Building Stock Trading Model with Deep Reinforcement Learning Algorithm ELENE6885 Course Project (Paper)

Zhibo Dai [1] **Zhenrui Chen** [1] **Tianyu Yang** [1] **Jiawei Chen** [1]

Abstract

A profitable automated stock trading model is very important in the financial sector, but in the complex and changeable stock market, designing a profit strategy is challenging. Everyone wants a winning strategy, but it is not easy. This project designs an automated trading solution for a single stock transaction. We used 30 component stock data from the Yahoo Finance API including opening, high, and low closing prices and trading volume to model the stock trading process as a Markov Decision Process (MDP). Then, we define our trading goal as a maximization problem using Deep Deterministic Policy Gradient (DDPG) algorithm and Advantage Actor Critic(A2C) for training. The performance of the trading agent with different reinforcement learning algorithms is evaluated and compared with both the Dow Jones Industrial Average index and the traditional min-variance portfolio allocation strategy. The proposed deep ensemble strategy is shown to outperform the three individual algorithms and two baselines in terms of the risk-adjusted return measured by the Sharpe ratio.

1. Introduction

Machine learning mainly involves building predictive models based on data. When the data is sequentially arranged, models will be able to predict their sequence and results too. One of the applications of machine learning is to predict the operation of the stock market in recent years. But it is proved to be extremely difficult, as so many factors are involved in the prediction. In the past, what machine learning mainly tried to do in the financial market was to predict the future return of financial assets through supervised learning such as artificial neural networks, support vector machines and even decision trees. However, many methods did not work out well for varied reasons. For example, labeled data sets with balanced class distributions are normally used in supervised machine learning. When it comes to the stock market, there is no such labeled data guiding the time window of buying or selling holdings. For questions like this, reinforcement-learning frameworks are more fitted, which is a behavior-based learning relying on tried experiments and supplemented by rewarding mechanisms. Once appropriate reward signals are defined, reinforcement learning will be able to generate this missing label. But there remain other issues that are specific to the stock market under such circumstances.

The stock exchange market changes at a high frequency, and these changes cannot be inferred from historical trends alone. They are affected by political, social, environmental and other factors in the real world. In this case, the signal-to-noise ratio will be very high, a fact makes it difficult to learn meaningful things. This type of environment can be modeled as a partially observable Markov decision process, where the agent enjoys limited degree of visibility to all environmental conditions. In modeling the agent's decision-making process, it is assumed that the system is determined by a stochastic control process in discrete time, but the agent cannot observe its basic state directly. Combining opinion mining of trading companies and related news, and in combination of reinforcement learning algorithms, the system learns appropriate strategies to trade the stocks of specified companies.

2. Background/related Work

Emerged at a rapid rate in the financial field, with the development of artificial intelligence. Reinforcement learning is the application of quantitative investment at the forefront under the rapid development of artificial intelligence, and can pick out trends embedded in the complex financial environment. It is capable of selecting actions based on public information in the environment by exploring and taking advantage of the mechanism of reinforcement learning, and find out a strategy that can obtain the greatest return through learning. The quantitative trading strategy proposed on the basis of reinforcement learning extended the application of deep reinforcement learning in the financial market, and also serves as an insightful idea for investors to obtain better returns from financial investments.

Reinforcement learning works by learning through interactions with environment, where tried experiments are made on a continuous basis in order to optimize gradually, and problems are solved through the Markov decision process. Strategies are learnt in the process of interacting with environment, and returns are maximized accordingly. Traditional reinforcement learning adopts the iterative Bellman equation to solve the value function. The cost of calculation will be too much when the state space is too large, and normally linear function approximators are used to represent the value function in approximation. In comparison, deep reinforcement learning uses a Deep Neural Network as a nonlinear function approximator to approximate a value function or strategy, and in this way it combines the perception ability of deep learning and the decision-making ability of reinforcement learning .

Due to its huge success in the field of games, deep reinforcement learning has attracted wide attention from the financial community. In traditional portfolio management, investors need to master a lot of knowledge relevant to financial investment, which constitutes challenges for the majority of investors. With the usage of deep reinforcement learning, there is no need for investors have too much knowledge of any specific field, except for basic trading rules. First of all, Pendharkar P C and others constructed an individual retirement portfolio management that contains two assets using traditional reinforcement learning methods. Also, they used SARSA(λ) and Q(λ) for discrete state and discrete action, and online TD (λ) algorithm for discrete state and continuous action. It is shown that continuous action reinforcement learning agents always perform the best on portfolio allocation (Pendharkar & Cusatis, 2018).

