SCIE 4500 Final Report: Policy Gradient Trading Algorithm by Maximizing Sharpe Ratio

Student: Wang, Xinyi
Supervisor: Prof. Yao, Yuan

September 29, 2019

Abstract

This project is the second half of my capstone project. In this project, I designed a trading algorithm using policy gradient to maximize the profit while incorporating the risk factor by directly maximizing the Sharpe ratio over a fixed period of time. Then I conduct experiments on a Bitcoin dataset to compare its performance with a Q learning algorithm.

1 Introduction

Reinforcement learning is gaining its popularity in the recent days, as surprisingly good results has been produced on the tasks like Go game and StarCraft. It also gets its application on training adaptive robots for harsh environments. However, in these cases the environments are well understood as they can be described with the gaming setup or physics rules in the real word. On the other hand, for the cases that the definition of the environment is ambiguous, reinforcement learning seems cannot perform as well as the examples mentioned above.

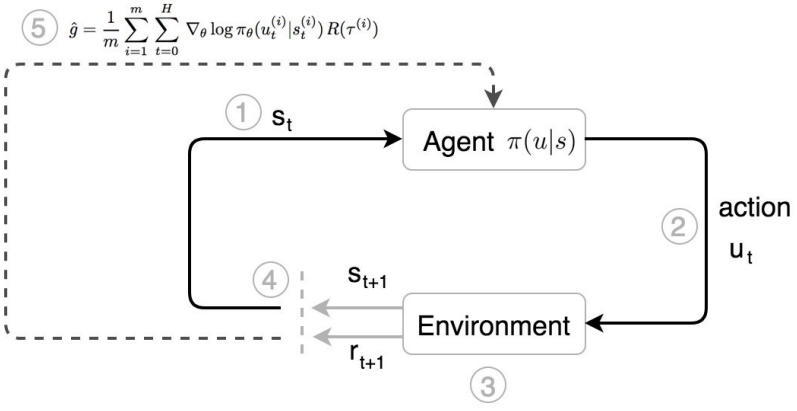


Figure 1: Reinforcement learning (using policy gradient)

To investigate the possibility of modeling the problem of searching for the best trading strategy as a reinforcement learning problem, I tried to describe the environment using prices and volumes and define an action of long or short. Although it seems to be

intuitive to defined the reward function as the immediate return, this reward function does not take the risk factors into consideration: the immediate return of an action might be high but it might also be very risky and we do not want to encourage this kind of actions.

The objective of most of the reinforcement learning algorithm is to maximize some utility function. e.g. profit, economic utility, risk-adjusted return, etc. Usually, this utility function be regarded as the accumulation of the immediate rewards obtained at each step. In this project, to incorporate the risk factor, I choose to use Sharpe ratio ($S_{[m,n]}$) [4] as the utility function, which is commonly used for the investors to understand the return of an investment compared to its risk over a period of time $[m, m+1, ..., n]$. The sharpe ratio can be formulated as:

$$S_{[m,n]} = \frac{Average(R_{[m,n]})}{Standard\ Deviation(R_{[m,n]})}$$

Where $R_{[m,n]}$ is the immediate return sequence in time period $[m, m+1, ..., n]$.

However, it is nontrivial to define a immediate reward which can be added up to be equal to the Sharpe ratio. There are a few previous attempts to use the Sharpe ratio as the utility function. For example, Moody and Saffell proposed to use a direct recurrent learning approach to directly maximize the Sharpe ratio by using the differential Sharpe ratio. [2]

My solution is to use the main setting of policy gradient algorithm to sample a trajectory of a fixed length at each step and update the model using its Sharpe ratio so that the immediate reward function do not need to be defined explicitly.

More detailed algorithm is described in section 2. In section 3, I compared the proposed algorithm with a basic Q learning algorithm.

