Advanced Reinforcement Learning Project: An RL Approach for Trading Strategy of Stock Index

Dayuan Wei, Hongyan Yang
Northeastern University, Khoury College of Computer Sciences
wei.da@northeastern.edu
yang.hongy@northeastern.edu

Abstract

We explore the problem of training and testing an automatic stock index trading strategy and the effectiveness of deep reinforcement learning algorithms including Deep Q-Learning and off-policy Actor-Critic (AC), to learn an optimal control policy for the task of maximizing the profit during a fixed investment period (e.g., 200 days) measured by discounted cash flow (or reward in RL's term).

We explore the impact of different designs of reward function named the Strict Mode and Flexible Mode and compared their effect on agent's learning rate and performance.

We also explore the impact on performance of several types of epsilon-value decaying methods to balance exploration and exploitation of the agent, including a fixed small epsilon value approach and a linear decaying epsilon value method.

We show that both DQN and off-policy Actor-Critic can solve the task.

Introduction

Our approaches are by nature the deep-learning version of technical trading skills. Conventional technical trading strategies largely rely on metrices derived from stock price, such as 100-day price's moving average, 15-day volatility applied in the famous risk parity model and candlestick patterns (e.g., Head and Shoulders Pattern known as a typical reverse signal), while our approaches apply the Deep Q Networks (DQN), a universal approximation to extract all relevant information from historical prices to complete the task in a more comprehensive and effective manner.

In more details, the theoretical basis of applying the past index price and trading volume information to forecast price movement in the future is Behavior Finance theory.

Behavior Finance Theory focuses on explaining why psychological factors play a dominant role in decision making regarding investment in security markets. Sentimental factors allied to investment like panic and loss aversion in the bear market or overly exuberance in the bull market causes herding behavior, resulting in a trend of stock price movement and an underlying momentum of price change.

However, based on the assumptions of Efficient Market Hypothesis (EMH), it is hard to make consistent excess return using traditional approaches because 1). share prices reflect all publicly available information which conveyed at a fair transmission cost and 2). stocks are traded at their market value after considering a fair rate of transaction cost.

Thus, an RL approach is in need to 1). automate the trading process to save transaction and labor costs and to 2). make information processing and investment operation way faster. The former is an important factor that determines if a certain strategy is feasible or not, especially for institutional investors. The latter is a factor that makes a machine approach superior to human operation.

What's more, an advanced RL approach using trained Deep Q Networks (DQN) is required to deal with one of the challenges in applying reinforcement learning to real-world problems, that is how to represent and store value functions and policies in an effective way. Unless the state set is finite and small enough to allow exhaustive representation by a lookup table, one must use a parameterized function approximation scheme. And instead of relying on sets of features carefully handcrafted based on human knowledge and intuition about the specific problem to be tackled, a deep multilayer ANN can automate the feature design process to obtain striking results by coupling reinforcement learning with backpropagation method. (Mnih et al., 2013, 2015).

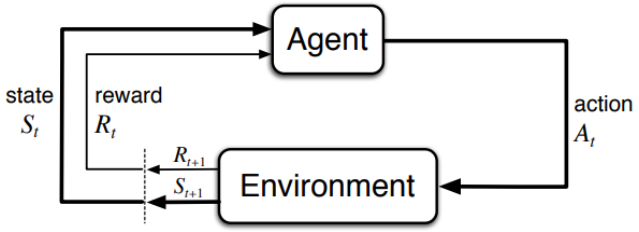
Using this as motivation, we apply a set of reinforcement learning algorithms to the problem of designing stock trading strategies. Our aim is to demonstrate the effectiveness of reinforcement learning to make constant profit not only in the training environment but also in the testing environment.

We implement two reinforcement learning algorithms—Q-learning and Actor-Critic—and examine their performance via plotting of their learning curves and a clear comparison of their performance in the training set and testing set. We also explore several types of hidden layer structures, epsilon-value decaying methods and analyze their impact on model's performance.

