COMP-767 Final Project Report

Deep Reinforcement Learning in Financial Portfolio Management

Hang Li
hang.li3@mail.mcgill.ca
260850432

Zhaoqi Xu
zhaoqi.xu@mail.mcgill.ca
260563752

Zhiguo Zhang
zhiguo.zhang@mail.mcgill.ca
260550226

I. Introduction

Reinforcement learning techniques have raised attention from financial industry, especially by employing reinforcement learning in portfolio managements. In this project, we explored three state-of-art reinforcement learning algorithms, including policy gradient (PG), deep deterministic policy gradient (DDPG) and proximal policy optimization (PPO). The goal is to train an intelligent agent that can continuously trade in stock market by allocating on different assets. In the course of this project, we performed hyper-parameter tuning and feature selection for each algorithm. Furthermore, we compared the return performance of different algorithms using both U.S. and China stock market data. In the following sections of this paper, we will formally define the portfolio management problem as a Markov decision process, describe our detailed methodology, and demonstrate our experimental comparison and results[1].

II. Related Work

Deep reinforcement learning is the combination of reinforcement learning (RL) and deep learning. Reinforcement learning refers to goal-oriented algorithms, which learn how to attain a complex objective or maximize along a particular dimension over many steps. Like a child incentivized by candy, agents trained by these algorithms are penalized when they make the wrong decisions and rewarded when they make the right ones. RL algorithms that incorporate with deep learning techniques can accomplish a wide range of more complex tasks that were previously infeasible for a machine. They can even beat world champions at the game of Go [1] as well as human experts playing Atari video games.

During the last decade, a variety of reinforcement learning methods have been attempted to solve the asset allocation problem. Du et al. investigated Q-learning and recurrent reinforcement learning (RRL) on simple CAPM model. The agent decides the investment weights on risky market portfolio and risk-free asset. They considered interval profit, Sharpe ratio and derivative Sharpe ratio as value functions. Recurrent reinforcement learning achieves better stability comparing with Q-learning [2]. Almahdi et al. extended previous work on recurrent reinforcement learning and constructed asset allocation based on the expected maximum drawdown. The improved RRL model with Calmar ratio, outperforms the previous RRL model with Sharpe ratio and Sterling ratio. It also showed better return performance than hedge fund benchmarks[3]. With the development of deep learning, Jiang et al. proposed a portfolio management reinforcement learning framework to utilize deep learning solutions. The framework consists of Convolutional Neural Network (CNN), Recurrent Neural Network (RNN), and a Long Short-Term Memory (LSTM) [4]. Tang proposed actor-critic based reinforcement learning method on portfolio management. The optimal investment is achieved by dynamic planning [5]. Liang et al. applied deep deterministic policy gradient (DDPG), proximal policy optimization(PPO) and adversarial policy gradient to portfolio management problem [6].

III. Problem Formulation

Portfolio management is essentially how to allocate given fund on several risky assets and risk-free assets. For simplicity, we assume risk-free asset is cash instead of treasury bill in real life. Moreover, we only consider non-negative weight on each asset, i.e. short-selling or loan from bank is not allowed. The stock market is assumed to be continuous. That is the closing price today equals to the opening price tomorrow. Based on the observed market state, our agent reallocates the portfolio weights at the end of each day. The transaction cost is considered when the agent tries to adjust the weights.

Portfolio management problem can be modeled as a Markov decision process, where tuple $(S, A, P, r, \rho_0, \gamma)$ represents the problem space. The set of states S consists of states that describe the previous open, closing, low, high price and other financial indices including stock volume, P/E ratio, P/B ratio and so on. The set of actions A is formed by the allocating weight vector a_t at time t, where $\sum_{i=0}^{M} a_{i,t} = 1$ with $M+1$ assets. $P : S \times A \times S \to \mathbb{R}$ denotes the transition probability distribution. $r : S \times A \to \mathbb{R}$ is the reward function. ρ_0 is the distribution of initial state s_0. γ is the discount factor. We also define v_t as the closing price at time t and $y_t = v_t / v_{t-1}$ as the price fluctuation vector.

