Application of Deep Reinforcement Learning on Automated Stock Trading

Lin Chen
School of Electronic and Information Engineering
Beihang University
Beijing, China
chenlin769555861@buaa.edu.cn

Qiang Gao
School of Electronic and Information Engineering
Beihang University
Beijing, China
gaoqiang@buaa.edu.cn

***Abstract*—How to make right decisions in stock trading is a vital and challenging task for investors. Since deep reinforcement learning (DRL) has outperformed human beings in many fields such as playing Atari Games, can a DRL agent automatically make trading decisions and achieve long-term stable profits? In this paper, we try to solve this challenge by applying Deep Q-network (DQN) and Deep Recurrent Q-network (DRQN) in stock trading and try to build an end-to-end daily stock trading system which can decide to buy or to sell automatically at each trading day. The S&P500 ETF is selected as our trading asset and its daily trading data are used as the state of the trading environment. The agent's performance is evaluated by comparing with benchmarks of Buy and Hold (BH) and Random action-selected DQN trader. Experiment results show that our DQN trader outperforms the two benchmarks and DRQN trader is even better than DQN trader mainly because the recurrence framework can discover and exploit profitable patterns hidden in time-related sequence.**

***Keywords*—stock trading; MDP; POMDP; DQN; DRQN**

I. INTRODUCTION

How to make a profitable stock trading policy is significant for institutional and individual investors. Successful traders have their own trading systems and can make correct trading decisions based on either fundamental or technological analysis or both. However, for normal investors these analysis methods are usually hard to be taken into practice due to the information asymmetry of stock market, lack of trading experience and human weakness, etc[1]. Advances of reinforcement learning (Sutton and Barton 1998) can learn good policies for sequential decision problems and have outperformed human beings in many tasks such as playing Atari games[2]. So can we train an RL agent to learn from trading experiences and make profitable trading policy automatically? Compared with common RL tasks, stock trading has a few difficulties to imply RL due to the following reasons.

Firstly, how to define the state of stock trading environment is crucial for the performance of RL trader. In tasks like playing Atari games, the state of the environment is defined as the last four continuous game screens and is a full representation of the environment. However, unlike the aforementioned deterministic and noise-free environment, the stock prices are known as full of noises and trading environment is changing over time[3]. As a consequence, we need to define a state that satisfies both of the following criteria:

- The state needs to contain as much information of the factors that affect stock prices as possible.
- The state needs to contain less noise so the agent can learn right experience and is more likely to choose right action at each time step so that a positive long-term accumulative reward can be acquired.

Secondly, since stock prices change continuously and there is no boundary of the prices, the state space can be extremely large[4], so it is hard to give an accurate estimation of the value of each action at each state with a relative limited dataset.

To address the aforementioned two problems, we explore a deep reinforcement learning algorithm, namely Deep Q-Learning (DQN) , to find profitable patterns in the complex and dynamic stock trading environment and learn a better policy to achieve long-term accumulated profits. This algorithm consists of three key components: (i) a Deep Q-network that models the Q value-function of each pair of *state and action*; (ii) a target Q-network that improves the stability of the whole framework during training; (iii) experience replay that removes the correlation between samples and improve the utilization of them.

Because the state of trading environment is composed of a fixed number of latest continuous daily stock data, those useful information hidden in the data beyond this time window will be omitted, which makes the trading process more like a Partially Observable Markov Decision Process (POMDP)[5]. Thus, we replace the fully connected layer in our DQN with a recurrent LSTM layer of the same size, and manage to prove that introducing recurrence can improve the performance of the DQN agent in stock trading environment.

The remaining parts of this paper are organized as follows. Section II describes the deep Q-learning and deep recurrent Q-learning approach. Section III introduces the detailed implementations of the DQN trading model and its recurrent extension as well as the implemented stock trading algorithm. Section IV shows the experiment results of our model and compared the performance of DQN and DRQN trading model with the two baselines. Section V gives out the conclusion of our work and indicates some future directions.

978-1-7281-0945-9/19/$31.00©2019 IEEE

II. DQN AND DRQN LEARNING APPROACH

This section will introduce the vital parts of Deep Q-learning and its application on automated stock trading. Deep Recurrent Learning approach will also be introduced in this section.