2 Method

2.1 Policy Gradient

As shown in Figure 1, reinforcement learning modeling the problem as follow: an agent currently in state s_t makes an action a_t, which translate it to a new state s_{t+1} and the agent receives a reward r_t by interacting with the environment. More formally, this is modeled as a Markov decision process $(\mathcal{S}, \mathcal{A}, \mathcal{R}, p, \gamma)$, such that

$$p(s', r|s, a) = Pr\left[S_{t+1} = s', R_{t+1} = r|S_t = s, A_t = a\right] \tag{1}$$

where $S_t, S_{t+1} \in \mathcal{S}$ (state space), $A_{t+1} \in \mathcal{A}$ (action space), $R_{t+1}, R_t \in \mathcal{R}$ (reward space), p defines the dynamics of the process. This means that, the decision of the agent only depend on the current state without considering all the previous states.

Usually, for a given trajectory $\tau = (s_1, a_1, r_1; s_2, a_2, r_2; \ldots; s_T, a_T, r_T)$, we define the total reward of a trajectory τ as $r(\tau) = \sum_{i=1}^{T} r_i$. Since in this problem, our objective is to maximize the Sharpe ratio and the immediate reward is not explicitly used in the following algorithm, we now directly define the total reward of a trajectory τ as its Sharpe ratio. Firstly, we need to define the return of the t-th ($1 \leq t \leq T$) step:

$$R_t = a_t(\log p_{t+1} - \log p_t) \tag{2}$$

Where p_t is the close price at step t. When the action is *long*, $a_t = 1$; when the action is *short*, $a_t = -1$. Then, we can define the total reward of trajectory τ as:

$$r(\tau) = \frac{\bar{R_\tau}}{\frac{1}{T}\sqrt{\sum_{i=1}^{T}(R_{k+i-1} - \bar{R_\tau})^2}} \tag{3}$$

Where $\bar{R_\tau} = \frac{1}{T}\sum_{i=1}^{T} R_{k+i-1}$ is the mean return over the whole trajectory τ.

Then the learning objective of reinforcement learning can be expressed as:

$$\arg\max_{\pi} \mathbb{E}_\pi \left[r(\tau)\right] \tag{4}$$

Where π is the policy that the agent should follow to maximize the total reward, which is defined as the probability distribution of actions given a state:

$$\pi(A_t = a|S_t = s) \;\; \forall a \in \mathcal{A}(s), \; \forall s \in \mathcal{S} \tag{5}$$

Suppose the parameter that we want to learn is θ, then the update rule using gradient ascent is:

$$\theta_{t+1} = \theta_t + \alpha \nabla_\theta \mathbb{E}_\pi \left[r(\tau)\right] \tag{6}$$

Now the problem becomes: how to compute the gradient of the objective function with respect to the model parameter? By abusing the notation π, we define the policy for a trajectory as:

$$\pi(\tau) = \mathcal{P}(s_0)\Pi_{t=1}^{T}\pi(a_t|s_t)p(s_{t+1}, r_{t+1}|s_t, a_t) \tag{7}$$

Where $\mathcal{P}$ represents the ergodic distribution of starting in some state s_0. Here, since a_t only depends on s_t, and (s_{t+1}, r_{t+1}) only depends on (s_t, a_t) by the Markov processing assumption, we can factorize the joint distribution of the trajectory as above.

Then by using the trick of $\nabla_\theta \pi(\tau) = \pi(\tau)\nabla_\theta \log \pi(\tau)$, we can get:

$$\begin{aligned}
\nabla_\theta \mathbb{E}_\pi \left[r(\tau)\right] &= \nabla_\theta \int \pi(\tau) r(\tau) d\tau \\
&= \int \nabla_\theta \pi(\tau) r(\tau) d\tau \\
&= \int \pi(\tau) \nabla_\theta \log \pi(\tau) r(\tau) d\tau
\end{aligned}$$