Let $\odot$ be the element-wise vector multiplication. Then, the weight vector a_t is updated by the following formula at the end of each day:

$$a_t = \frac{a_{t-1} \odot y_{t-1}}{a_{t-1} \cdot y_{t-1}}.$$

[1]Code for this project can be found at **https://github.com/ZhaoqiXu/comp-767-final-project**

We define the transaction cost is proportion to the amount of transaction, $\mu \sum_{i=0}^{M} |a_{i,t-1} - a_{i,t}|$ with $\mu = 0.25\%$ in our experiment. The fluctuation of wealth at time $t-1$ is $a_{t-1} \cdot y_{t-1}$. We notice that the conventional reinforcement learning objective function takes form of $R_t = \sum \gamma^t r(s_t, a_t)$. However, in the context of portfolio management, the wealth at time T is calculated by $R_T = \prod_{t=0}^{T} R_0 r(s_t, a_t)$ [6]. As a result, we take logarithm on the change in wealth and convert the product into summation form. Therefore, the reward at time $t-1$ is defined as

$$r(s_{t-1}, a_{t-1}) = \log(a_{t-1} \cdot y_{t-1} - \mu \sum_{i=0}^{M} |a_{i,t-1} - a_{i,t}|)$$

IV. Methodology

In our project, we implemented three algorithms, Policy Gradient (PG), Deep Deterministic Policy Gradient (DDPG) [7] and Proximal Policy Optimization (PPO) [8], to find optimal assets allocation strategies for high return. The PG algorithm has been well explained in class so we will mainly focus on the latter two algorithms. Both DDPG and PPO can be classified as Deep Reinforcement Learning category.

Deep reinforcement learning has three strengths compared with merely using reinforcement learning or deep learning only in portfolio management problem. First of all, stock performance can be affected by various factors, such as economic growth, interest rates, market confidence and expectations and so on, which make stock performance predictions inaccurate. Fortunately, deep reinforcement learning does not explicitly involve predictions of stock performance. Secondly, function approximation using neural network can bring the flexibility of design specific neural network structure and overcome the limitations of number of states and actions that has been shown as an obstacle for conventional reinforcement learning. Moreover, the portfolio management problem can be well defined as a RL-friendly problem, in which the market information is defined as input and the allocating vector is defined as output. This does not require any investment expertise, and still achieves self-improvement.

The rest of the section will demonstrate the details of two deep reinforcement learning algorithms, PPO and DDPG.

A. Actor Critic Methods

Both DDPG and PPO are built on the Actor Critic model. Actor Critic model [9] is a "hybrid method" combining value based methods and policy based methods. It consists of two neural networks. A Critic measures how good the action taken is, whereas an Actor controls how our agent behaves.

The problem of conventional policy gradient method is that reward can only be calculated by the end of episode. High reward at the end may indicate that actions in this trajectory are good, even if some are not. Consequently, the convergence is slow. The Actor Critic model has a better score function. Instead of waiting until the end of each episode, it updates at each step similar to TD learning. To do an update at each step, we need to train a Critic model which approximates the value function. This value function replaces the reward function in PG.

At the beginning, the Actor has very limited knowledge about the task and takes some actions randomly. The Critic observes the action and provides some feedback. By learning from the feedback, the Actor improves its policy accordingly. Meanwhile, the Critic also updates its approximation to provide better feedback. Therefore, the idea of Actor Critic is to make use of two neural networks, where the estimation of both run in parallel. Furthermore, since there are two sets of parameters, they must be optimized separately.

B. Proximal Policy Optimization

The main idea of Proximal Policy Optimization is to avoid significant policy update. To achieve this, PPO computes a ratio that indicates the difference between the new and old policy, and clips this ratio between 0.8 and 1.2. This ensures that policy could not update significantly. Furthermore, PPO introduces a new trick, that is training the agent by running K epochs of gradient descent over sampling mini batches. PPO is based on Advantage Actor Critic (A2C).

The idea of PG is to encourage the agent to take actions that lead to higher rewards and avoid bad actions. However, if the step size is too small, the training process is too slow, while if it is too large, there is too much variability during the training. Fortunately, PPO improves the stability of the Actor training by limiting the policy update at each training step. PPO introduces a new objective function called Clipped Surrogate Objective function which constraints the policy change in a small range by using a clip.