A. Q-Learning

Reinforcement Learning (Sutton and Barton 1998) is a kind of machine learning algorithm that an agent try to learn and improve a policy through trials and rewards to acquire a maximum long-term accumulated reward in a certain task. The policy tells the agent which action to choose under each state of the environment in order to acquire a better accumulated reward. An reinforcement learning task is usually described as a Markov Decision Process (MDP), which can be represented by a tuple $\langle S, A, P, R, \gamma \rangle$. In this tuple, S is a finite set of states, A is a finite set of actions, P is a state transition probability matrix formulated as follows:

$$P_{ss'}^{a} = \mathbb{P}\left[S_{t+1} = s' \mid S_t = s, A_t = a \right] \quad (1)$$

R is a reward function formulated as follows:

$$R_s^a = \mathbb{E}\left[R_{t+1} \mid S_t = s, A_t = a \right] \quad (2)$$

γ is a discount factor and $\gamma \epsilon [0,1]$.

When the agent learns a policy π in a certain environment, it is actually learning the distribution over actions with given states:

$$\pi(a \mid s) = \mathbb{P}[A_t = a \mid S_t = s] \quad (3)$$

Given a policy π, the agent can know which action to choose under each state. According to the way to improve policy π, reinforcement learning can be divided into value-based RL and policy-based RL[6]. Value-based learning has a Q-function represents the expected return starting from state s, taking action a, and then following policy π. If the agent has learned an accurate Q-function of this environment, the agent can acquire the best accumulated reward by choosing the action with highest Q-value in each step. Compared with policy-based RL, value-based RL has shown excellent performance in tasks that have low dimensions and discrete action space. All of above characteristics make value-based RL qualified to solve financial trading problem of a single stock.

Q-learning (Watkins and Dayan 1992) is a value-based model-free off-policy algorithm for estimating. The core of Q-learning is a Q-table which represent the Q-value for each pair of state and action in the environment.

$$Q_\pi(s,a) = \mathbb{E}_\pi[R_t \mid S_t = s, A_t = a] \quad (4)$$

In formula (4), π is the decision policy, and R_t is the accumulated reward given by:

$$R_t = \Sigma_{t=0}^{+\infty} \gamma^t r_{t+1} \quad (5)$$

Here γ can be interpreted as the weight of the future rewards. if $\gamma = 1$, it represents that the agent pays attention to the future reward as the same as the immediate reward. On the contrary, if $\gamma = 0$, it represents that the agent only cares about the immediate reward. The agent improve its policy by the Bellman optimality equation:

$$Q^*(s,a) = \mathbb{E}\left[r + \gamma \max_{a'} Q^*(s', a') \right] \quad (6)$$

$$Q'(s,a) = Q(s,a) + \alpha(Q^*(s,a) - Q(s,a)) \quad (7)$$

By iteratively updating the Q-table, the Q-function will eventually converge to the optimal Q-function $Q^*(s,a)$[6].

B. Deep Q-learning

Deep Q-learning is Q-learning with the Q-table be replaced with a deep neural network. This approach aims to deal with the problem that when the state space and action space are continuous or infinitely discrete, it becomes impossible to iterate the Q-value in Q-table with finite samples. Since it has been proved that a neural network with 3 layers is able to express any function[7], the main idea of deep Q-learning is to use a neural network to approximate the Q-function.

The Q-network can be trained by minimizing the error between the improved Q-value given by formula (6) and the original Q-value. The loss function is formulated as follows:

$$L_t(\theta_t) = \mathbb{E}_{s,a,r,s'}\left[(y_t - Q_{\theta_t}(s,a))^2 \right] \quad (8)$$

where t is the current time step, and $y_t = r + \gamma \max_{a'} Q_{\theta_t}(s', a')$.

The loss function can be minimized by the stochastic gradient descend (SGD) algorithm. And the gradient is given by:

$$\nabla_{\theta_t} L_t(\theta_t) = \mathbb{E}_{s,a,r,s'}\left[(y_t - Q_{\theta_t}(s,a)) \nabla_{\theta_t} Q_{\theta_t}(s,a) \right] \quad (9)$$

C. Deep Recurrent Q-learning

In the DQN framework, there is a hidden suppose that the state we defined is a full observation of the environment at each time step. However, for stock trading environment, variables that exist in time dimension usually cannot be well discovered by the DQN framework.

To deal with this problem, we reference the Deep Recurrent Q-networks introduced by Hausknecht and Stone(2015). In the DRQN framework, h_t is the hidden state of the LSTM units and represent the information from the previous time step.

LSTM can discover the information hidden in temporal correlations and can keep the important features in previous state[8]. Thus, implementing LSTM in our DQN framework can make the state we defined more close to a full observation of the trading environment.

III. PROBLEM STATEMENT AND APPROACH

This section introduces how we model the stock trading problem into a MDP and our definition of the state, action and reward.

