Gordon Ritter RL for Trading Implementation

-Karamjeet Singh

Abstract:

This project applies reinforcement learning (RL) on an OU process generated trading environment, under almost realistic market conditions, including trading costs. Our motive is to show that multiperiod portfolio choice can be solved by reinforcement learning methods. In other words, we try to show that machines can learn to trade and handle risk-averse expected utility maximization using RL. We provide proof of concept in a controlled numerical simulation in a detailed example, that permits an approximate arbitrage, and we verify that the Q-learning agent finds and exploits this arbitrage.

Methodology:

The implementation centers on a Q-learning algorithm, which updates the action-value function iteratively. Key components of the methodology include:

1. Generation of Trading environment using Ornstein-Uhlenbeck process:

We have assumed a tradable security with a strictly positive price p_t , and an equilibrium price p_e such that it follows: $\boldsymbol{x_t = \log(p_t/p_e)}$ with dynamics $dx_t = -\lambda x_t + \sigma\xi_t$, where $\xi_t \sim N(0,1)$ and p_t tends to revert to its long-run equilibrium with mean reversion rate λ. We have taken parameters to $\lambda = \log(2)/H$, where $H = 5$ is the half-life, $\sigma = 0.1$, and $p_e = 50$

To simulate the trading environment, we generate two price lists:

1. A training dataset with 10^7 steps.
2. An evaluation dataset with 5000 steps.

Both datasets are generated with a tick size of 0.1, and prices are restricted to the range [0.1,100.0]

2. Action space and State spaces:

State space consists of current Holdings and Price of the security, So we have taken Holding Space and Price Space as follows

Holding Space

we define the possible values of holdings nt to be bounded by M round lots, where $M = 10$. The set of holdings is expressed as: $H = \{-M, -(M-1),\ldots, -1,0,1,\ldots,M\} \times$ LotSize, where Lotsize = 100
For M = 10, this yields 21 possible holdings: {-1000, -900, …., 0, …, 900, 1000}

Price Space

To maintain practicality, the boundaries of the price space are chosen such that the sample paths of the process exit the space with negligible probability. For our parameters, the space of possible prices is given us: $P = \{p \in \mathrm{R} : p = \text{tick\_size} \times k,\ k \in \{1,2,\ldots., 1000\}\}$, corresponding to the range: $P = [0.1.\ 100.0]$

Action Space

In this setup, the **action space** represents the allowable trade sizes in a single interval. We define the trade size to be bounded by K round lots, where each "round lot" is 100 shares (LotSize = 100). The action space is expressed as: $A=\{-K,-(K-1),\ldots,-1,0,1,\ldots,K\}\times \text{LotSize}$.

For K=5 , A = {-500, -400, -300, -200, -100, 0, 100, 200, 300, 400, 500}.
Each action corresponds to buying or selling a specific number of shares in multiples of the round lot size, or taking no action (0).

3. Action Selection Process

The agent chooses an action $a_t \in A$ using an ϵ-greedy policy, which balances exploration (trying new actions) and exploitation (choosing the best-known action). This process ensures that, as the number of steps increases, every action is sampled infinitely often, leading to convergence to the optimal policy.

Action Selection Logic

State Identification:
The agent observes the current state $s_t = (n_t, p_t)$, where: n_t : Current holdings, p_t : Current price of security.

Random Exploration vs. Greedy Exploitation:

- With probability ϵ, the agent explores by randomly selecting an action $a_t \in A$ from the action space.
- With probability $1 - \epsilon$, the agent exploits its current knowledge by selecting the action a_t that maximizes the estimated value function $q(s_t,a)$: $a_t=\text{argmax}_{a \in A}\, q(s_t,a)$
-Handling Multiple Optimal Actions: If there are multiple actions that maximize $q(s_t,a)$, the agent randomly selects one among these optimal actions to ensure variability.

4. **Reward Function and Calculation Process**

The reward function evaluates the agent's trading performance by accounting for portfolio value changes and penalizing excessive risk. It is computed as: $R_t = \delta v_t - (\kappa/2)\,(\delta v_t)^2$

Key Variables:

• δv_t: Change in portfolio value, calculated as: $\delta v_t = n_t \cdot (p_{next} - p_{current}) - total\_cost(\delta n)$
where: - n_t : Holdings at time t, p_current: Current price , p_next: next price of the security and δn: Change in holdings (action taken by the agent).

• κ: Risk-aversion coefficient, controlling the penalty for variance in rewards.
Compute δv_t using the portfolio change and trading costs, and finally evaluate the reward using the reward function formula.

5. Q-Learning Algorithm:

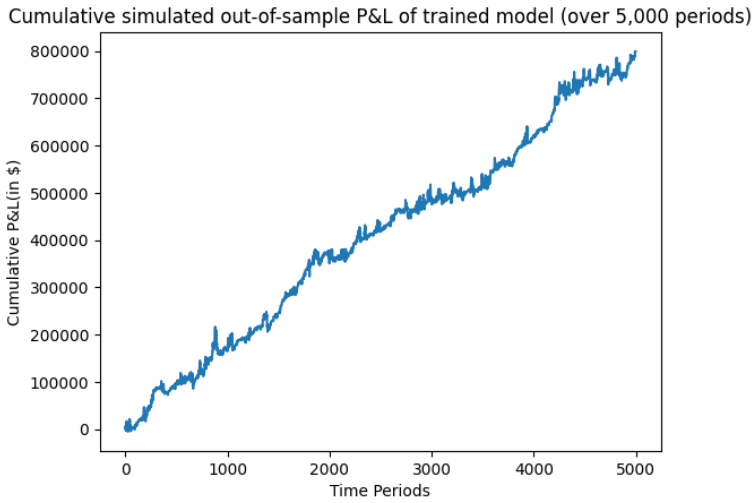
We have used the Q-learning algorithm to learn the optimal policy by updating the action-value function Q(s, a) iteratively. The update rule is given as: $Q(s, a) \leftarrow Q(s, a) + \text{alpha} \cdot [R + \text{gamma}.\max_a Q(s', a') - Q(s, a)]$.

Steps in Q-Learning:

1. Initialization:
- Initialize Q(s, a) to zeros for all states s and actions a.
- Set learning rate alpha, discount factor gamma, epsilon, and risk-aversion coefficient kappa.
2. Iterative Updates:
- Observe the current state $s_t = (n_t, p_t)$.
- Choose an action at using the epsilon-greedy policy.
- Execute at and observe the next state $s_{(t+1)} = (n_{(t+1)}, p_{(t+1)})$ and reward Rt.
- Update the Q-value using the given update rule.
3. Convergence:
-Repeat the updates until the Q-values converge or a predefined number of training steps are completed.

Training Parameters: alpha = 0.001, gamma = 0.999, kappa = 10^(-4), epsilon = 0.1.
- Training steps: 10^7 steps.
- Evaluation steps: 5000 steps.

Results:



Conclusion:

The excellent out of sample performance is observed with cumulative PnL of over 800,000 on 5,000 evaluation steps. This project successfully demonstrates the application of reinforcement learning (RL) in a controlled trading environment. The Q-learning algorithm successfully learns an optimal trading policy, balancing portfolio returns and risk under realistic market conditions. The results validate the potential of RL methods for solving multiperiod portfolio optimization and utility maximization problems, providing a foundation for further exploration in real-world trading scenarios.