1) Clipped Surrogate Objective: The Clipped Surrogate Objective is a replacement for the policy gradient objective function that is designed to improve the training stability by limiting the change one update could make at each step. The objective function,

$$L^{PG}(\theta) = \hat{\mathbb{E}}_t \left[\log \pi_\theta \left(a_t | s_t \right) \hat{A}_t \right]$$

is commonly used to optimize the neural network in vanilla PG. $\hat{A}_t$ is often replaced by the discounted return. By taking a gradient ascent step on this loss with respect to the network parameters, the actions that led to higher reward are chosen.

However, instead of using the log probability of the action to evaluate the impact of them, there are also other similar functions introduced in [10]. They use the probability of the action under the current policy, divided by the probability of the action under the previous policy as the following:

$$r_t(\theta) = \frac{\pi_\theta \left(a_t | s_t \right)}{\pi_{\theta old} \left(a_t | s_t \right)}.$$

If $r_t(\theta) > 1$, it means that the action is more probable in the current policy than the old one. If $0 < r_t(\theta) < 1$, it means that the action is less probable for current policy than for the old one. Replacing this with the log probability, we get a new objective function

$$L^{TRPO}(\theta) = \hat{\mathbb{E}}_t \left[\frac{\pi_\theta \left(a_t | s_t \right)}{\pi_{\theta old} \left(a_t | s_t \right)} \hat{A}_t \right] = \hat{\mathbb{E}}_t \left[r_t(\theta) \hat{A}_t \right]$$

and this is what is done in the TRPO method [11]. Note that if the action is much more probable for the current policy, $r_t(\theta)$ can be very big, which means that the agent is going to take big gradient steps here. To deal with this issue, the TRPO method applies some extra techniques (e.g., KL Divergence constraints) to limit the policy update step.

What PPO does is one step further. Instead of adding the extra techniques, PPO directly build these properties into the objective function. Hence, Clipped Surrogate Object is defined as

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t\left[\min\left(r_t(\theta)\hat{A}_t, \operatorname{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t\right)\right].$$

The first term inside the minimization is the same term we saw in the TRPO objective. The second term is a version where the $r_t(\theta)$ is clipped between $(1-\epsilon, 1+\epsilon)$. Finally, the minimization of both of these terms is taken. The clipping limits the effective change of one update at each step in order to improve stability. More details are discussed in the original paper. The pseudocode of PPO is attached below.

```
Algorithm 2 PPO
 1: Initialize actor $\mu : S \to \mathbb{R}^{m+1}$ and
    $\sigma : S \to diag(\sigma_1, \sigma_2, \cdots, \sigma_{m+1})$
 2: for $i = 1$ to M do
 3:     Run policy $\pi_\theta \sim N(\mu(s), \sigma(s))$ for T timesteps and
        collect $(s_t, a_t, r_t)$
 4:     Estimate advantages $\hat{A}_t = \sum_{t'>t} \gamma^{t'-t} r_{t'} - V(s_t)$
 5:     Update old policy $\pi_{old} \leftarrow \pi_\theta$
 6:     for $j = 1$ to N do
 7:         Update actor policy by policy gradient:
                $\sum_i \nabla_\theta L_i^{CLIP}(\theta)$
 8:         Update critic by:
                $\nabla L(\phi) = -\sum_{t=1}^{T} \nabla \hat{A}_t^2$
 9:     end for
10: end for
```

Fig. 1. PPO Pseudo Code

C. Deep Deterministic Policy Gradient

Deep Deterministic Policy Gradient is an algorithm which concurrently learns a Q-function and a policy. It has four networks: online actor, online critic, target actor and target critic. Actor is the current policy and critic estimates the value of the current policy. When the agent observes a state, the online actor provides an optimal action in continuous action space. Then the online critic evaluates the actor's choice and updates online actor. Moreover, the target actor and target critic are used to update online critic. DDPG uses off-policy data and the Bellman equation to learn the Q-function, and uses the Q-function to learn the policy.