Later, more and more people followed suit by applying reinforcement learning to portfolio management. Among which, Zhengyao Jiang and others proposed a model-free reinforcement learning framework, and then conducted portfolio management in the crypto-currency market. Experimental results show that this framework is capable of achieving at least fourfold of returns within 50 days (Jiang et al., 2017); Lili Tang et al. proposed a model-based actor-critic algorithm in an uncertain environment, which realizes stable investment and thus obtains stable growth in return (Tang, 2018); Lin Li et al. used recursive reinforcement learning for direct portfolio selection, which can outperform some of the latest portfolio selection methods (Li, 2019); Yuh Jong Hu et al. used the GRU network and risk-adjusted return function, together with reinforcement learning to solve the strategy optimization problem of investment portfolio (Hu & Lin, 2019).

However, most of these studies are based on data from foreign countries, and very few studies are done on Chinese stock market. With respect to the domestic stock market, Tu Shenhao et al put forward a series of model to improve and optimize the traditional mean-variance model, in view of the shortcomings of portfolio theory, and the deep reinforcement learning model is introduced to study market timing . Xiong, Zhuoran, et al et al. used DDPG algorithm in deep reinforcement learning to invest in stock market by limiting the investment weight of a single stock, and the portfolio value was significantly higher than the average investment strategy (Xiong et al., 2018).

3. Algorithm Development

Considering the action space is continuous in the stock trading problem, we choose to employ the Deep Deterministic Policy Gradient (DDPG) algorithm and Advantage Actor-Critic(A2C) algorithm to build the trading model. Besides the implementation, we also compare the performance of our model with the automated stock trading model built-in FinRL (Liu et al., 2020).

3.1. Advantage Actor Critic

Asynchronous Advantage Actor-Critic(A3C) was introduced by Google Deep Mind in a paper proposing many asynchronous methods for DRL(Mnih et al., 2016). The main idea of the algorithm is to use a critic updated using the advantage function and an actor who shares the parameters with the critic. The advantage function provides a baseline to evaluate the quality of the state-action pair. In this framework, the critic approximates the advantage function and corrects the actor based on it, thus reducing the policy gradient's variance and improving the stability. In A3C, the critic approximates the value function while the actor maintains a policy. After a certain number of steps, the critic updates the actor. The researcher adds some noise to the loss function, preventing the agent from converging very early and getting stuck in the local optimum.

However, a researcher from open AI claims that there are no notable benefits of introducing noise due to the asynchronous, and they propose the synchronous version of A3C, which is the A2C. In the A2C algorithm, all the parameters will be updated and broadcast to all workers simultaneously when all the agents have finished. The parameters are updated by averaging the gradient of all agents.

In A2C algorithm, there are two main components, an actor learns the policy and a critic which learns a value function to evaluate the state-action pair, which provides a reinforcing signal to the actor. *Actor*: actors learn the policy π_θ using the gradient:

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_t[A_t^\pi \nabla_\theta log\pi_\theta(a_t|s_t)]$$

critic: the critics evaluate the state-action pair and use it to generate the advantage function.

The advantage function, in general, measures the quality of an action, and the advantage is equal to the TD error. The Advantage function tells us if a state is better or worse than expected. If an action is better than expected (the advantage is greater than 0), we want to encourage the actor to take more of that action. If an action is worse than expected (the advantage is less than 0), we want to encourage the actor to take the opposite of that action. If an action performs exactly as expected (the advantage equals 0), the actor doesn't learn anything from that action. With the actor and critic, we can build a simple A2C model, the implementation is summarized in algorithm 1.

Algorithm 1 Advantage Actor Critic

```
Set actor learning rate $\alpha_A$, critic learning rate $\alpha_C$,;
Randomly initialize the actor and critic parameters $\theta_A$, $\theta_C$;
for episode = 1, $M$ do
    Gather and store date by acting in the environment using the current policy;
    for $t$ = 1, $t$ do
        Calculate predicted V-value $V^{\pi}(s_t)$ using the critic network $\theta_C$;
        Calculate predicted V-value $A^{\pi}(s_t, a_t)$ using the critic network $\theta_C$;
        Calculate $V^{\pi}_{tar}(s_t)$ using the critic network $\theta_C$;
    end for
    Calculate value loss and policy loss using MSE:
        $$L_{val}(\theta_C) = \frac{1}{T}\sum_{t=0}^{T}(V^{\pi}(s_t) - V^{\pi}_{tar}(s_t))^2$$
        $$L_{pol}(\theta_A) = \frac{1}{T}(-A^{\pi}(s_t, a_t) log \pi_{\theta_A}(a_t|s_t))$$
    Update actor and critic parameters using SGD
        $$\theta_C \leftarrow \theta_C + \alpha_C \nabla_{\theta_C} L_{val}(\theta_C)$$
        $$\theta_A \leftarrow \theta_A + \alpha_A \nabla_{\theta_A} L_{val}(\theta_A)$$
end for
```