Which results in the Policy Gradient Theorem [3]:

$$\nabla_\theta \mathbb{E}_\pi \left[r(\tau)\right] = \mathbb{E}_{\pi_\theta} \left[r(\tau) \nabla_\theta \log \pi(\tau)\right] \tag{8}$$

By taking logarithm of Equation 7, we can eliminate $\mathcal{P}(s_0)$ and $p(s_{t+1}, r_{t+1}|s_t, a_t)$ since they are independent of θ:

$$\log \pi(\tau) = \log \mathcal{P}(s_0) + \sum_{t=1}^{T} \log \pi(a_t|s_t) + \sum_{t=1}^{T} \log p(s_{t+1}, r_{t+1}|s_t, a_t) \tag{9}$$

$$\nabla_\theta \log \pi(\tau) = \sum_{t=1}^{T} \nabla_\theta \log \pi(a_t|s_t) \tag{10}$$

$$\nabla_\theta \mathbb{E}_\pi \left[r(\tau)\right] = \mathbb{E}_\pi \left[r(\tau) \left(\sum_{t=1}^{T} \nabla_\theta \log \pi(a_t|s_t) \right) \right] \tag{11}$$

To estimate the expectation in Equation (11), we can randomly sample a large number of trajectory (say N) and average over them to form an unbiased estimator:

$$\nabla_\theta \mathbb{E}_\pi \left[r(\tau)\right] \approx \frac{1}{N} \sum_{i=1}^{N} r(\tau_i) \left(\sum_{t=1}^{T} \nabla_\theta \log \pi(a_{i,t}|s_{i,t}) \right) \tag{12}$$

In this way, the likelihood of actions yielding negative rewards is decreased and the likelihood of actions yielding positive rewards is increased.

In this project, I proposed to use a neural network to approximate the policy function $\pi(A_t|S_t)$. i.e. The probability of taking action A_t at state S_t. Define the Sharpe ratio weighted cross entropy loss of a sampled trajectory $\hat{\tau}$ as:

$$\mathcal{L}(\theta)_{\hat{\tau}} = -r(\hat{\tau}) \sum_{t=1}^{T} \log \pi(\hat{a}_t|\hat{s}_t) \tag{13}$$

This loss function $\mathcal{L}(\theta)$ is especially designed to make its gradient equal to the gradient of the original policy gradient objective function.

2.2 Policy Network

I use a Long Short Term Memory (LSTM) network to simulate the policy function $\pi(A_t|S_t)$. In details, I map the t-th state s_t in a trajectory τ to two scalar: $\pi(A_t = \text{long}|S_t = s_t)$ and $p(A_t = \text{short}|S_t = s_t)$.

First, I define x_t =('Open', 'High', 'Low', 'Close', 'Volume'). 'Open' is the price at the first minute of hour t. 'High' is the highest price in hour t. 'Low' is the lowest price in hour t. 'Close' is the Bitcoin price at the last minute of hour t. I use 'Close' as p_t to calculate the return R_t. All the numbers are pre-processed to take their log form and subtract the number in the previous hour and scaled by the mean and standard deviation of the train data. Then I feed the sequence $(x_1, x_2, ..., x_T)$ to a LSTM layer with sequence length T:

$$f_t = \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \tag{14}$$
$$i_t = \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \tag{15}$$
$$o_t = \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \tag{16}$$
$$c_t = f_t \circ c_{t-1} + i_t \circ \sigma_c(W_c x_t + U_c h_{t-1} + b_c) \tag{17}$$
$$h_t = o_t \circ \sigma_h(c_t) \tag{18}$$

Where W, U, b are learnable parameters and $\circ$ is the element-wise product. This produce a sequence of output vectors $(h_1, h_2, ..., h_T)$.

Basically, a LSTM unit controls the extent of forgetting the information of the previous hidden state h_{t-1} by forget gate f_t. And it always keep a clean path of back propagation through the inner states of the whole sequence to reduce the change of gradient vanishing or exploding.