- State s: the state is defined as the daily adjusted close data in the last 20 days.

$$s_t = [p_{t-20}, p_{t-19}, \ldots\ldots, p_{t-2}, p_{t-1}] \qquad (10)$$

p_t represent the adjusted close price of day t.

- Action a: the action space only includes three available values [1,0,-1], representing buy, do nothing and sell for one share of stock respectively. Also, because we focus on daily trading process, we define that taking an action means trading one share of stock with the prices of the current day's adjusted close and will automatically close the position with the price of next day's adjusted close.
- Reward r: the reward is calculated by the difference between next day's adjusted close and the current day's adjusted close.

$$r_t = (p_{t+1} - p_t) \times action - value \qquad (11)$$

We firstly using a small set of training data to initialize the parameters of Q-networks. Then, at each trading day, the state is used as the input of the Q-network and the output are the Q-values for three different actions. The agent chooses the action with the highest value with a probability of $(1-\varepsilon)$ and will get a reward. Then, the trade sample $\langle s, a, r, s' \rangle$ will be stored in the buffer. Because the training daily dataset is rather small for a deep neural network and may cause overfitting. To enrich the data used for updating the Q-network parameters and make the updating more stable, we actually store three samples with three different rewards into the buffer. After doing this, the accumulated reward is calculated and a number of batch-size samples will be taken randomly from the buffer to update the Q-network parameters. The whole process will be carry forward till the end of the test data and an accumulated reward can be obtained.

Although deep Q-network can achieve effectively learning of the stock market features, there still have room for improvement. As it is known that the financial market is full of noise and uncertainty, and the factors influenced the stock price are multiple and will change over time. This makes the stock trading process more like a Partially Observable Markov Decision Process (POMDP) because the state we used is not equal to the real state of the stock trading environment. Thus, we try to make use of the memory characteristic of LSTM to discover the features that change over time, and we want to explore that given the same state as the input of the DQN or DRQN, LSTM can integrate the information hidden in time-dimension and thus possible to make the POMDP more close to a MDP[9].

Although deep Q-learning performs well on many tasks, one well-known drawback of deep Q-learning is that it is quite unstable, especially in the noisy and time-varying stock trading environment. So we apply the two famous approaches[2] to make the DQN more stable.

One approach is to use a separate target network $\tilde{Q}$ to compute the improved Q given by the Bellman equation, formulation (6) is then changed to:

$$Q(s,a) = r + \gamma \max_{a'} \tilde{Q}(s', a') \qquad (12)$$

we implement the approach introduced in [2] to update the parameters of target network $\tilde{Q}$, that is updating the target network every 100 iterations with the parameter of the original Q-network, so that the target network $\tilde{Q}$ will update more slowly than the original Q-network and be more stable.

The algorithm is as follows:

Algorithm1 Deep Q-learning with Experience Replay

```
Initialize replay memory $D$ to capacity $N$
Initialize network with random weights $\theta^Q$
Initialize target network with weights $\theta^{Q'} \leftarrow \theta^Q$
for episode = 1, $M$ do
  for t = 1, $T$ do
    With probability $\varepsilon$ select a random action $a_t$
    otherwise select $a_t = \max Q(s', a'; \theta)$
    Execute action $a_t$ and observe reward $r_t$
    Set $(s_t, a_t, r_t, s_{t+1})$, store trade sample in $D$
    Sample a random mini-batch of transitions from $D$
    Set $y_j = r_j + \gamma \max Q(s_{t+1}, a_{t+1}; \theta)$
    Perform a gradient descent step on $(y_j - Q(s_j, a_j; \theta))^2$
    according to equation 9
    Update the target network weights:
        $\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$
  end for
end for
```

IV. Experiment Results

Our deep Q-network and deep recurrent Q-network is evaluated on SPY(the S&P500 ETF) and the data are downloaded from Yahoo finance. The training and testing dataset is as follows:

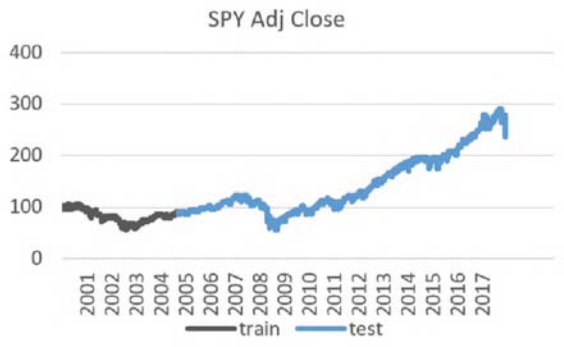

Fig. 1. The Adjusted Close of SPY

A. Data and settings

The data used in the experiment involves 19 years ranging from 01/01/2000 through 12/31/2018. The databased is divided into a training dataset (01/01/2000-12/31/2004) and a test dataset (01/01/2005-12/31/2018). Only the daily closing price is used in this study.