This approach follows the same idea as Q-learning: if we have the optimal action-value function $Q^*(s,a)$, then in any given state, the optimal action $a^*(s)$ is found by solving

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

One of the reason that we choose DDPG over naive Q-learning is that DDPG is adapted specifically for environments with continuous action spaces. When there are finite number of discrete actions, we can just compute the Q-values for each action separately and directly compare them. However when the action space is continuous, we cannot exhaustively evaluate the action space and solving the optimal problem is non-trivial. The good news is that because the action space is continuous, the function $Q^*(s,a)$ is expected to be differentiable and this allows us to set up a gradient-based learning rule for a policy $\mu(s)$. Therefore, to compute $\max_a Q(s,a)$, we can approximate it with $\max_a Q(s,a) \approx Q(s,\mu(s))$ instead of using a normal optimization algorithm.

DDPG learns the Q-function by minimizing a mean-squared Bellman error (MSBE) loss function. The Bellman equation about the optimal action-value function is given by

$$Q^*(s,a) = \underset{s'\sim P}{\mathrm{E}}\left[r(s,a) + \gamma \max_{a'} Q^*\left(s', a'\right)\right]$$

where s' is sampled by the environment from a distribution $P(\cdot|s,a)$.

Suppose the approximator is a neural network $Q_\phi(s,a)$, with parameters ϕ, and that we have collected a set D of transitions (s,a,r,s',d). We can use a MSBE function to evaluate how closely Q_ϕ comes to satisfying the Bellman equation:

$$L(\phi, \mathcal{D}) =$$
$$\underset{(s,a,r,s)\sim D}{\mathrm{E}}\left[\left(Q_\phi(s,a) - \left(r + \gamma(1-d)\max_{a'} Q_\phi\left(s', a'\right)\right)\right)^2\right]$$

Thus, the function approximator is derived by minimizing the loss function above.

It is worth to mention that there are two tricks used in the implementation of DDPG.

1) Replay Buffer: DDPG uses experience replay to update neural network parameters. During each trajectory roll-out, we save all the experience tuples $(state, action, reward, nextState)$ and store them in a finite sized cache - a "replay buffer". Then, we sample random mini-batches of experience from the replay buffer when we update the value and policy networks. As a result, the data is more independently distributed.

2) Target Networks: Q-learning algorithms make use of target networks. The term

$$r + \gamma(1-d)\max_{a'} Q_\phi\left(s', a'\right)$$

is called target, because we want the Q-function to be as close as to this target. One issue of the target network is that it uses the same parameters that we are trying to train. This causes the MSBE converging slowly and unstable. To overcome this issue, we use a second network with a time delay of the first one. The second network is called target network.

Recalling the goal of this algorithm is to learn a deterministic policy $\mu_\theta(s)$ which gives that action that maximizes $Q_\phi(s,a)$. Because the action space is continuous, we perform gradient ascent to solve

$$\max_\theta \underset{s\sim\mathcal{D}}{\mathrm{E}}\left[Q_\phi\left(s,\mu_\theta(s)\right)\right].$$

Taking derivative of the object function is equivalent to take derivative of policy. Formally, the update scheme for the online actor is

$$\nabla_{\theta\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q\left(s,a|\theta^Q\right)\big|_{s=s_i,a=\mu(s_i)} \nabla_{\theta^\mu}\mu\left(s|\theta^\mu\right)\big|_{s_i}.$$

Another issue for DDPG is that it seldom performs exploration for actions. A solution for this is adding noise on the parameter space or the action space. Figure 2 is the pseudocode of the DDPG.

