Stocks Trading with Reinforcement Learning

Youzhi Luo
Texas A&M University
College Station, TX 77843
yzluo@tamu.edu

Limei Wang
Texas A&M University
College Station, TX 77843
limei@tamu.edu

Abstract

Stocks trading can be easily transferred to a decision making problem, so it is natural to apply reinforcement learning to it. The objective of our course project is to build a stocks trading system with a deep reinforcement learning algorithm as the decision agent. We apply Deep Q-learning and Deep duel Q-learning algorithm, and try fully connected network and 1D convolutional neural network in this project. Experimental results show that our trained system can successfully learn to do stocks trading and keep making profits in test data.

1 Introduction

Since the appearance of first financial market, people have been trying to predict future trend of price movements for making as many profits as possible. One of the most widely happened financial activity in people's daily life is stocks trading. In this financial condition, what people is looking forward to achieving is to buy stocks at low prices and to sell them at high prices.

In fact, the process of stocks trading can be formalized as a decision problem easily — at each trading moment, people are access to some observation (e.g. historical prices), and make some decisions (e.g. buy, sell or wait) to maximize their expected future rewards (profits). Thus it is possible to apply reinforcement learning to stocks trading problem.

In this project, we model the stocks trading process as a Markov decision process, clearly define the state, action and reward. We develop a deep reinforcement learning based system, use two types of deep neural network architecture as the Q-value estimator, train on our collected stocks pricing data to enable the agent to learn to make profits during stocks trading. All the code is publicly available on GitHub[1] and the project video is publicly available on YouTube[2].

2 Related Work

Reinforcement learning is a main branch in the field of machine learning. It focuses on the problem that given an environment, how intelligent agents ought to take actions in this environment in order to maximize the cumulative rewards. Formally, a Markov Decision Process (MDP) is usually used to model this process. An MDP is typically a tuple of four objects $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \pi, p)$. $\mathcal{S}$ is the state space containing all possible states of the environment, $\mathcal{A}$ is the action space containing all possible actions that the agent can take, $\mathcal{R}$ is the reward space, $\pi(a|s)$ is a policy denoting the probability of taking action a under the state s, $p(s', r|s, a)$ denotes the environment dynamics defining the probability of transferring to the state s' and receiving the reward r when taking action a under the state s in the environment. The objective of single-agent reinforcement learning is to find an optimal policy π^* so

[1] https://github.com/lyzustc/stocks-trading-with-RL
[2] https://youtu.be/5GkFCeuyhfc

as to maximize the expected cumulative rewards along the time:

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{s,r\sim p,a\sim\pi}(\sum_{t=0}^{\infty} \gamma^t r_t)$$

Deep reinforcement learning is the combination of reinforcement learning with deep learning. In the most recent years, people have made great progress in Atari game, Go chess [2,8] and many other complicated control problems, by combining the classical reinforcement algorithms with fancy deep neural network structure.

Basically, there are two class of methods in reinforcement learning. One is value-based methods, and the most widely used is Q-learning [3] method. This method constructs a $\epsilon-$greedy policy for the agent with a value function table or approximator, and use the following temporal-difference updates to modify value function estimation in each time step as a way of improving policy:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$$

Deep Q-learning (DQN) [1] uses a deep neural network as the function approximator to compute value function. Instead of back propagating at each time step, the network is updated with a random selected batch of data from past time steps, or experience replay buffer. The network is updated with the following loss function, where Q^{past} is computed by target network with old parameters.

$$L(\theta) = \mathbb{E}_{s,r\sim p,a\sim\pi}[r + \gamma \max_{a'} Q^{past}(s',a'|\theta^{past}) - Q(s,a|\theta)]^2$$

Based on DQN, Wang et al. [14] proposed to use a dueling network architecture, to learn the state value function and state-action advantage function instead of directly learning Q-value function. They argue that there is no need to estimate each action value for all states, but estimating the state value is significant. The introduction of dueling network has been proven to outperform the naive DQN algorithm.

