CS221 Project Final Report
Deep Reinforcement Learning in Portfolio Management

Ruohan Zhan rhzhan@stanford.edu
Tianchang He th7@stanford.edu
Yunpo Li yunpoli@stanford.edu

Abstract

Portfolio management is a financial problem where an agent constantly redistributes some resource in a set of assets in order to maximize the return. In this project, we wish to apply the methodology of deep reinforcement learning to the problem with the help of Deterministic Policy Gradient (PDG). We designed and developed models using Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN). The performances are evaluated and compared with several benchmarks and baselines.

1. Introduction

Portfolio management is a multi-armed bandit problem where an agent makes decision on continuously reallocating some resources into a number of different financial investment assets to maximize the return [1]. Traditional portfolio management algorithms are usually prior-constructed with little machine learning elements [3][2]. The recent advance of deep learning has seen many attempts to solve the portfolio management problem with deep learning techniques. Some try to predict price movements for the next period based on history prices [4][5]. The performances of these algorithms depends highly on the degree of prediction accuracy given that the market is difficult to predict.

Previous attempts are model-free and regression-based, but we can also try to pose the problem as a Reinforcement Learning (RL) one where the agent allocates the resources and collects reward at the next step. There has been previous works to utilize deep RL to solve the problem [6][7]. They used novel architectures such as Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN) with Long Short Term Memory (LSTM) cell to predict action (a distribution denoting the allocation) instead of price trends. [7] in particular achieved 8-fold return in a period of about 50 days with a step size of 30 minutes with cryptocurrency market data. However, some failed to replicate their results [8].

In this project, we wish to tackle this problem with data from a number of S&P 500 companies. We designed and implemented CNN and RNN models with different hyperparameters and compared the performances with an oracle, a baseline algorithm (Eigen-portfolio) and other benchmarks including *Best Stock* and Uniform Buy And Hold (UBAH).

2. Problem Statement

2.1. Assumptions

Throughout the project we are making two basic assumptions on the behavior of the stock market:

i. Negligible Market Impact: our action has minimum impact on the stock market. In other words, the stock prices are given as input data and should not be affected by the agent's actions.

ii. Continuity of Stock Unit: instead of discrete stock number, we assume that the agent can trade continuous amount of stocks (e.g., 3.5 stocks). This is just a simplification of the more complicated integer-based model.

2.2. State-based Model for the Portfolio Management Problem

In this project, we frame the portfolio management problem as a state-based model in order to use reinforcement learning. Here, we give the definition of our states, actions, rewards and policy:

2.2.1 States

A state contains historical stock prices and the previous time step's portfolio. Consider a stock market with L different stocks, we limit the portfolio management action only happens at discrete time steps $\{0, T, 2T, 3T, 4T, \dots\}$. At each time step kT, the total assets are M^k, and we assume the agent use M_k plus this time step's return as the new assets at the next time step. The stock prices are:

$$S^k = (S_1^k, \dots, S_L^k). \tag{1}$$

The portfolio is

$$P^k = (P_0^k, P_1^k, \dots, P_L^k) \tag{2}$$

where $\sum_{i=0}^{L} P_i^k = 1$, P_0^k is the percentage of reserved capital, $P_j^k \geq 0, j = 1, \dots, L$ is the percentage of capital allocated into each stock. We can express the states as:

$$\text{State}^k = (S^{k-N}, S^{k-N+1}, \dots, S^{k-1}, S^k, P^{k-1}) \tag{3}$$

where N can be a fixed finite number, which means we use the windowed prices in states. We can also let $N \to +\infty$, then recurrent neural network could be applied to this problem.

2.2.2 Actions

Actions are defined as the portfolio P^k we choose at each time step. According to formula (2), we are setting each percentage P_j^k in continuous space, while the summation of all percentages should be one.

2.2.3 Rewards

We define the reward for each $((S^{k-N}, S^{k-N+1}, \dots, S^{k-1}, S^k, P^{k-1}), P^k)$ pair to be the return of investment following portfolio P^k. The return is determined by profit and transaction cost together. If we follow P^k at time step kT, the corresponding profit is:

$$r^k = \sum_{i=1}^{L} \frac{M^k P_i^k}{S_i^k} (S_i^{k+1} - S_i^k) \tag{4}$$

The transaction cost is defined in terms of two parts, the cost proportional to the change of stock assets, and the cost due to the change of portfolio:

$$C^k = \sum_{i=1}^{L} c_i \left| \frac{M^k P_i^k}{S_i^k} - \frac{M^{k-1} P_i^{k-1}}{S_i^{k-1}} \right| S_i^k + c_0 \mathbf{1}_{[P^k \neq P^{k-1}]} M^k \tag{5}$$

where c_i is the cost ratio of change in stock i, and c_0 is the cost ratio of change in portfolio.
Now the return of following P^k at kT can be expressed as:

$$R^k = \frac{r^k - C^k}{M^k} = \sum_{i=1}^{L} \frac{P_i^k}{S_i^k}(S_i^{k+1} - S_i^k) - \sum_{i=1}^{L} c_i \left| \frac{P_i^k}{S_i^k} - \frac{M^{k-1} P_i^{k-1}}{M_k S_i^{k-1}} \right| S_i^k - c_0 \mathbf{1}_{[P^k \neq P^{k-1}]} \tag{6}$$

2.2.4 Policy

For the deep reinforcement learning methodology, we use our learned deep neural networks as the policy. Given the states, we use these networks to extract features and give corresponding actions. In the following section, we will talk about how the neural networks were trained to maximize the reward at each time step.