3.2. Deep Deterministic Policy Gradient

We also employ the Deep Deterministic Policy Gradient(DDPG) algorithm to formula the stock trading strategy. DDPG is developed based on the Deterministic Policy Gradient(DPG) algorithm, and it combines both Q-learning and policy gradient methods. DDPG uses off-policy data and the Bellman equation to learn the Q-function and uses the Q-function to learn the policy. In Q-learning, we know that:

$$\pi^*(s) = arg\max_a(Q^*(s, a))$$

In a stock trading problem, the action space is continuous, we cannot exhaustively evaluate every action and then solve this equation, but because the action space is continuous, the function $Q^*(s, a)$ is Differentiable, thus instead of computing $\max_a Q(s, a)$ every time, we approximate it with

$$\max_a Q(s, a) \approx Q(s, \mu(s))$$

where μ is the policy.

Q-learning of DDPG: In Bellman equation we know that:

$$Q^*(s, a) = E_{s'}[r(s, a, r') + \gamma \max_{a'} Q^*(s', a')]$$

In DDPG, we use a neural network to approximate the value function, and we can set up the mean-squared Bellman error function(MSBE). The Q-learning is performed by minimizing the MSBE loss with the gradient method. Thus just like in DQN, we need to use the experience replay and target network technique to reduce the data correlation.

Experience Replay: In DDPG, we are trying to approximate a complex value function $Q(s, a)$ with a neural network, and the error function is minimized by SDG. However, to use SGD optimization, the training data should be independent and identically distributed. To solve this problem, Google Deep Mind introduced the experience replay technique in DQN. In their work, instead of updating the Q value network following the most recent experience, they create an experience pool and update the network using the data randomly chosen from the experience pool, which is very similar to the memory process in nature. In programming, we can do this by using a large experience buffer and sample training data from it. This technique prevents our model from overfitting and improves the stock prediction accuracy.

Target Network: In Q-learning, the normal update method is to update the value based on the most recent experience. On the one hand, the normal strategy will cause correlation. On the other hand, the training network depends on the same parameters used in the target. Thus a small change can have a significant influence on the network, which will cause the training to unstable. To solve this, a target network is introduced. We create target network parameters. The parameters are not updated every time step but periodically synchronized with the parameters of the main Q-network, thus improving the stability of the training.

Policy learning of DDPG: We want to learn a deterministic policy that maximizes return, and as we mentioned above, the action space is continuous. Thus we can take the derivative of the objective function with respect to the policy parameter, and since the update is based a batch of memory, we can take the average of the sum of gradients calculated from the mini-batch.
The process of DDPG is summarized in Algorithm 2.

Algorithm 2 Deep Deterministic Policy Gradient

```
Randomly initialize critic network $Q(s, a|\theta^Q)$, and actor $\mu(s|\theta^\mu)$;
Set target parameters: $\theta^{Q'}_{targ} \leftarrow \theta^Q$, $\theta^{\mu'}_{targ} \leftarrow \theta^\mu$;
Initialize the replay buffer $\mathcal{D}$;
for episode = 1, $M$ do
    Initialize a random process $\mathcal{N}$ for action exploration;
    Receive initial observation $s_1$;
    for $t$ = 1, $t$ do
        Select action according to the current policy and exploration noise;
        Execute action and observe the reward and $s'$
        Store transition $(s, a, r, s')$ in $\mathcal{D}$
        Sample a random batch of N transitions from $\mathcal{D}$
        Update main network by minimizing the loss:
            $$L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^\mu)^2$$
        where
            $$y_i = r_i + \gamma Q'(s', \mu'(s'|\theta^{\mu'}|\theta^{Q'}))$$
        update the policy using the policy gradient
            $$\nabla \approx \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s)} \nabla(s|\theta^\mu)|_s$$
        Update the target network:
            $$\phi_{targ} \leftarrow \rho\phi_{targ} + (1-\tau)\phi$$
            $$\theta_{targ} \leftarrow \rho\theta_{targ} + (1-\tau)\theta$$
    end for
end for
```

4. Experiment results

4.1. Data Preprocessing

To collect the stock data, firstly, we download the data from Yahoo Finance API, and then we select the data and discard the missed value, after that, we can generate the needed parameters from the dataset, which includes boll upper bound(bool_ub), bool lower bound(bool_lb), The relative strength index(RSI_30), Simple Moving Average(close_30_sma, close_60_sma) and Commodity Channel Index(cci_30) , all of them will be the input for our training model.