Algorithm 1 DDPG

```
 1: Randomly initialize actor $\mu(s|\theta^\mu)$ and critic $Q(s,a|\theta^Q)$
 2: Create $Q'$ and $\mu'$ by $\theta^{Q'} \to \theta^Q$,$\theta^{\mu'} \to \theta^\mu$
 3: Initialize replay buffer $R$
 4: for $i = 1$ to M do
 5:     Initialize a UO process $\mathcal{N}$
 6:     Receive initial observation state $s_1$
 7:     for $t = 1$ to T do
 8:         Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$
 9:         Execute action $a_t$ and observe $r_t$ and $s_{t+1}$
10:         Save transition ($s_t$,$a_t$,$r_t$,$s_{t+1}$) in $R$
11:         Sample a random minibatch of N transitions
            ($s_i$,$a_i$,$r_i$,$s_{i+1}$) in $R$
12:         Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
13:         Update critic by minimizing the loss:$L = \frac{1}{N}\sum_i(y_i - Q(s_i,a_i|\theta^Q))^2$
14:         Update actor policy by policy gradient:
            $\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_{\theta^\mu} Q(s,a|\theta^Q)|_{s=s_t,a=\mu(s_t|\theta^\mu)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s_t}$
15:         Update the target networks:
                $\theta^{Q'} \to \tau\theta^Q + (1-\tau)\theta^{Q'}$
                $\theta^{\mu'} \to \tau\theta^\mu + (1-\tau)\theta^{\mu'}$
16:     end for
17: end for
```

Fig. 2. DDPG Pseudo Code

V. Experiments

We conducted extensive analysis of the models based on various hyperparameters such as the learning rate of neural network and features combination. We evaluate the performances based on the average daily return (ARR), Sharpe Ratio (SR) and maximum drawdown(MMD).

A. Dataset

Our experiment not only use stock data in U.S. like most of previous works, but also involve more volatile China stock market. Our experimental data sources are investing[2] and Wind[3]. We select a set of stocks with lower or even negative correlation between each other for the two stock markets in order to demonstrate the ability of RL agents in different assets. We choose the open price, close price, the highest price and the lowest price as the feature space. In the China stock market, we selected 16 stocks to test the performance of RL agents on large-scale asset allocation issues. In the U.S. stock market, we selected 5 stocks. In addition, we select time period from 1/1/2015 to 12/31/2016 as our training dataset and from 1/1 2017 to 1/1/2018 as our backtest period(testing dataset).

The stock information is shown in Table I.

market	code	market	code
China	000725	U.S.	AAPL
China	000002	U.S.	ADBE
China	600000	U.S.	BABA
China	000862	U.S.	SNE
China	600662	U.S.	V
China	002066		
China	600326		
China	000011		
China	600698		
China	600679		
China	600821		
China	600876		
China	600821		
China	000151		
China	000985		
China	600962		

TABLE I
Stock code information

B. Network structure

For finite historical time dependency, which assumes current optimal portfolio only depends on finite N historical stock prices, CNN can be a good choice. Different from previous work [12], we alternate standard CNN with Deep Residual Network.

As we know, increasing network depth by simply stacking layers together may not achieve satisfying results. Deep networks are hard to train because of the notorious vanishing gradient problem. Repeated multiplication may make the gradient infinitely small. Consequently, as the network goes deeper, its performance gets saturated or even starts degrading rapidly. Deep residual network solves this problem by adding a shortcut for layers to jump to the deeper layers directly, which could prevent the network from deteriorating as the depth adds. Deep Residual Network has gained remarkable performance in