The other class of methods is policy-based methods. These methods use a policy estimator (e.g. deep neural network) to directly estimate the probability $\pi(a|s)$ of taking each action a under a certain state s. The policy estimator is updated using the policy gradient $\nabla_\theta \log \pi(a_t|s_t;\theta)R_t$ [4]. Deep Deterministic Policy Gradient (DDPG) [5] uses two deep neural networks in training — one is the actor neural network which determines the action, and a critic network which outputs the value function. The critic network is updated with the same method in Deep Q Learning, while the actor network is updated with the deterministic policy gradient (DPG) [6]. Proximal Policy Optimization (PPO) [7] is another kind of policy gradient method. It aims to take the biggest possible improvement step on a policy using the data we currently have, without stepping so far that performance collapse is caused.

Currently, deep learning and reinforcement learning have not been widely used in finance, but there are some exploration in applying them in business management, price prediction or trading. Heaton et al. [9] applied deep learning method in finance, and claimed that they are more powerful than some standard analysis methods in finance with a case study. Bao et al. [10] used a combination of wavelet transforms, stacked auto-encoders and long-short term memory (LSTM) [11] in the problem of stock price forecasting. Deng et al. [12] applied deep reinforcement learning techniques in trading problem.

3 Methods

3.1 System Overview

In this project, we try developing two stocks trading system, one based on naive deep Q-learning [1] algorithm and the other based on deep duel Q-learning [14]. In addition, we try applying two different deep neural network structures in each system— fully connected neural network and 1D convolution neural network as the function approximator to estimate the Q-value for each action.

The overall architecture of our stocks trading system is shown in Figure 1. At time t, an observation denoting the current pricing information of the stocks is derived from the stocks trading environment, which is further transformed to the state vector by the agent. The agent inputs the state vector into the neural network, outputs the estimation of Q-value for each action, and then uses $\epsilon-$greedy policy to decide the action to take. After the action is taken, a reward is generated from the environment as the feedback. During the training stage, the reward is utilized to train the deep neural network. In the test stage, we use the accumulated sum of rewards as the metric to evaluate the algorithm's performance.

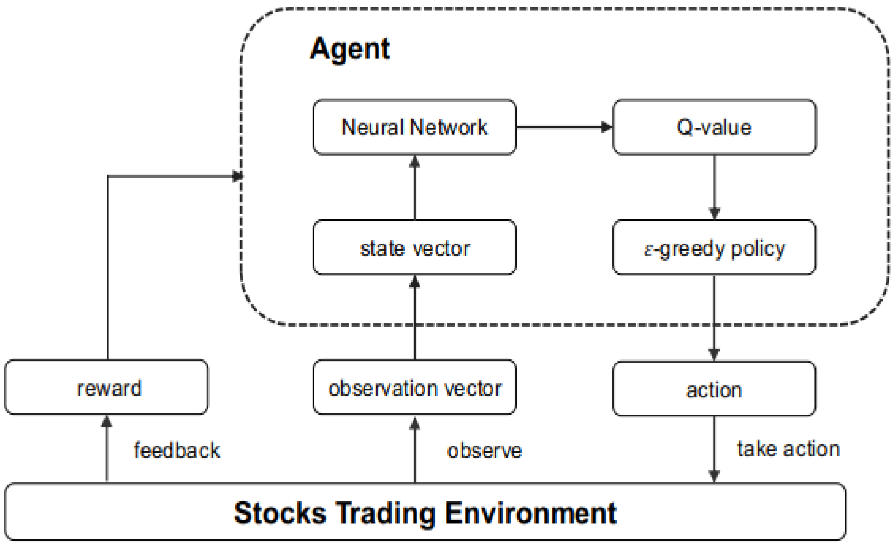


Figure 1: Overall architecture of system

3.2 Stocks Trading Environment

The stocks trading process in the real world can be very complicated — one person can buy or sell multiple stocks in any time. To simplify the problem, we add some constraints to our agent. Since we only have the stock's pricing data of one company, in our project we only consider making agent learn to make trading about this company's stock. We split the time into intervals, where each interval is one minute. The agent can observe the stock's pricing information during each interval, but it is allowed to make trading only at the end of the interval, and the agent is only allowed to buy a share or sell a share of stock at the trading moment. It is expected that our trained stocks trading program is able to make intelligent trading decision based on the observed historical price information of stocks so as to buy or sell a share of stock at the best time. We make use of an implemented gym-like stocks trading environment [13] to simulate the stocks trading process. Detailed description about state, action, reward and data is below.