After the data processing, we can then use the historical daily prices from 2016-01-01 to 2021-12-17 to train the agent and test the performance. Our experiment consists of two stages, training and trading. In the trading stage, we use data from 2016-01-01 to 2020-07-30 to generate a well-trained agent, and in the trading stage, we test our agent's performance on the trading data, which is from 2020-08-01 to 2021-12-17. Our model was built on the ElegantRL framework, and the file structure is as shown in Fig 1.

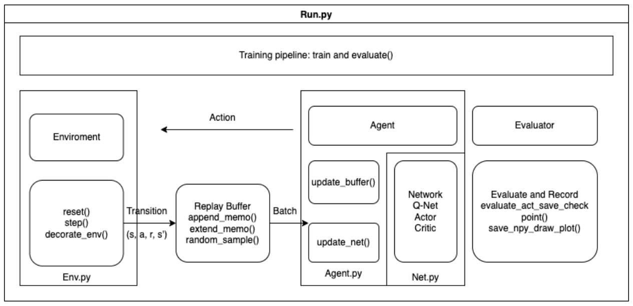


Figure 1. File structure of the training environment

4.2. Experimental Setting and Results of Stock Trading

MDP model of stock trading

- State s = [p, h, b]: a vector that includes stock prices, the stock shares, and the remaining balance..
- a vector of actions over all stocks. The allowed actions on each stock include selling, buying, or holding, which result in decreasing, increasing, and no change of the stock shares, respectively.
- Reward $r(r, s, s')$:the direct reward of taking action a at state s and arriving at the new state $s\prime$.
- Policy $\pi(s)$: the trading strategy at state s, which is the probability distribution of actions at state s.

- Q-value $Q_\pi(s, a)$: the expected reward of taking action a at state s following policy π.

At time t an action is taken and the stock prices update at $t + 1$, accordingly the portfolio values may change from ”portfolio value 0” to ”portfolio value 1” (Buy), ”portfolio value 2” (Hold), or ”portfolio value 3” (Sell), respectively.

We define our reward function as the change of the portfolio value when action a is taken at state s and arriving at new state s + 1. The goal is to design a trading strategy that maximizes the change of the portfolio value $r(s_t, a_t, s_{t+1})$ in the dynamic environment, and we employ the deep reinforcement learning method to solve this problem.

Using this model, we implement both A2C and DDPG algorithm to predict the stock tendency, the performance of both algorithm is as shown in table 1.

Table 1. Trading Performance of A2C and DDPG

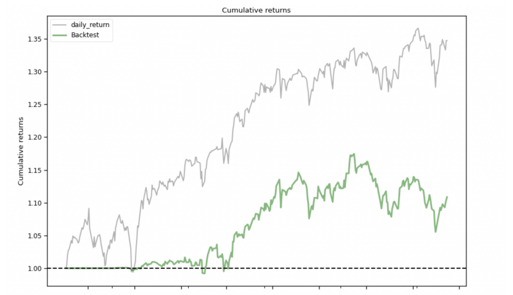
Performance	A2C	DDPG
Annual return	7.753%	23.133%
Cumulative returns	10.862%	33.291%
Annual volatility	10.21%	13.7647%
Sharpe ratio	0.78	1.59

To measure the performance of our DDPG model, we also implemented the model from the ElegantRL, RLlib, and Stablebaselines3 library. We compared the annual return, cumulative return, annual volatility, and Sharpe ratio. The result is as shown in table 2. From the table, we can see that the result of our model is very close the the model from outside library, which indicates our implementation is correct and meaningful.

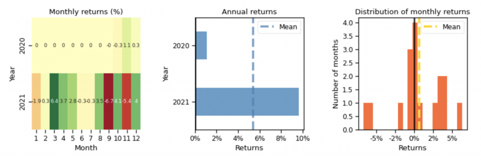
Table 2. Trading Performance of A2C and DDPG

Performance	ElegantRL	RLlib	SL3
Annual return	25.878 %	22.32%	25.763%
Cumulative returns	37.413%	32.078%	37.239%
Annual volatility	13.548 %	16.883 %	15.951 %
Sharpe ratio	1.77	1.28	1.53

The accumulative return and Monthly and Annually return plot of A2C algorithm as shown in Fig 2;