Then I define the state as $s_t = (h_{t-1}, x_t)$, which is a concatenation of the current input and the previous hidden state. Then the output vector h_t is dropped by a dropout rate of 0.5 and feed into a fully connected layer:

$$(\pi(\text{long}|s_t), \pi(\text{short}|s_t)) = \text{softmax}(W_o h_t + b_o) \tag{19}$$

The overall network architecture is shown in Figure 2. Note that the weights of the dense layer is shared among different time steps.

2.3 Model Inference

At the k-th training step, randomly sample a sequence $(x_1, x_2, ..., x_T)$. The action of each state s_t are chosen by the output $\pi(a_t|s_t)$. Then the forward propagation is Equation

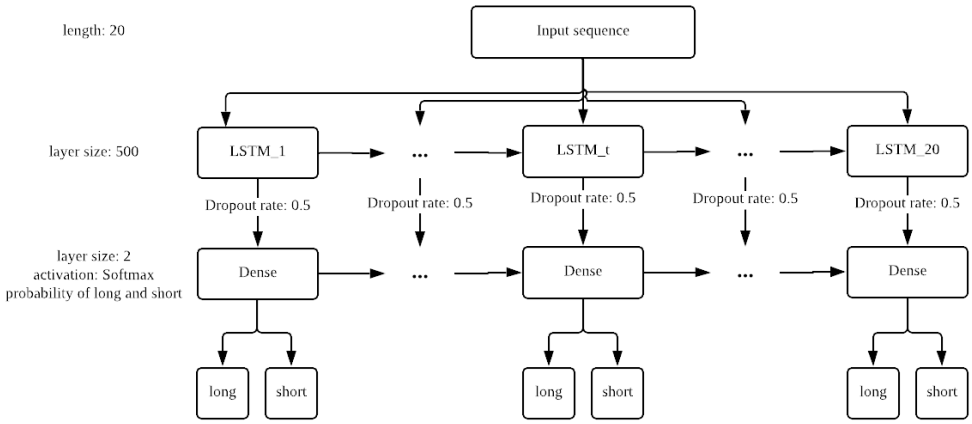


Figure 2: Network structure used to simulate the policy function $\pi(\tau)$.

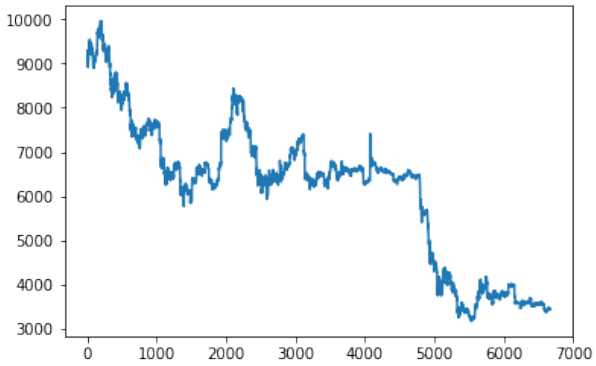


Figure 3: The close prices in the dataset.

(13). And the back propagation is:

$$\frac{\partial \mathcal{L}(\theta)_\tau}{\partial \theta} = -r(\tau) \sum_{t=1}^{T} \frac{\partial \log \pi(a_t|s_t)}{\partial \theta} \tag{20}$$

$$\theta \leftarrow \theta - \alpha \frac{\partial \mathcal{L}(\theta)_\tau}{\partial \theta} \tag{21}$$

3 Experiments

3.1 Data

The data I used is the per-minute Bitcoin data from 2019-01-31 01:20:00 to 2018-04-28 03:05:00 obtained from the MAFS6010U course. The per-minute data is aggregated to per-hour data to reduce the variance. 5336 data is used for training; 640 data is used for validation and testing respectively. All the close prices in the aggregated dataset are shown in Figure 3.