[2]https://lpi.invest.com/invest.com&bittrex
[3]http://www.wind.com.cn/

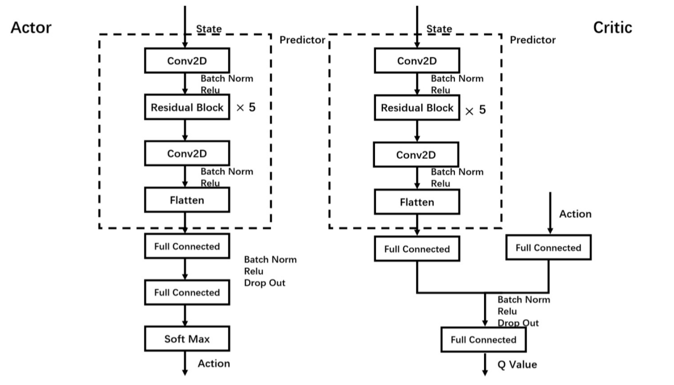

Fig. 3. DDPG Network Structure in our experiments

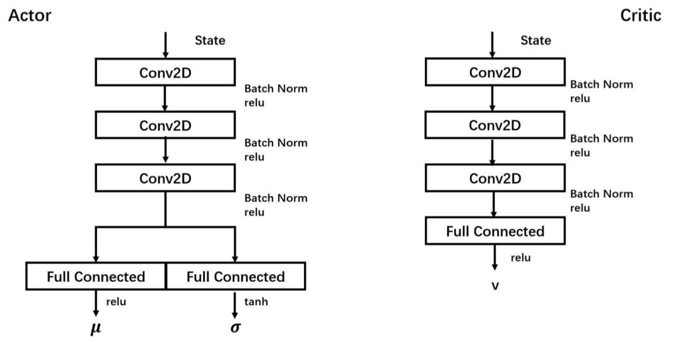

Fig. 4. PPO Network Structure in our experiments

image recognition and greatly contributes to the development of deep learning[13].

The Actor and Critic network structure of DDPG method in our experiments is shown in Fig.3. The PPO network structure is shown in Fig.4

C. Result

1) learning rate: The learning rate is a configurable hyper-parameter used in the training of neural networks that has a small positive value, often in the range between 0.0 and 1.0. The learning rate controls the rate or speed at which the model learns. Specifically, it controls the amount of apportioned error that the weights of the model are updated with each time they are updated, such as at the end of each batch of training examples.

Therefore, we implement PG and DDPG, and test them using different learning rates on U.S. stock market. Fig.5 shows the PG loss under different learning rates. Fig.6 shows the Critic loss under different critic learning rates. These results show that learning rates have significant effect on loss. A learning rate that is too large or too small will greatly influence the convergence of the model. Based on the experimental results, we found that learning rates in the range of 0.0001 and 0.001 generally ensure the convergence

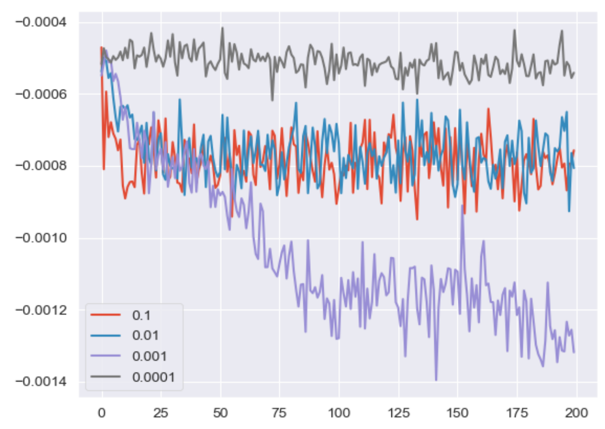

Fig. 5. PG loss under different learning rates

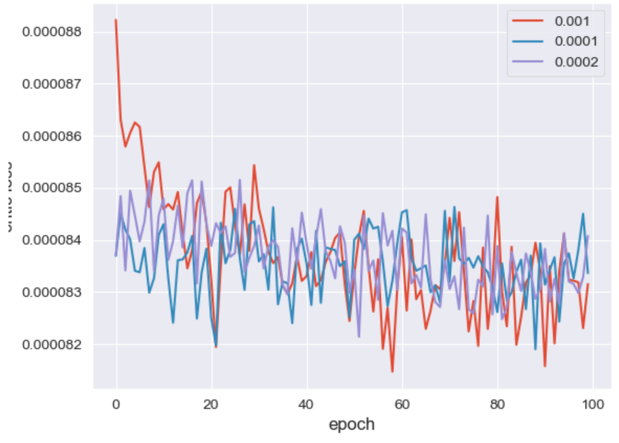

Fig. 6. Critic loss under different critic learning rates.

of our model. Furthermore, we notice that loss fluctuates with a decreasing trend due to Actor Critic model.

2) feature combination: In contrast with robot control whose input is pixels, abundant features can be taken into considerations in portfolio management. A good feature combination is crucial to deep learning agent, so we should select those features which contribute most to our task.

