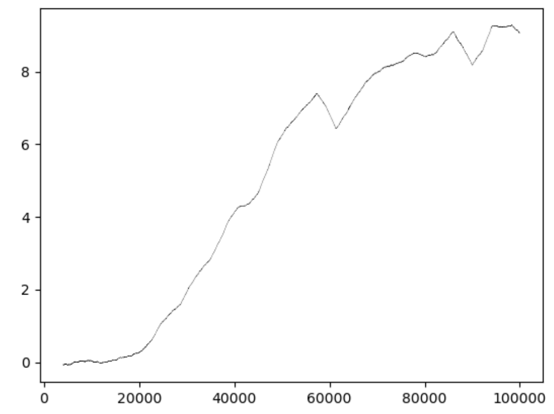
Figure 1: PnL under Strong Alphas

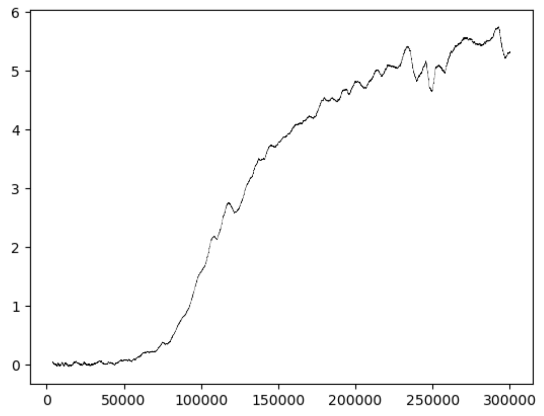
Figure 2: PnL under Real-world Strength Alphas

Since we are having 1,5,10,30 minutes signals, an ideal gamma could be between 0.7 and 0.9. We chose 0.8 in our experiment.

4. Entropy Multiplier Decaying

We also find out that in a highly stochastic environment, the corresponding optimal policy should also be stochastic instead of deterministic, so we set the minimum entropy multiplier higher than zero to get a more roust policy.

Empirical Results

Strong Alphas

We construct an environment equipped with strong alphas: the IC for all alphas are set as 0.5. In real world, this could never happen, but this could serve as a good start point. We set κ, the penalty parameter for variance as 0.5. We can see from figure 1 that AlphaTrader keeps improving its performance during the 100,000 samples training process. At the same time, the policy entropy of the policy drops (figure 3). When the training stops, the entropy is still larger than zero, which means the policy is stochastic, rather than deterministic. We can also see from figure 5 that the loss of the critic is rather low and stable during the whole training process.

Under strong alphas, DRL can reach great result, so for the next step, we would like to try it using the real-world alpha strength.

Real-world Strength Signals

In real-world, world-class minute-level alphas should have IC around 0.2 (Grinold & Kahn, 2001). Hence, we give 1-min alpha IC 0.3, 5-min and 10-min alphas IC 0.2, and 0.1 for 30-min alpha. We

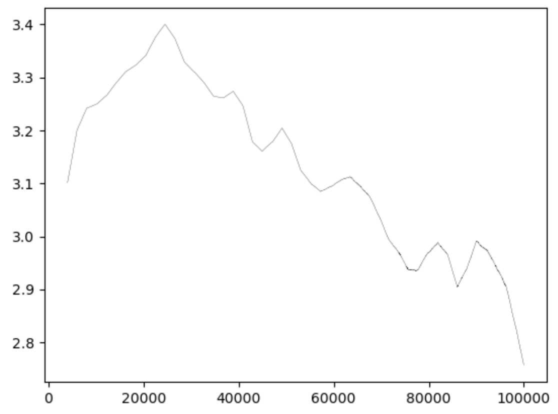
Figure 3: Entropy under Strong Alphas

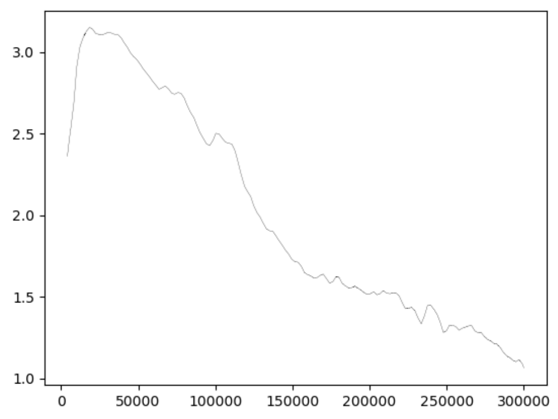
Figure 4: Entropy under Real-world Strength Alphas

also set κ, the penalty parameter for variance as 0.5. We can see from figure 2, that this time agent also reached great result although more samples are needed. The entropy of the policy is also larger than zero. Looking at the loss of the critic, we can see that the loss becomes a little volatile but does not harm the performance too much.

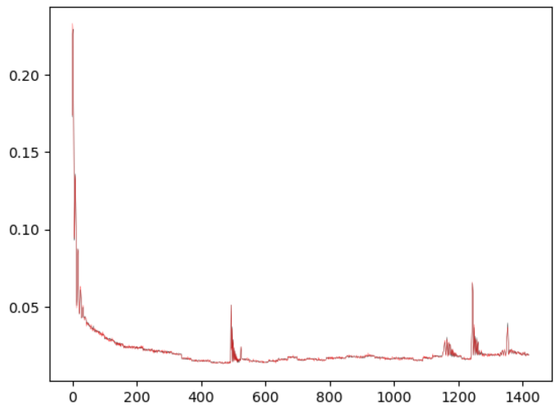

Figure 5: Critic Loss under Strong Alphas

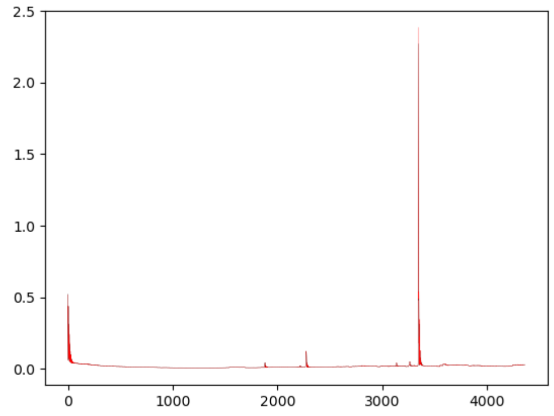

Figure 6: Entropy under Real-world Strength Alphas

Results

In this paper, we created AlphaTrader, a modified version of the PPO algorithm, and then we could see from the empirical results that AlphaTrader works very effectively and provides satisfactory performance given real-world strength alphas. We also believe that another potential of using this kind of framework is that we could skip the procedure of fitting different alphas into one final alpha signal per time horizon, both saving effort and keep as much information as possible, and put the diferent alphas into state directly instead. We believe that AlphaTrader have huge potential in time-series trading and could possibly lead the next revolution of algorithm trading.

References

Arrow, K. J. (1971). *Essays in the theory of risk-bearing*. Wiley.

Grinold, R. C., & Kahn, R. N. (2001). *Active portfolio mangement, a quantitative approach for providing superior returns and controlling risk*. McGraw-Hill.

Pratt, J. W. (1964). Risk aversion in the small and in the large. *Econometrica: Journal of the Econometric Society*, 122-136.

Ritter, G. (2017). Machine learning for trading. *Risk*, *30*, 84-89.

Schulman, J., Levine, S., Moritz, P., Jordan, M. I., & Abbeelv, P. (2015). Trust region policy optimization. *arXiv:1502.05477*.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv:1707.06347*.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT Press.