Background

Formulating the Reinforcement Learning Problem

In an RL problem, an agent learns from interaction to achieve a goal. The agent and its environment interact continually, the agent selecting actions and the environment responding to these actions and presenting new situations to the agent. The environment also gives rise to rewards, special numerical values that the agent seeks to maximize over time through its choice of actions. (Sutton et al, 2018)



Markov Decision Processes

The MDP framework is a considerable abstraction of the problem of goal-directed learning from interaction. As a classical formalization of sequential decision making, MDPs involve delayed reward and the need to trade immediate and delayed reward. In MDPs we estimate the value $q*(s, a)$ of each action a in each state s, or we estimate the value $v*(s)$ of each state given optimal action selections. MDPs are a mathematically idealized form of the reinforcement learning problem for which precise theoretical statements can be made. (Sutton et al, 2018)

Using this framework, our goal is to learn an estimation of the value function mappings from state s, or state-action pairs (s, a), assuming optimal action selection along each episode. These estimations are then used for the Generalized Policy Iteration (GPI) to guide optimal action selection.

Q-Learning

Q-Learning is an off-policy TD control algorithm, where the learned action-value function, Q, directly approximates $q*$, the optimal action-value function, independent of the policy being followed. (Sutton et al, 2018)

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\right]$$

As more trajectories sampled, the estimated Q functions will converge to the optimal value $q*$ and thus an optimal policy $\pi*$ can be derived.

Double Q-Learning

Q-Learning suffers from the maximization bias like other control algorithms which involve maximization in the construction of their target policies. It can lead to a significant positive bias because the algorithm takes the maximum of estimates as the true value and makes all relevant value updates toward that direction. It makes algorithms slower to converge and derived optimal policies with higher variance.

Double Q-Learning avoids maximization bias by using one set of samples to determine the maximizing action and using the other sample set to estimate the value. We applied Double Q-Learning and compared its performance with vanilla Q-Learning in the training process.

With 0.5 probabilility:

$$Q_1(S, A) \leftarrow Q_1(S, A) + \alpha\Big(R + \gamma Q_2\big(S', \operatorname{argmax}_a Q_1(S', a)\big) - Q_1(S, A)\Big)$$

else:

$$Q_2(S, A) \leftarrow Q_2(S, A) + \alpha\Big(R + \gamma Q_1\big(S', \operatorname{argmax}_a Q_2(S', a)\big) - Q_2(S, A)\Big)$$

Policy Gradient Methods

Different from the action-value methods such as the Q-Learning that learns the values of actions and then selected actions based on their estimated action values, policy gradient methods learn a parameterized policy that can select actions without consulting a value function. With this method, action preferences are predicted as a probability distribution over the action space. (Sutton et al, 2018).

The Actor network we constructed in the Actor Critic Model follows a policy gradient method and it outputs the probability distribution over discrete actions.

Related Work

DQN with convolutional layer

The convolutional layer in a Deep Q Network applies the convolution operation to the former input layer. This operation converts every element and other elements within its neighbor field into a single value. Using convolutional layer has the advantage of preserving the input's neighborhood relations and spatial locality in their latent higher-level feature representations.

We tried to use convolutional layer in our DQN model, but our current state S is a one-dimensional vector representing a period of past stock price changes. It does not satisfy the requirement of a higher dimensional input for conducting the convolutional transfer operation.

In the future, we plan to add volume of daily trading to the state S and apply convolutional layer in our model. It requires much more time to design a suitable state size and preprocessing treatment accordingly.

Project Description

We describe the problem of learning a control-policy to make stock trading decisions aiming at maximizing profits during a given investment horizon.

Markov Decision Process Formulation

We can define our problem as a finite-horizon Markov Decision Process, represented by the four-tuple (*S, A, T, R*). The state is described by the following features of the environment:

$$S = \{price_{vector}, i, \beta_{balance}\}$$

$$price_{vector} = \left(f(\Delta p^{t=i-T}), f(\Delta p^{t=i-T+1}), \dots, f(\Delta p^{t=i})\right)$$

where $(price_{vector}, i, \beta_{balance})$ correspond to the historical stock price in a period (e.g., past 50 days), current time stamp in the time series data set and agent's remaining balance, respectively. $price_{vector}$ in practice is a vector of daily price change transformed by a Sigmoid logistic function for regularization purposes.