Thus, we implement PG and PPO, and test them using different feature combinations: 1. only with closing prices, 2. with closing and the highest price, 3. with closing and the lowest price. Fig.7 shows the reward result of PG on China stock market during the training process with three feature combination. Fig.8 shows the reward result of PPO on U.S. stock market. From these result, we can see that feature combinations matter in the training process. By selecting closing and the highest price, the agent shows the best performance

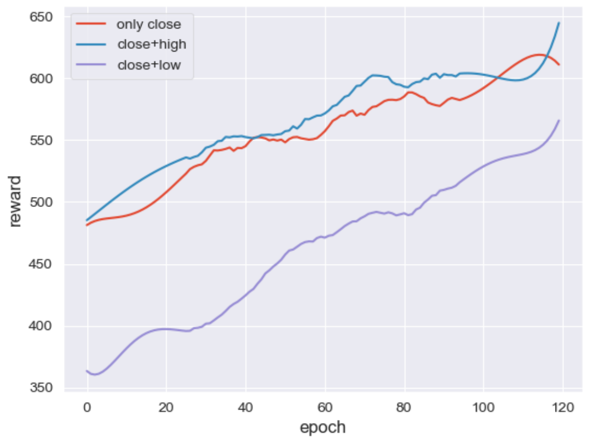

Fig. 7. Comparison of PG reward with different features combinations

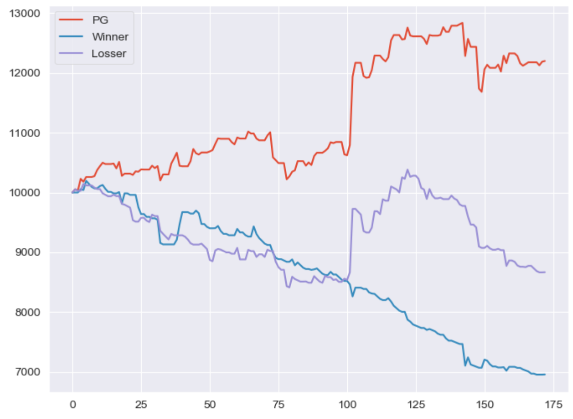

Fig. 9. Backtest of PG in U.S. Stock Market

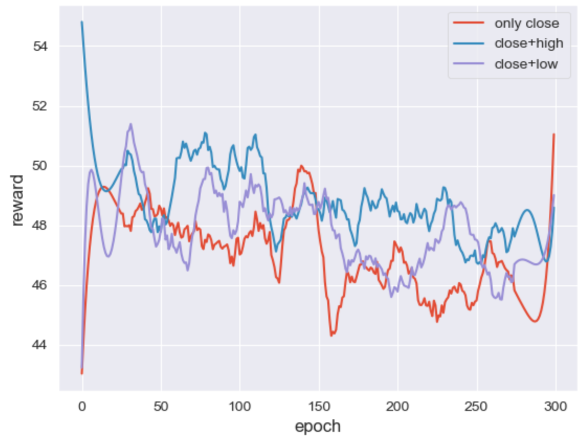

Fig. 8. Comparison of PPO reward with different features combinations

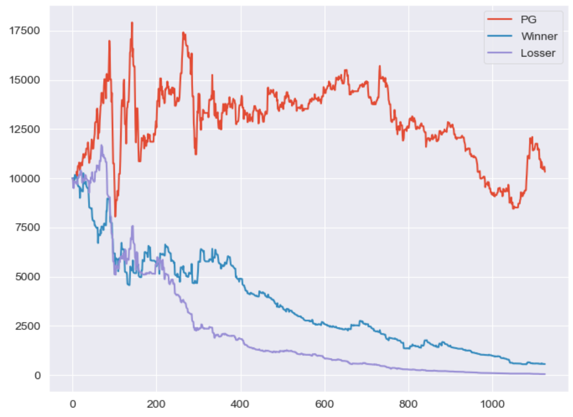

Fig. 10. Backtest of PG in China Stock Market

in our experiments.

3) compare and analysis: Under satisfying set of hyper parameters and feature combination, we conduct training for 100 epochs for PG method in both U.S. annd China stock market, the backtest result shows in Fig.9 and Fig.10. In each Figure, `Winner` means each action the stock trader took is based on the highest closing price of the previous day. `Losser` means each action the stock trader took is based on the lowest closing price of the previous day.