3.2 Baseline

I use the tabular Q-learning algorithm as my baseline. I discretize the states using a moving window of the previous states to fit the current price into three slots. Suppose the current price p_t is compared to m previous prices, then let $\mu_{[t-m+1,t]} = \frac{1}{m}\sum_{i=t-m+1}^{t} p_i$ and $\sigma_{[t-m+1,t]} = \sqrt{\frac{1}{m}\sum_{i=t-m+1}^{t}(p_i - \mu_{[t-m+1,t]})^2}$. We have the discretized price p_t^*:

$$p_t^* = \begin{cases} 0 & \text{if } p_t < \mu_{[t-m,t]} - \lambda\sigma_{[t-m,t]} \\ 1 & \text{if } \mu_{[t-m,t]} - \lambda\sigma_{[t-m,t]} \leq p_t \leq \mu_{[t-m,t]} + \lambda\sigma_{[t-m,t]} \\ 2 & \text{if } p_t > \mu_{[t-m,t]} + \lambda\sigma_{[t-m,t]} \end{cases} \tag{22}$$

In the expriments, I take $\lambda = 0.425$. An illustration of this is shown below:

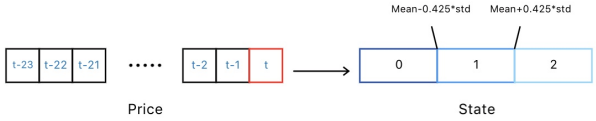


Figure 4: Discretize a continuous price series of 24 into 0, 1, 2

I define the accumulated reward as $\sum_t \gamma^t R_t$ where γ is the discount factor and R_t is return we defined in Equation (2). Then I use a Q function $Q(S_t, A_t)$ to approximate the maximum accumulated reward, which can be updated by the Bellmen equation:

$$Q(S_t, A_t) = R_t + \gamma \max_h Q(S_{t+1}, A_h) \tag{23}$$

We can see that in this setting we are not considering the risk faxtor of the trading strategy. So I use the Sharpe ratio of the previous T steps $S_{[t-T+1,t]}$ as a weight of R_t to perform the update, that is:

$$Q(S_t, A_t) = S_{[t-T+1,t]} R_t + \gamma \max_h Q(S_{t+1}, A_h) \tag{24}$$

Since the number of states and actions are finite, a Q table can be kept and can be updated recurrently using dynamic programming. Random exploration is considered at the beginning of the training to avoid the agent getting stuck in a local maximum.

3.3 Results

I use a trajectory length (or episode length for Q learning) of $T = 100$, a learning rate of $\alpha = 0.0001$ and a batch size of 4. The test data is shown in Figure 5. The cumulative returns $\sum_t R_t$ of both methods on the test set are shown in Figure 6.

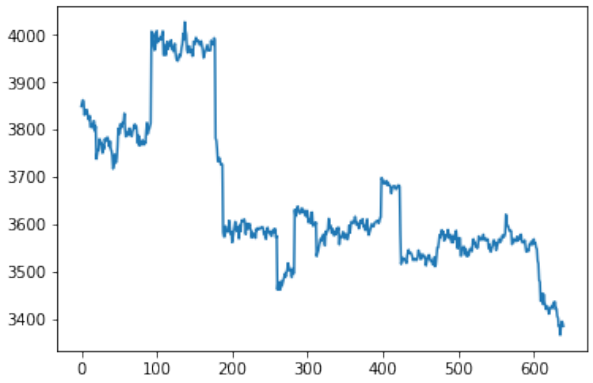
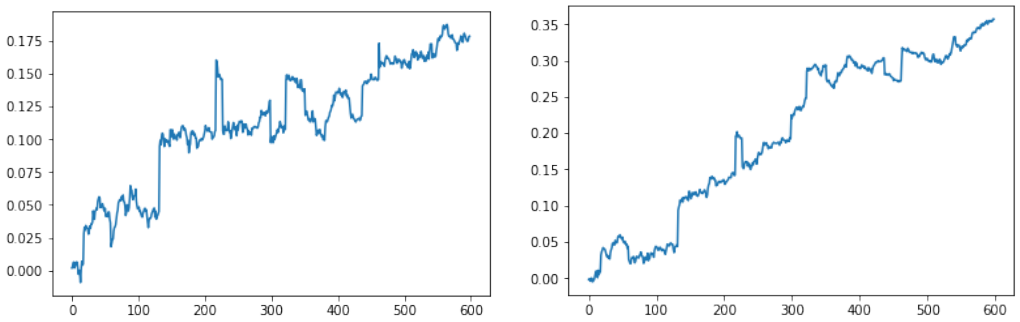