The discretized actions are defined as:

$$A = \{(-1,0,1)_{t \in T}\}$$

Corresponding to the operation of sell, hold and buy one share of stock given an observation of state *S*.

We define two modes of executing these actions: Strict Mode—where actions update the balance $\beta_{balance}$ of the agent by the logic of day trading: action a_t is followed by a counter action a_{t+1} the next trading day. In this mode, the trained model makes profit by correctly forecasting the price change direction of one day; and Flexible Mode—where the agent makes long term investment by choosing action a_t without following a counter action the next day. In this mode, profit is calculated follows a FIFO logic and the agent might make a sequence of buy actions in a bull market to maximize the total return.

The transition function *T* is stochastic and assumed to be independent of agent's actions due to the nature of the stock market. By operating only one share of stock at time stamp *t*, we assume we would not affect the market price.

Finally, the reward function is defined as:

1). In the Strict Mode:

$$R = A * (Price_{t+1} - Price_t), t \in T$$

2). In the Flexible Mode:

$$R = \max\left((Price_t - Price_{t*}), 0\right) \; if \; A = -1$$

where $Price_{t*}$ is the price at *t** when the earliest buy action was conducted and $R = 0$ in other situations.

DQN and Double-DQN

We train and explore the performance of a deep learning variation of Q-Learning known as DQN on the problem. We apply a Deep Q Network to approximate the Q functions and apply a TD update to adjust the weights of the Deep Q Network, and a backpropagation method is applied to train the network.

We define the loss variant as follows:

$$loss_t = R_{t+1} + \gamma * argmax_a Q(S_{t+1}, a) - Q(S_t, A_t)$$

We applied stochastic gradient descent as the optimizer by taking the gradient of the $loss_t$ with respect to the weights of the network under training. And the weights are updated by a learning rate *lr* toward the correct direction.

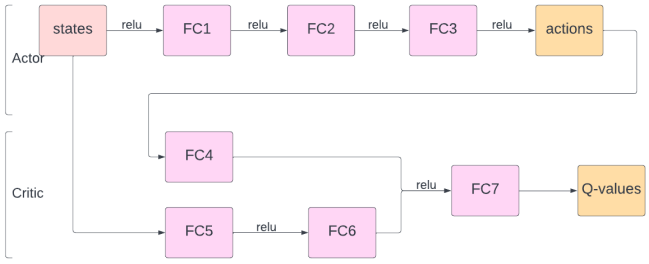
We also train and explore the performance of a Double-DQN. Their learning curve and total discounted returns are compared. Because a Double-DQN avoids the maximization bias, it is expected to converge more smoothly and to perform better.

Both DQN and Double-DQN are trained in the Strict Mode following the reward function in the mode.

Off-Policy Deep Actor-Critic

We also apply the Actor-Critic method to solve the problem and evaluate its performance. In our AC model, the Actor network has three hidden fully connected layers, and it uses a SoftMax activation function to map each state to the discrete action space. A probability distribution is generated, and it represents the preference of different actions.

We then pass the output from the Actor network to the Critic network, which takes actions' probability distribution and state as its input. During the forward propagation, it combines these two inputs in one hidden layer and output predicted Q-values of all actions.



In both the Actor and the Critic network, we apply relu activation function and batch normalization function right after each hidden layer. The relu function outputs the input directly if it is positive, otherwise, it will output zero and it helps to solve the vanishing problem. Batch normalization is used to normalize each layer's input to make the training faster and more stable.

Furthermore, to stabilize and form an off-policy variation of AC model, we also integrate a standard replay buffer. At each training step, we sample from the replay buffer and use TD error to compute the Actor and Critic losses. We add transitions in the form of

$$\{S_t, A_t, R_t, S_{t+1}, Done\_t\}$$

into the queue at each time step and randomly sample from the buffer to train the model.

Soft Update of Actor-Critic Weights

To reduce variance in the training process and obtain smooth convergence. We update the target model's weights following a Soft Update algorithm (Taisuke Kobayashi, Wendyam Eric Lionel Ilboudo, 2021).