We also conduct training for 1000 epochs for DDPG method on China and U.S. stock market. The backtest results show in Fig.11 and Fig.12. These results demonstrates that RL agents could increase accumulative portfolio value and perform well in stock market. Our PG and DDPG agents generally have better performance than traditional trading strategies, i.e. `Winner` and `Losser`. For PG agent, we observe that it performs better in the U.S. stock market than in the China stock market. However, for DDPG agent, the result is opposite. This observation implies that the difference between two markets in terms of capacity and volatility should be considered in the future work.

VI. Conclusion

In this project, we have formulated the financial portfolio management problem into a deep reinforcement learning problem. We successfully implemented the PG, DDPG and PPO models and trained them on China and U.S. stock market. The experiments show that the strategy obtained by PG algorithm can outperform `Winner` and `Losser` in assets allocation. The stock data is noisy and many external factors other than historical prices can influence the price trend, which limits the performance of our other model.

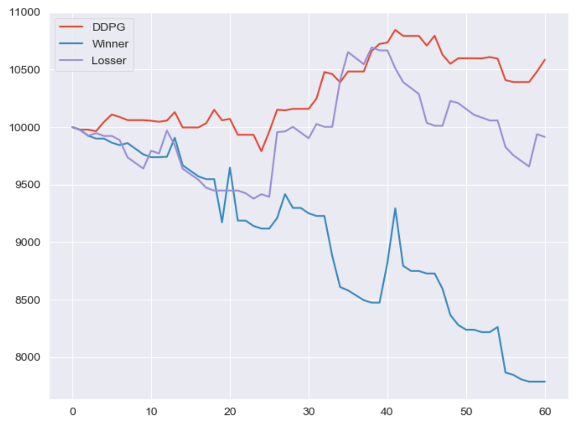

Fig. 11. Backtest of DDPG in China Stock Market

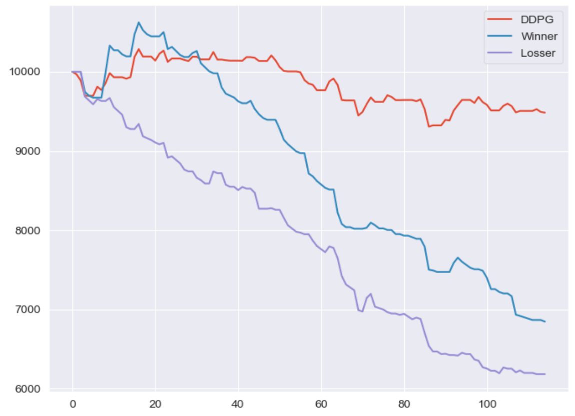

Fig. 12. Backtest of DDPG in U.S. Stock Market

References

[1] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, #jan# 2016.

[2] Xin Du, Jinjian Zhai, and Koupin Lv. Algorithm trading using q-learning and recurrent reinforcement learning. *1*, 2009.

[3] Saud Almahdi and Steve Y. Yang. An adaptive portfolio trading system: A risk-return portfolio optimization using recurrent reinforcement learning with expected maximum drawdown. *Expert Systems with Applications*, 87:267 – 279, 2017.

[4] Zhengyao Jiang, Dixing Xu, and Jinjun Liang. A deep reinforcement learning framework for the financial portfolio management problem, 2017.

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

[6] Zhipeng Liang, Kangkang Jiang, Hao Chen, Junhao Zhu, and Yanran Li. Deep reinforcement learning in portfolio management. *CoRR*, abs/1808.09940, 2018.

[7] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In Eric P. Xing and Tony Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning*, number 1 in Proceedings of Machine Learning Research, pages 387–395, Bejing, China, 22–24 Jun 2014. PMLR.

[8] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[9] Vijaymohan Konda. *Actor-critic Algorithms*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 2002. AAI0804543.

[10] Sham Kakade and John Langford. Approximately optimal approximate reinforcement learning. In *Proceedings of the Nineteenth International Conference on Machine Learning*, ICML '02, pages 267–274, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc.

[11] John Schulman, Sergey Levine, Philipp Moritz, Michael Jordan, and Pieter Abbeel. Trust region policy optimization. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pages 1889–1897. JMLR.org, 2015.

[12] Zhengyao Jiang, Dixing Xu, and Jinjun Liang. A deep reinforcement learning framework for the financial portfolio management problem. *arXiv preprint arXiv:1706.10059*, 2017.

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.