(a) Accumulative return of A2C

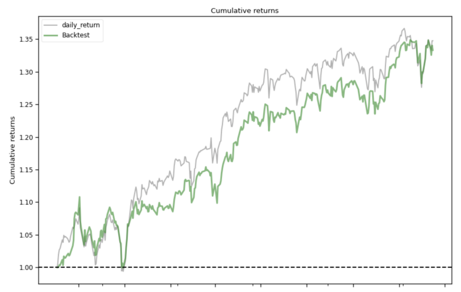


(b) Monthly and Annually return of A2C

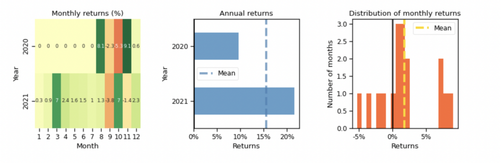
Figure 2. Return of A2C model

From the plot, we can see that the A2C algorithm didn't perform very well, the accumulative return did't improve a lot after training, which also provides us the reason the use a more efficient algorithm, the DDPG.

The accumulative return and Monthly and Annually return plot of A2C algorithm as shown in Fig 3.



(a) Accumulative return of DDPG



(b) Monthly and Annually return of DDPG

Figure 3. Return of DDPG model

We also compared the monthly and annually return of our DDPG model with the model from outside library shown in Fig 4;

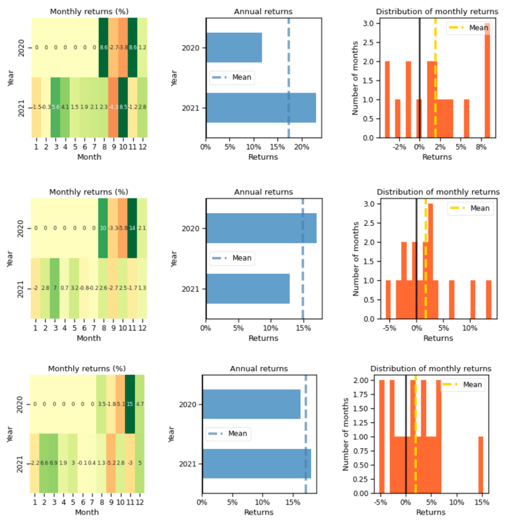


Figure 4. Return of ELRL, RLlib, Stb3 DDPG moderespectively

The accumulative return comparison of our model and the model from ElegantRL, RLlib, and Stablebaselines3 library is as shown in Fig 5, from the plot we can see that our result is very close to the outside library, and the performance

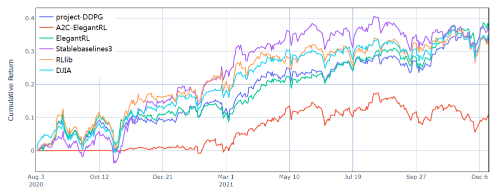


Figure 5. Performance comparison

if better than stb3 library. In a word, our DDPG model is meaningful and can do the stock trading problem just as we expected.

5. Conclusion

In this paper, we explore the potential of training A2C and DDPG to learn stock trading strategies. The results demonstrate that we can get 7.753 % annual return from the A2C model and 23.133 % annual return from the DDPG model. We also compared our result with the ElegantRL, RLlib, and Stablebaseline, which indicates our model is better than the model from RLlib. Also, the comparison on the sharpe ratio indicates our method is robust and efficient.

A statement on the contributions of each member of the team.

Paper writing: Jiawei Chen and Tianyu Yang; Algorithm building: Zhibo Dai and Zhenrui Chen;

References

Hu, Y.-J. and Lin, S.-J. Deep reinforcement learning for optimizing finance portfolio management. In *2019 Amity International Conference on Artificial Intelligence (AICAI)*, pp. 14–20. IEEE, 2019.

Jiang, Z., Xu, D., and Liang, J. A deep reinforcement learning framework for the financial portfolio management problem. *arXiv preprint arXiv:1706.10059*, 2017.

Li, L. Direct portfolio selection using recurrent reinforcement learning. *Available at SSRN 3479973*, 2019.

Liu, X.-Y., Yang, H., Chen, Q., Zhang, R., Yang, L., Xiao, B., and Wang, C. D. Finrl: A deep reinforcement learning library for automated stock trading in quantitative finance. *arXiv preprint arXiv:2011.09607*, 2020.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pp. 1928–1937. PMLR, 2016.

Pendharkar, P. C. and Cusatis, P. Trading financial indices with reinforcement learning agents. *Expert Systems with Applications*, 103:1–13, 2018.

Tang, L. An actor-critic-based portfolio investment method inspired by benefit-risk optimization. *Journal of Algorithms & Computational Technology*, 12(4):351–360, 2018.

Xiong, Z., Liu, X.-Y., Zhong, S., Yang, H., and Walid, A. Practical deep reinforcement learning approach for stock trading. *arXiv preprint arXiv:1811.07522*, 2018.