Algorithm: Soft Update

Input: local network, target network
Parameter: τ, which measures the update ratio
Output: target network weight

$weight_{local}$ = get_wight(local network)
$weight_{target}$ = get_weight(target network)
if len($weight_{local}$) != len($weight_{target}$) **then**
    break
else
    $weight_{target} = \tau * weight_{local} + (1 - \tau) * weight_{target}$
end if
return $weight_{target}$

Experiments

In total we ran approximately fifty experiments for both DQN and Actor-Critic methods as several grid-searches over hyperparameters. We experimented with state S with different sizes to determine how many past trading days' prices changed should be considered. We also tried varying dimension sizes for the neural network layers, the number of layers per network, and especially the learning rate for the Actor-Critic network.

We also evaluated the performance of the Strict Mode and Flexible Mode on both DQN and Actor-Critic models. The two modes have different reward functions as described above.

Environment

The environment that we use is the standard and poor's 500. It is An American stock market index based on the market capitalizations of 500 large companies having common stock listed on the NYSE or NASDAQ.

The date in the training set is as follows:

$$Start: 2\ Jan, 2002$$
$$End: 31\ Dec, 2014$$

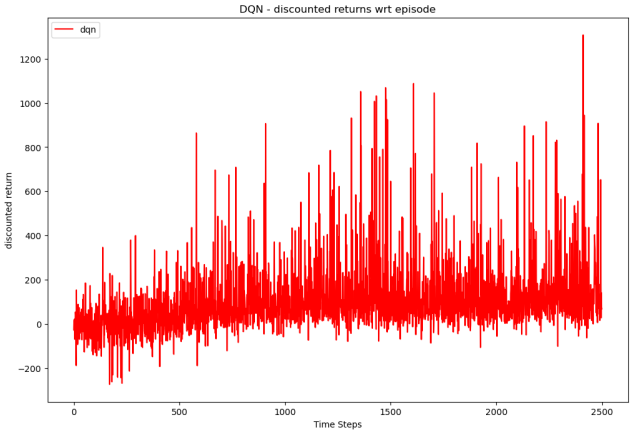
The date in the testing set is as follows:

$$Start: 2\ Jan, 2015$$
$$End: 23\ Nov, 2022$$

Results & Conclusions

We found that, the hyperparameters that have most profound impact on learning curve and discounted returns are state size in both Strict Mode and Flexible Mode, the learning rate lr in the Adam Optimizer and whether the reward function is defined in Strict Mode or not.

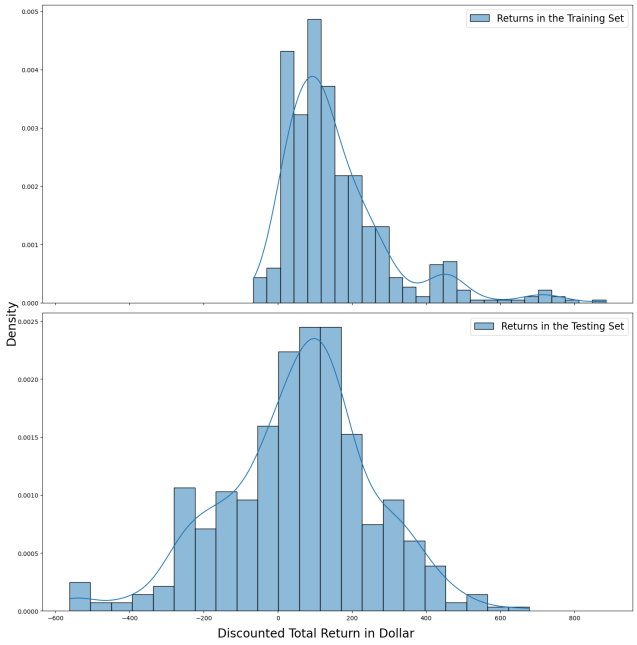
DQN Results



When training under the Strict Mode, the loss wrt training steps have high volatility as expected. The reason is that the model is trained to forecast a precise price change in the next trading day instead of just the price change direction as in the Flexible Mode. So, the parameters of the underlying Deep Q Network are updated towards the correct direction at each time step but hard to converge to an optimum. Although the discounted returns wrt episodes also performs with high volatility, it has a clear upward trend in the early time steps with the average discounted return during the first 300 steps around 60 dollars in the given investment horizon

(200 days in the model) and it increases to around 180 dollars in the latter time steps.