3.2.1 State Space

The observation at each time is represented as a four-item vector. Each observation vector contains the price information of the stock in the interval of one minute, that is, the trading price at the beginning of the interval p_t^o, the highest price during the interval p_t^h, the lowest price during the interval p_t^l and the price at the end of the interval p_t^e.

To assist the network to be more stable during training, we do not use the raw observation as the input to the network, but defining the relative price at each time t as the state. The price information from the observation is encoded in the following way:

$$r_t^h = \frac{p_t^h - p_t^o}{p_t^o}$$

$$r_t^l = \frac{p_t^l - p_t^o}{p_t^o}$$

$$r_t^e = \frac{p_t^e - p_t^o}{p_t^o}$$

At any time t, we will use a history of 10 time steps' price information as the state. In addition to the price information, we will also encode the information of whether the agent has a share of stock currently or not h_t into the state vector ($h_t = 1$ if the agent is holding a share of stock otherwise

$h_t = 0$), and the value of the profit rate η_t if the holding stock is sold at this time ($\eta_t = \frac{p_t^e - p_{t'}^e}{p_{t'}^e}$ if the agent currently holds a share of stock which is bought at time t', otherwise $\eta_t = 0$).

We will use a fully connected network or a 1D convolutional neural network as the function approximator to estimate Q-value. If the network is a fully connected network, the state is a $32-$d vector, forming by connecting $r_{t-9}^h, r_{t-9}^l, r_{t-9}^e, \ldots, r_t^h, r_t^l, r_t^e$ and h_t, η_t together. But if the network is a 1D convolutional neural network, we will construct a matrix of size 5×10 as the state, where the first three rows are r^h, r^l, r^e from last 10 time steps, the fourth row is h_t copying 10 times and the fifth row is η_t copying 10 times.

3.2.2 Action Space

There are only three action at the trading moment — sell a share of the stock (only if the agent has a share of stock), buy a share of the stock (only if the agent does not have a share of stock) and do nothing. So the agent can only hold one share of the stock at any time.

3.2.3 Reward

The reward is proportional to the amount of profit to receive when the agent sell its share of stock, that is, the difference between the selling price and the buying price. In our project, we define the reward at time t as $r_t = 100\eta_t$ if the holding share of the stock is sold at this time, where η_t is defined in section 3.1.1, otherwise $r_t = 0$.

3.2.4 Data

We have collected the stock market price records of a Russian company from the year of 2015 and 2016 as our data in all experiments, which are saved as '.csv' files. Each record contains a four-number tuple corresponding to $p_t^h, p_t^l, p_t^o, p_t^e$ at each time step t. We choose data from the year of 2015 for training and evaluate the trained stocks trading system on the data from the year of 2016.

3.3 Neural Network Model

We utilize two types of deep neural network — a 3-layer fully connected neural network and a 4-layer 1D convolutional neural network in our system. The detailed architectures of these two networks are shown in Table 1 and Table 2 separately.

Table 1: Fully Connected Neural Network

layer	number of filters
fc1	512
fc2	512
fc3	3

Table 2: 1D Convolutional Neural Network

layer	number of filters	kernel size
1D-conv1	128	5
1D-conv2	128	5
fc3	512	-
fc4	3	-

3.4 Reinforcement Learning Algorithm

We apply two types of deep Q-learning algorithm in our stocks trading system. The first is based on naive single-stream deep Q-learning algorithm. In this framework, a single deep neural network is used to directly output the estimation of Q-value for each action. The second is based on dueling

Q-learning. This framework uses a two-branch deep neural network, where one branch predicts the state value and the other predicts the advantage for each action. Then the Q-value for each action is computed as:

$$Q(s,a) = V(s) + A(s,a) - \frac{1}{|\mathcal{A}|}\sum_{a'} A(s,a')$$

For dueling Q-learning, we also try applying two neural network architectures — fully connected neural network and 1D convolutional neural network. In the first architecture, $V(s)$ and $A(s,a)$ are estimated with two different neural network θ_1 and θ_2 separately, where θ_2 has the structure of table 1 and the first two layers of θ_1 is the same as in Table 1 but the third layer has only one output channel. In the second architecture, a single two-branch neural network predicts $V(s)$ and $A(s,a)$. The two 1D convolutional layers are shared, but two different fully-connected branches predicts $V(s)$ and $A(s,a)$ separately.

