Actor-Critic-Method-for-Financial-Trading

Implementation of the DDPG algorithm for Automatic Finance Trading

In this project, I implemented the DDPG algorithm to solve the optimization problem of large portfolio transactions. This is an example of how Deep Reinforcement Learning can be used to solve real-world problems by simulating the problem in the form of an environment. Just like the various Deep RL algorithms, for example, DQN, DDPG and Multi-Agent DDPG algorithms are used to solve tasks related to robotics, gaming or even Autonomous Car Driving, these algorithms can be used to solve many supervised learning problems by properly formulating the tasks into an environment.

The 'environment' is basically a python class having methods and attributes related to the task. It generates 'States' for an Agent. The agent is a class that follows one of the above algorithms. The agent then generates 'Actions' and gets rewarded based on that action. The important thing is to define carefully the 'State', 'Action' and 'Reward' for the environment based on the problem.

Simulation Environment

The simulation environment generates random stock prices based on the Geometric Brownian Motion. The task is to determine the optimal number of shares to sell at a time in order to minimize expected shortfall for the trader.

In order to solve this problem through reinforcement learning, we need to restate the optimal liquidation problem in terms of States, Actions, and Rewards.

State

In order to get an optimal trading weights, our state vector must tell us how many shares are remaining to be sold and how much time is remaining to sell them. By showing the number of trades remaining, we will know about the time remaining to sell the remaining shares. We also need to know what the stock price trend was in the recent past to determine current strategy. Hence, we will also include log returns of past five trades.

The state vector looks as follows -

$$[r_{k-5}, r_{k-4}, r_{k-3}, r_{k-2}, r_{k-1}, r_k, m_k, i_k]$$

where

- $r_{(t-i)}$ = normalized log return at time t-i.
- M_t = number of trades remaining at time t, normalized by total number of trades.
- I_t = number of shares remaining to sell at time t, normalized by total number of shares.

Please note that more stock related parameters can be added to the state like leading or lagging financial indicators.

Action

By solving this environment, we are trying to determine the optimal number of shares to sell at a time. Therefore, for a given state the action should be number of shares to sell or the percentage of shares to sell. The actor networks will take as input the state vector and output a real number between 0 and 1. This can then be multiplied to the total number of shares remaining to get the number of shares to sell at time t.

Reward

Determining how to rewards an agent, is the trickiest part when formulating a task for Reinforcement Learning. The basic logic is to provide a reward that will increase as the agent does better. There are many ways to do this, but we need to find the simplest and optimal way where the utility of the agent is improving. Here we use the Almgren and Chriss model. We calculate the utility as mentioned in the Almgren and Chriss (AC) model at each time step. The reward at a time step is then the difference in the utility from the next time step. Hence the agent will choose an action in order to improve future utilities.

The utility function given by the AC model is,

$$U(x) = E(x) + \lambda V(x)$$

The details of the Almgren and Chriss model can be found here.

The reward at time t is,

$$Rt = Ut - Ut+1 \; / \; Ut$$

The reward has been normalized to make the convergence easier.

Let's look at the default parameters of our simulation environment. We have set the initial stock price to be S_0 = 50 and the total number of shares to sell to one million. This gives an initial portfolio value of $50 Million dollars. The trader's risk aversion to λ=10^{-6}

The stock price will have 12% annual volatility, a bid-ask spread of 1/8 and an average daily trading volume of 5 million shares. Assuming there are 250 trading days in a year, this gives a daily volatility in stock price of $0.12/250 = 0.8\%$. We will use a liquidation time of T = 60 days and we will set the number of trades N = 60. More details on the default parameters can be found in the paper and the module syntheticChrissAlmgren.py.