It's convincing that the model is well trained in the training environment as shown in the following plots which share the same x-axis. We followed the investment decision (buy, hold, or sell) offered by the trained model and conducted another 500 independent investments in both the training and testing environment and the results are as follows:



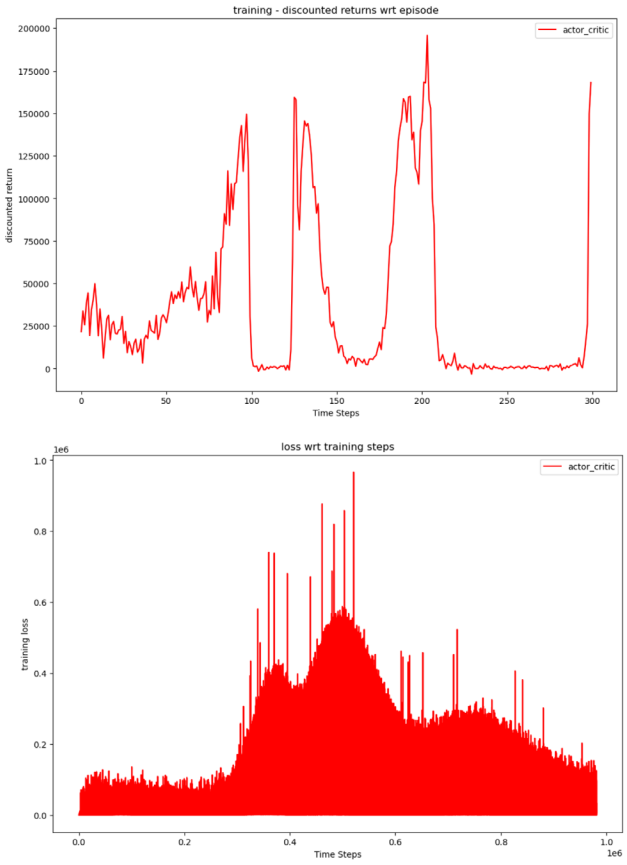
The upper plot shows how the trained model performs in the training environment. It makes correct investment decisions almost all the time as shown by the highly right-skewed distribution with almost no negative returns. It's worth noting that the agent doesn't take the *hold* action too many times as shown by a low density corresponding to 0 discounted return in dollar. Considering the reward function in the Strict Mode, where the agent only makes a positive return by forecasting one trading day's price change correct and the return value is exactly the price change that totally depends on the market. The model obviously learned and was well trained in the training environment.

The lower plot shows how the trained model performs in the testing environment. It makes correct investment decision more than half of the time as can be shown by the bell-shaped distribution with the mean around 110 dollars. It has positive returns with high density almost the same as that in the training set as the two plots share the same x-axis. However, the return distribution in the test environment doesn't show the highly right skewness. It can be explained as follows: 1). Under the Strict Mode the agent makes positive return by correctly forecasting one day's price change but due to the nature of stock market and a transition function that's independent of agent's actions, the embedded volatility in the market makes it a challenge to make a correct forecast. 2). By the design of our environment, the training and testing sets are continuous real time series data rather than random shuffled ones. Thus, the market suffers from regime change risks and policy risks and even a perfect well-trained model will fail to grasp all relevant information in the "new" market to make correct actions. 3). The Deep Learning Networks we use has three hidden layers and many hyperparameters tuned. The model is expected to be overfitting with high variance and little bias as shown in the two plots.

Overall, the DQN model performs well and is expected to make positive returns around 110 dollars in the testing set under Strict Mode with high density, which is almost the same as it did in the training set.

We also applied a Double DQN model to avoid the maximization bias with the same hyperparameters set. The model converges slightly smoother but output similar performance. We also tried a fixed epsilon rather than a linear decayed epsilon, and the model also outputs similar results.