Figure 5: Close prices in the test set.



(a) Tabular Q learning algorithm

(b) Proposed policy gradient algorithm

Figure 6: Cumulative returns on the test set.

The proposed method gives a better cumulative return on the test set. Also, when we look at the Sharpe ratio of them, the proposed policy gradient algorithm has a Sharpe ratio of 0.1108, while the tabular Q learning algorithm has a Sharpe ratio of 0.0551.

4 Conclusion

In conclusion, the proposed policy gradient algorithm can take the risk factor into consideration without explicitly define a immediate reward function. And by directly maximize the desired utility function (i.e. Sharpe ratio), it can get a better result than Q learning which only incorporates the risk factor into the immediate reward function as a weight constant.

5 Future Work

Since a trajectory is actually a large joint distribution of policy at different stages, the variance can be very large which is harmful for the optimization. In the experiments, I noticed that the proposed algorithm is not very robust and it can display very different training results in different trails with identical hyper-parameters. For future work on this, some regulations could be introduced to lower the variance and increase the

robustness of the algorithm. A variance of the policy gradient algorithm is the Actor-Critic algorithm, which introduced the state value as a 'critic' to judge whether the cumulative reward of the trajectory is good or not. It is possible to use this idea to reduce the variance and get a better result.

We define a state value as the mean Sharpe ratio for a trajectory τ:

$$V(\tau) = \mathbb{E}_\pi[r(\tau)] \tag{25}$$

We can approximate this state value using a separate neural network with parameter ω and use a running average of $r(\tau)$ as the target values for update:

$$V_{k+1}(\tau) \leftarrow \gamma r(\tau) + (1-\gamma)V_k(\tau) \tag{26}$$

Where $\gamma < 1$ is a small value determining the influence of the total reward of a new trajectory on the state value.

Then the Actor-Critic gradient can be accordingly updated as: [1]

$$\nabla_\theta \mathbb{E}_\pi \left[r(\tau)\right] = \mathbb{E}_\pi \left[\left(r(\tau) - V(\tau)\right) \left(\sum_{t=1}^{T} \nabla_\theta \log \pi(a_t|s_t) \right) \right] \tag{27}$$

And the previous loss function in Equation 13 becomes:

$$\mathcal{L}(\theta)_\tau = -(r(\tau) - V(\tau)) \sum_{t=1}^{T} \log \pi(a_t|s_t) \tag{28}$$

Note that we still need to update the new parameter ω. The critic's objective is taken to be the Mean Squared Loss:

$$J(\omega) = \frac{1}{2} \left(r(\tau) - V(\tau)\right)^2 \tag{29}$$

The the back propagation is:

$$\omega \leftarrow \omega - \alpha(r(\tau) - V)\frac{\partial V(\tau)}{\partial \omega} \tag{30}$$

References

[1] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Tim Harley, Timothy P. Lillicrap, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, pages 1928–1937. JMLR.org, 2016.

[2] John Moody and Matthew Saffell. Learning to trade via direct reinforcement. *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, 12:875–89, 07 2001.

[3] John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *CoRR*, abs/1506.02438, 2015.

[4] William F. Sharpe. Mutual fund performance. *The Journal of Business*, 39(1):119–138, 1966.