The training flow of deep Q-learning is shown in Algorithm 1. Both naive deep Q-learning and dueling Q-learning need a target network θ' to make training more stable, which is updated by copying from evaluation network periodically.

Algorithm 1 Deep Q-learning

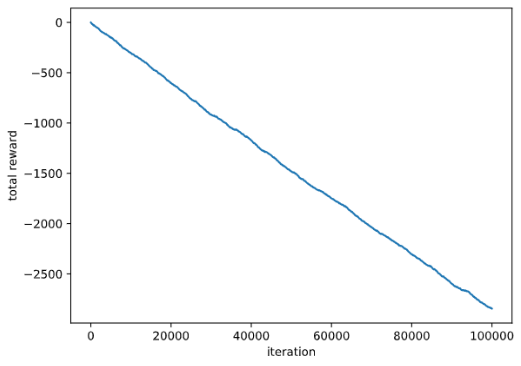
```
Initialize evaluation network parameters $\theta$
Initialize target network parameters $\theta' = \theta$
Initialize empty replay buffer $\mathcal{D}$
for episode=1→M do
    Initialize state $s_1$
    for $t = 1 \to T$ do
        Choose action $a_t$ with $\epsilon-$greedy policy
        Execute action $a_t$ in the environment
        Observe reward $r_t, s_{t+1}$ and done signal $d_{t+1}$ which indicates whether $s_{t+1}$ is terminal
        Store $(s_t, a_t, r_t, s_{t+1}, d_{t+1})$ in $\mathcal{D}$
        Sample random minibatch $(s_j, a_j, r_j, s_{j+1}, d_{j+1})$ from $\mathcal{D}$
        Compute $y_j = r_h + \gamma(1 - d_{t+1}) \max_{a'} Q'(s_{j+1}, a'; \theta')$
        Perform a gradient descent step of $(y_j - Q(s_j, a_j; \theta))^2$ with respect to $\theta$
        Every $C$ steps, reset target network parameters $\theta' = \theta$
    end for
end for
```

4 Experimental Results

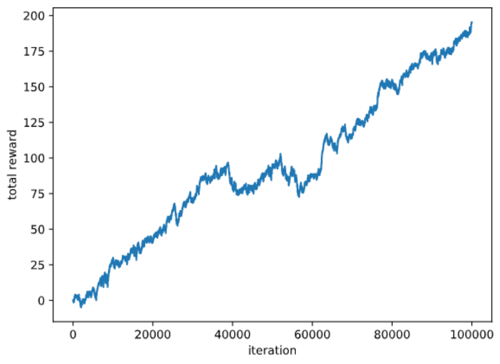
We train the model on two types of deep neural networks with two types of deep Q-learning algorithms separately and choose data from the year of 2015 for training and evaluate the trained stocks trading system on the data from the year of 2016.

The number of iterations is an important parameter during training. A too-small value will cause the model to not converge to the optimal value function while a too-large value will take too long time to train the model. In order to determine the number of iterations, firstly we set iteration to 100K and train a 3-layer fully connected neural network with naive deep Q-learning algorithm on the data from the year of 2015, and then evaluate the trained model on the data from the year of 2016. In Figure.2(b), we have some bad time as total rewards go down, but the overall results are good during validation, which means that our trained model does work. But in Figure.2(a), the total rewards during training doesn't increase, which means that our model hasn't converge. Besides, in a CPU, it trains at a speed of 50-60 iterations per second. Considering the training and validation results in Figure.2 and total training time, we choose the number of iterations to be 1M to train our system.

Figure.3 shows the validation results of two types of deep neural networks with two types of deep Q-learning algorithms. For deep Q-learning algorithm, a 4-layer 1D convolutional neural network receives a few better result. For deep duel Q-learning algorithm, a 3-layer fully connected neural network performs better. There is no obvious difference between the two type of neural networks, maybe due to the fact that the training data is not complex and these two neural networks all can powerfully extract sufficient feature and temporal information from the data. For the naive and duel deep Q-learning algorithms, the second performs better, which shows that decoupling the value and advantage function is advantageous to our task.



