Project 2 - Reinforcement Learning

B(E)4M36SMU

Monday 18th March, 2024

In the second homework assignment, we will apply reinforcement learning techniques to Bitcoin trading. Your reinforcement learning agent will be given a set of training and testing data and will try to learn decisions so that it generates a profit. Again, we will use the *Gymnasium* library [1] to test our code, this time with the environment taken from the `gym-trading-env` library [2].

The data contain 1-hour windows with information about open, high, low, and close prices. Those numbers mean the price at the start of the window, the highest price over the window span, the lowest price, and the price at the end of the window. Your task will be then to calculate the position (i.e., the policy) that should be held in Bitcoin to maximize your profit.

1 Warning — Comparison to Real-world

Important: Despite one can find many papers about applying reinforcement learning techniques to cryptocurrency trading, for example, in [3], it is not a good idea to apply such a simple algorithm we will develop in a real-world. It might generate a profit for a while, but it is very likely to generate a loss as well. The real world does not live strictly in the MDP setting; it is slowly developing, and such a *concept drift* requires continuous adjustment of the algorithm decisions. At some points, events that radically change the behavior of the environment occur, and algorithms that worked before suddenly stop working - such events are called *black swans* [4] with examples as the Great Depression in 1930, or the *dotcom bubble crash* at the beginning of this century. One of the reasons for the change in the principles of trading can be seen in the fact that if one thing is profitable, many people try to copy it, thus making it less profitable for others, and at one point, the environment ends up in a new equilibrium [5]. To sum it up, in reinforcement learning language - the environment is not *stationary.*

Other reasons why the direct application of such a simple code is not a good idea can be seen in some simplifications we have made. For example, the official currency being x does not mean you can trade the currency for x. If you are a large investor, there might simply be not enough people willing to buy or sell enough of the currency. We also expected that the environment is only a probability distribution, but in the real world, it is formed by many agents trying to maximize their gain. For all those reasons, please do not try to use this toy code in the real world. No warranties about its outcomes can be given.

2 Implementation

In the downloaded archive, you can find the `main.py` file that constructs the environment for you and allows you to train and test the provided data. Change this class to your own will, but keep in mind that all changes made to this class will not be reflected in BRUTE.

The other file, `tradingagent.py`, contains a draft of the class to implement. There, you might find several methods you will need to change. Change some or all of them; however, keep the method signatures; otherwise, the automatic evaluation will not work. The contents of the methods are taken from the `gym-trading-env` documentation [2]. Changing them is highly recommended.

- Method `reward_function` can be modified to define reward. Currently, it calculates the logarithm of the percentual gain of your method, but there are definitely better ones. See [6] for more details.
- Method `make_features` constructs features. We will use only static features. In this method, you will get a data frame with open, close, high, and low valuations, and your goal will be to create numeric features that will be good for learning. For example, Bollinger bonds or moving averages are good examples of such features. Create new columns with names in the form of `feature_xy` to create a feature that will be returned by the `step` method. See [7] to find documentation.

  Important: You are given the training and testing data in the data frame here. Please make sure that you do not create any look-ahead feature. To calculate the feature at time t, use only values from time $\leq t$. I will briefly look into your code to ensure that no such feature exists; creating a look-ahead feature might result in 0 points from the homework.
- The positions available to the agent can be customized in `get_position_list` method. In this method, you will specify the set of actions available to the agent; the values between 0 and 1 mean split of the value into USD and Bitcoin, and values outside this interval represent borrowing either currency. There is a small cost for each exchange, as well as for a loan. More details can be found at [8].
- Method `get_test_position` will be your policy in test.
- Method `train` will be used for training. You can run one or multiple epochs; each will be on the same 40,000 samples. You can use any reinforcement learning algorithm. This time, I don't care whether you follow their assumptions as GLIE or Robbins-Munro theorem; you only need to achieve the performance of 10 % on test data and 50 % on validation data. Do not hardcode any strategy; use machine learning (only you can encode the inputs of the strategy into features).

3 Problem Specification

Implement any reinforcement learning algorithm of your choice in `tradingagent.py` file and reach 10 % on test data and 50 % on validation data (both for five points). Do not hardcode any strategy, and while constructing the features, do not use any look-ahead features.

4 Submission and Evaluation

- All students must work individually.
- Upload the results to `https://cw.felk.cvut.cz/brute/`.
- Strict deadline is Thursday 11th April, 2024 11:59 pm. I cannot guarantee answers to your queries placed three days before (or any time after) the deadline.
- A penalty for late submission is -1 points for each day of delay.
- Submit `tradingagent.py`.
- The automatic evaluation runs Python 3.8, and PyTorch 1.6, together with `gymnasium pandas gym-trading-env` libraries and their dependencies.

- The project is worth 10 points in total; five points are given for gain more than 10,% on the training dataset, five points for 50,% gain on the validation dataset.
- Do not use any look ahead features, do not hardcode any strategy, or use any reinforcement learning algorithm.
- Training time is set to 5 minutes; if you need a raise, email me. Similarly, if you miss a library that you think should be available to all submissions.
- Should you have any questions, or you find a bug in the code or project specification, feel free to email me and/or ask for a consultation.

References

[1] `https://gymnasium.farama.org/index.html`

[2] `https://gym-trading-env.readthedocs.io/en/latest/`

[3] Otabek, S., Choi, J. Multi-level deep Q-networks for Bitcoin trading strategies. *Sci Rep* 14, 771 (2024). `https://doi.org/10.1038/s41598-024-51408-w`

[4] `https://en.wikipedia.org/wiki/Black_swan_theory`

[5] James Owen Weatherall. he Physics of Wall Street: A Brief History of Predicting the Unpredictable. (2013)

[6] `https://gym-trading-env.readthedocs.io/en/latest/customization.html#custom-reward-function`

[7] `https://gym-trading-env.readthedocs.io/en/latest/features.html`

[8] `https://gym-trading-env.readthedocs.io/en/latest/environment_desc.html#action-space`