Actor-Critic Results

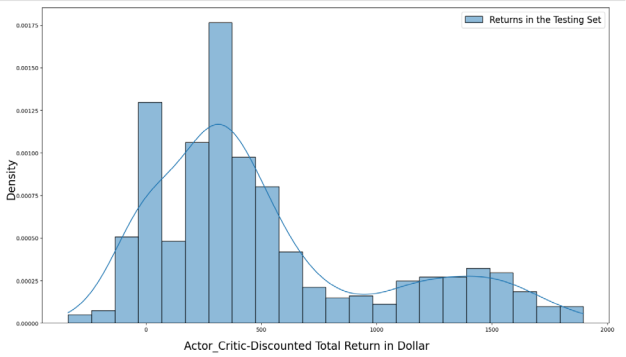


When trained under the Flexible Mode, the loss wrt training steps of the model has lower volatility compared to that under Strict Mode. The loss decreases as the iteration proceeds,

showing the model's weights converges. It is achieved by the soft update strategy we applied to adjust the model weights as we explained before and by a careful choice of the learning rate *lr* in the Adam optimizer. Our choice of *lr* is quite small to avoid the overshoot that makes the learning jump over minima. We also tried smaller *lr* but a too low learning rate takes too long to converge and sometimes the model get stuck in an undesirable local minimum.

We tried a myriad of hyperparameters to balance the variance and bias, and to have the model tend to converge at the training environment while not overfitting to be useless in the testing environment.

We finally trained a well performed model that also makes constant positive returns in the testing set as shown below:



It is worth mentioning that under the Flexible Mode, the agent has a discrete action space to choose action from and the agent acts quite cautiously compared to that under the Strict Mode. This time the agent chooses $a = Hold$ around 80 percent of the time and it always makes profit when it chooses $a = Buy$ or $a = Sell$ the rest of the time.

Under Flexible Mode, the agent is not required to perform counter action the next trading day, a well-trained agent makes profit by buying low and selling high without the limit of investment horizon. Our model clearly learns the state *s* that's likely to lead a bull market, it also learns to spot the state *s* that is likely to lead a bear market. So, the model gets a positive return by buying at a low stock price and selling at a high stock price. The model chooses to do nothing when the market trend is unclear.

References

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

Kobayashi, Taisuke, and Wendyam Eric Lionel Ilboudo. "T-soft update of target network for deep reinforcement learning." *Neural Networks* 136 (2021): 63-71.

Lee, Jae Won. "Stock price prediction using reinforcement learning." *ISIE 2001. 2001 IEEE International Symposium on Industrial Electronics Proceedings (Cat. No. 01TH8570)*. Vol. 1. IEEE, 2001.

Derman, Esther, et al. "Soft-robust actor-critic policy-gradient." *arXiv preprint arXiv:1803.04848* (2018).

Naseer, Mehwish, and Dr Bin Tariq. "The efficient market hypothesis: A critical review of the literature." *The IUP Journal of Financial Risk Management* 12.4 (2015): 48-63.

Zarkias, Konstantinos Saitas, et al. "Deep reinforcement learning for financial trading using price trailing." *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2019.

Zejnullahu, Frensi, Maurice Moser, and Joerg Osterrieder. "Applications of Reinforcement Learning in Finance--Trading with a Double Deep Q-Network." *arXiv preprint arXiv:2206.14267* (2022).

Tsantekidis, Avraam, et al. "Price trailing for financial trading using deep reinforcement learning." *IEEE Transactions on Neural Networks and Learning Systems* 32.7 (2020): 2837-2846.

Peters, Jan, Sethu Vijayakumar, and Stefan Schaal. "Natural actor-critic." *European Conference on Machine Learning*. Springer, Berlin, Heidelberg, 2005.

Grondman, Ivo, et al. "A survey of actor-critic reinforcement learning: Standard and natural policy gradients." *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 42.6 (2012): 1291-1307.

Jia, Yanwei, and Xun Yu Zhou. "Policy gradient and actor-critic learning in continuous time and space: Theory and algorithms." *Journal of Machine Learning Research* 23.154 (2022): 1-55.