(a) training



(b) validation

Figure 2: total reward during training and validation with iteration=100K

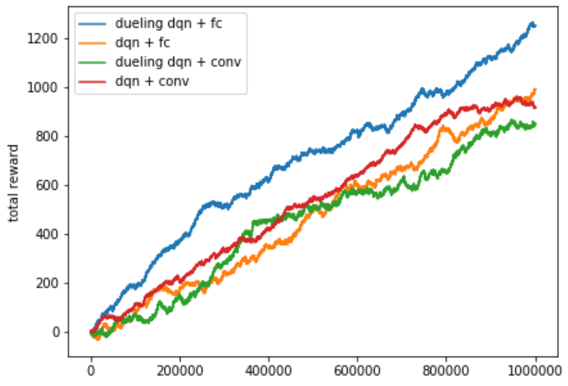


Figure 3: Validation result of four methods with iteration=1M

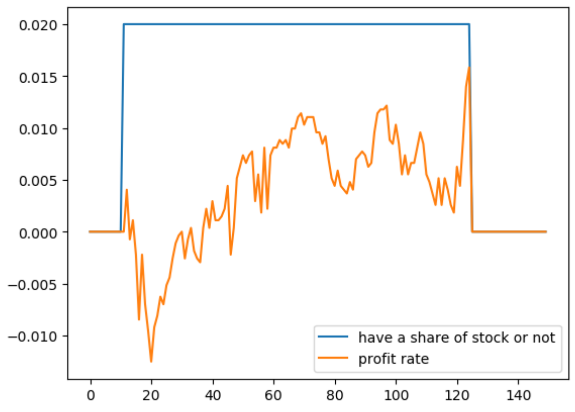


Figure 4: An example of trained fully connected model with dueling deep Q-learning

Figure.4 gives an example of the agent's action. The blue line represents the information of whether the agent has a share of stock currently or not and a positive value means the agent is holding a share of stock. So the rising edge in the blue line means the agent buy a share of stock, while the falling edge means the agent sell a share of stock. The orange line is the value of the profit rate currently relative to the purchase price. In this example, the agent buys a share of stock firstly, then waits until a high profit rate to sell the single share. This demonstrates that our trained stocks trading agent is able to make similar trading decisions as human — buy stock at low prices and sell it at high prices.

5 Summary

We build a stocks trading system with a deep reinforcement learning algorithm to decide to buy, hold, or sell stocks in a simulated stocks trading scene. We apply Deep Q-learning and Deep duel Q-learning algorithm, and try fully connected network and 1D convolutional neural network in this project. Experimental results show that our trained system can successfully learn to do stocks trading and keep making profits during validation.

References

[1] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... & Petersen, S. (2015). Human-level control through deep reinforcement learning. *Nature, 518(7540),* 529-533.

[2] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., ... & Dieleman, S. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature, 529(7587),* 484.

[3] Watkins, C. J. C. H. (1989). Learning from delayed rewards.

[4] Sutton, R. S., McAllester, D. A., Singh, S. P., & Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. *In Advances in neural information processing systems (pp. 1057-1063).*

[5] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., ... & Wierstra, D. (2015). Continuous control with deep reinforcement learning. *arXiv preprint* arXiv:1509.02971.

[6] Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., & Riedmiller, M. (2014, June). Deterministic policy gradient algorithms.

[7] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint* arXiv:1707.06347.

[8] Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., ... & Chen, Y. (2017). Mastering the game of go without human knowledge. *Nature, 550(7676),* 354-359.

[9] Heaton, J. B., Polson, N. G., & Witte, J. H. (2016). Deep learning for finance: deep portfolios. *Applied Stochastic Models in Business and Industry.*

[10] Bao, W., Yue, J., & Rao, Y. (2017). A deep learning framework for financial time series using stacked autoencoders and long- short term memory. *PLoS ONE,* 12(7).

[11] Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation, 9*(8), 1735-1780.

[12] Deng, Y., Bao, F., Kong, Y., Ren, Z., & Dai, Q. (2016). Deep direct reinforcement learning for financial signal representation and trading. *IEEE Transactions on Neural Networks and Learning Systems.*

[13] Shmuma. (2019). https://github.com/PacktPublishing/Deep-Reinforcement-Learning-Hands-On-Second-Edition/blob/master/Chapter10/lib/environ.py

[14] Wang, Z., Schaul, T., Hessel, M., Hasselt, H., Lanctot, M. & Freitas, N.. (2016). Dueling Network Architectures for Deep Reinforcement Learning. *Proceedings of The 33rd International Conference on Machine Learning, in PMLR 48*:1995-2003