We firstly used the training data to initialize the deep Q-network and its recurrent version by ten iterations, then the performance of the models are tested on the test dataset. The structure of the basic Q-network is as follows:

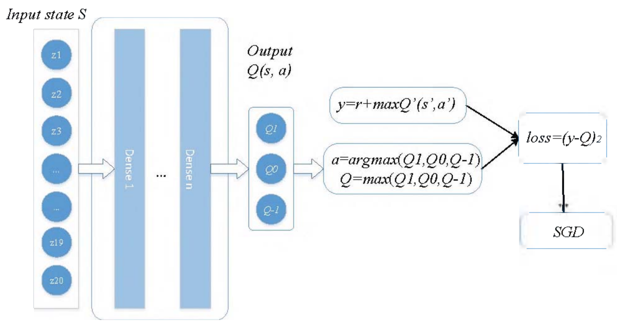

Fig. 2. The Structure of Deep Q-network

The deep Q-network contains 4 layers and the number of units for each layer is 20,10,10 and 3 respectively. We use ReLU as the activation function. The learning rate is set to be exponential decayed every 100 steps on the training dataset and the initial rate is 0.001, while on the test dataset, the learning rate is set as a fixed value 0.001. For the Q-learning part, the discount factor is set to $\gamma = 0.79$, this value is quite small because the problem we defined is a short-term daily trading process so that we pay more attention to current reward. The max ε is set to 0.99 and start with 0. The batch size is set to 100 and the parameters for target network $\tilde{Q}$ is replaced every 100 iterations. As for the DRQN model, the time step is set to 20, and the number of RNN units is set to 6 .

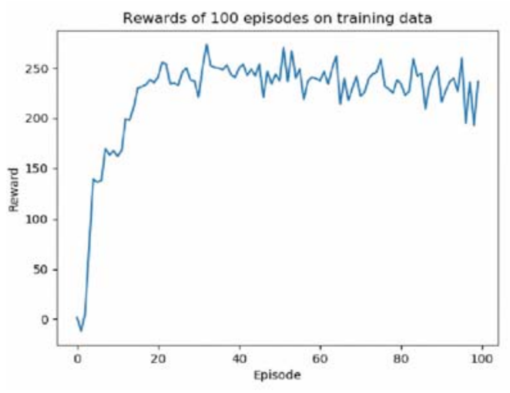

Fig.3. Accumulated Rewards of DQN of the first 100 episodes on the SPY Train Dataset

In this paper we use two baselines for our model, which are buy and hold and random DQN. The former means to buy one share of stock on the first day of the training dataset and hold the stock till the end. And the random DQN trader means at each time-step, the agent chooses the action randomly regardless of the output Q-value for each actions. The parameters are set as same as the DQN trader.

To verify that a deep structure will bring improvement for the DQN trader, we also explore the performance of DQN with different number of layers.

B. Experimental results

a) Discount Factor: We explore the performance of the DQN with different layers and the results are as follows. Table1 shows the length of reward window for different discount factors.

TABLE I. THE CORRESPONDING LENGTH OF REWARD WINDOW FOR DIFFERENT DISCOUNT FACTORS

γ	***0.46***	***0.63***	***0.79***	***0.84***
length of reward window	3	5	10	20

TABLE II. THE AVERAGE ACCUMULATED REWARD OF DIFFERENT DQN MODELS

γ	**Number of layers of DQN**			
	Zero	*One*	*Two*	*Three*
0.46	-29.6	144.6	165.6	-89.5
0.63	-21.7	160.2	260.6	-82.9
0.79	6.4	193.3	**295.0**	-23.2
0.84	-46.1	213.0	209.9	-101.6

From the result in table 2 we can conclude that for our daily trading problem, the discount factor 0.79 is the most appropriate one.

b) Learning Rate: Learning rate is a vital factor that affect the performance of the model. We compare the performance of different learning rate in Table 3.

TABLE III. DIFFERENT LEARNING RATE AND THE CORRESPONDING ACCUMULATED REWARD

learning rate	***0.01***	***0.001***	***exp decay***	***our learning rate set***
DQN with 2 layers	-34.3	193.3	215.1	**260.6**

c)Performance of DQN and DRQN: Table 4 shows the performance of DQN and DRQN on the test dataset. By comparison with the two baselines, we can see that the deep Q-learning approach can discover the features automatically and make right decisions at most times to acquire an accumulated reward better than the BH strategy.

TABLE IV. AVERAGE ACCUMULATED REWARD OF DOFFERENT MODELS

	Performance Comparison			
	BH	*Random DQN*	*DQN*	*DRQN*
Average Total Profits	159.4	5.0	**285.8**	**338.8**

In Fig.4, we can see that the DQN and DRQN model outperform the Buy and Hold baseline even if the SPY has a long term of rising trend. As we can see from Table 4, the DQN strategy achieves an annualized return of 22.33%, which is much higher than Dow Jones Industrial Average's 12.25% and the DRQN strategy achieves a higher annualized return of 23.48%. The results demonstrate that DQN and DRQN can effectively learn a profitable strategy from history data and DRQN even enjoys a better performance than DQN.

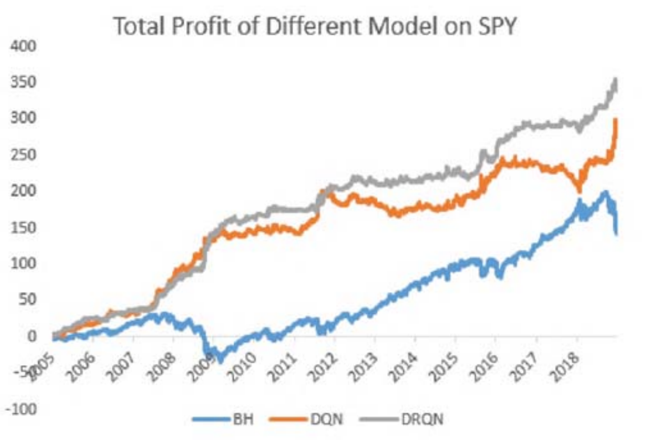

Fig.4. Total Profit Curves of BH, DQN and DRQN on the SPY Test Dataset

V. Conclution

In this paper we explore the performance of deep Q-learning and deep recurrent Q-learning for stock trading. The results show that deep Q-network can learn profitable patterns from raw stock trading data and utilize them to achieve high accumulated reward. Because the stock trading environment is more close to a POMDP, we also observe that introducing recurrence into the Q-learning can bring performance improvement in the stock trading task. The result shows that DRQN is an alternative method to bring systematic improvement.

In the future, we will explore the implementation of other reinforcement learning approaches on stock trading task. Trading portfolios of multiple financial asset[10] is also an optional directions to explore.

References

[1] Philip Treleaven, Michal Galas, and Vidhi Lalchand, "Algorithmic trading review," Communications of the ACM, vol. 56, no. 11, pp. 76-85, 2013.

[2] Mnih, Volodymyr, et al. "Human-level control through deep reinforcement learning." Nature 518.7540 (2015): 529.

[3] Deng, Yue, et al. "Deep direct reinforcement learning for financial signal representation and trading." IEEE transactions on neural networks and learning systems 28.3 (2016): 653-664.

[4] Xiong, Zhuoran, et al. "Practical deep reinforcement learning approach for stock trading." arXiv preprint arXiv:1811.07522 (2018).

[5] Hausknecht, Matthew, and Peter Stone. "Deep recurrent q-learning for partially observable mdps." 2015 AAAI Fall Symposium Series. 2015.

[6] Sutton, Richard S., and Andrew G. Barto. Introduction to reinforcement learning. Vol. 135. Cambridge: MIT press, 1998.John Moody, Lizhong Wu, Yuansong Liao, and Matthew Safell, "Performance functions and reinforcement learning for trading systems and portfolios," Journal of Forecasting, vol. 17, no. 56, pp. 441-470, 1998..

[7] Hornik, Kurt, Maxwell Stinchcombe, and Halbert White. "Multilayer feedforward networks are universal approximators." Neural networks 2.5 (1989): 359-366.

[8] Gers, Felix A., Jürgen Schmidhuber, and Fred Cummins. "Learning to forget: Continual prediction with LSTM." (1999): 850-855.

[9] Lample, Guillaume, and Devendra Singh Chaplot. "Playing FPS games with deep reinforcement learning." Thirty-First AAAI Conference on Artificial Intelligence. 2017.

[10] Jae Won Lee, Jonghun Park, O Jangmin, Jongwoo Lee, and Euyseok Hong, "A multiagent approach to q-learning for daily stock trading," IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans, vol. 37, no. 6, pp. 864-877, 2007.