3. Dataset

We obtain the prices of 150 stocks in S&P 500 from Yahoo Finance within the period of 3 years (from 2014/10/22 to 2017/10/22) as our data and divided them into training, validation and testing. The time interval for trading date selection is one week. We choose 10 stocks (AAPL, ADSK, EBAY, FB, GOOGL, INTC, INTU, NFLX, ORCL, SYMC) to show model performance in testing data, while training and validation can depend on the whole 150 stocks.

4. Methodology

In our project, we wish to use neural networks to predict the action (portfolio) given current state (historical stock price and previous portfolio). At each time step, we have a new state. Based on the policy, we give action (portfolio). Then at next time step, we can evaluate our portfolio based on the return.

The network **input** comes from our states defined in (3). Normalization is necessary here due to the ratio property of return. We chose log return + previous portfolio as out input.

$$\text{Input}^k = (\log \frac{S^{k-N+1}}{S^{k-N}}, \ldots, \log \frac{S^k}{S^{k-1}}, P^{k-1}) = (x^k, P^{k-1}) \tag{7}$$

where N can be a fixed finite number, which means we use the windowed historical prices in states.

The loss is defined as negative return, with the intuition that the policy should be modified in the direction which increases each time period's return. We use Adam optimizer to train the neural network.

4.1. DPG

In traditional RL algorithms such as Q-learning or TD-learning we would estimate the value or utility based on different actions.

Suppose we have policy π. The reward we are going to collect at the next state s' is $r(s, \pi(s), s')$. The expected reward from a stochastic policy is

$$\hat{r}(s, \pi(s), s') = \sum_a p_\pi(s, a, s') r(s, a, s') \tag{8}$$

However when action is in continuous space the curse of dimensionality dominates - the total number of actions increases exponentially as the discretization precision increases. It was proposed by [9][10] to throw away the stochasticity and produce an deterministic action for continuous space action and then evaluate and update parameters accordingly. Hence at each time step we can just produce one action based on policy π and make further updates based on it.

In our case, we use neural networks to produce the action of the next time step based on current state. The gradients will be calculated by first backpropagating through 6 to the action and then back-propagating in the neural nets (which is automatically taken care of by deep learning libraries such as TensorFlow).

4.2. CNN

For finite historical time dependency, which assumes current optimal portfolio only depends on finite N historical stock prices, CNN can be a good choice. In this way, the state inputs we defined share many items sequentially. Specifically, this is a stride of LogReturn in time domain, which is very similar to the image patch in image processing, thus motivates us to use CNN for this scenario.

Each input is a $N \times L$ log return matrix plus a vector of previous portfolio with length L. Firstly, for each stock we output a score based only on this stock's historical return and its previous portfolio percentage using CNN and then the scores are combined using a softmax to produce a distribution which becomes our action (portfolio). The architecture of the neural network is shown in Figure 1.

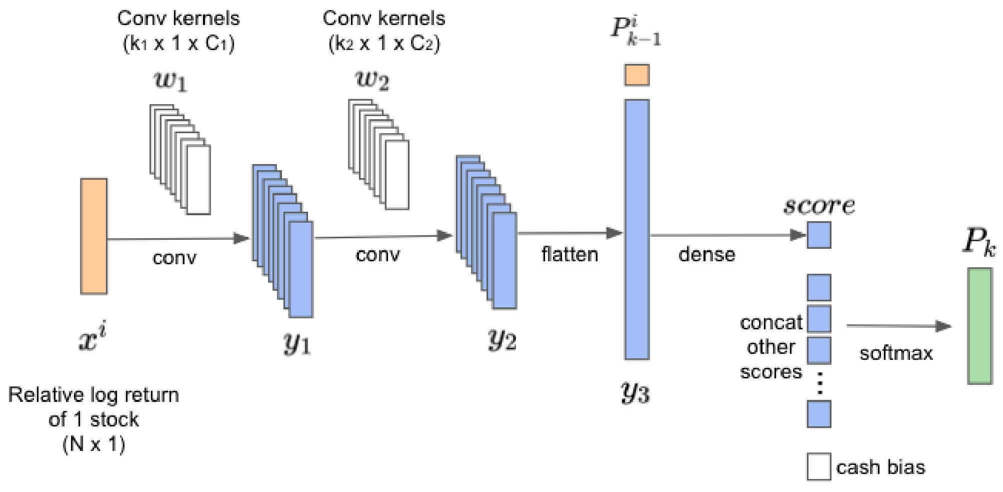


Figure 1. Architecture diagram of CNN model.

4.3. RNN

Compared to feed-forward neural nets, RNN has the advantage of being able to take input sequence of arbitrary length. For basic RNN cell, the relation can be expressed as

$$h_t = f(W_h * h_{t-1} + W_x * x_t) \tag{9}$$

where h_t is the hidden state/output of the cell at time step t, x_t is the input at t and f is a nonlinear activation function (usually tanh or sigmoid). Hence to make predictions (or take actions, in this case) based on the states of N previous days, we can take the log return at each time step, feed them into a RNN and calculate the action based on the output states of RNN cells. The basic architecture of our RNN model is shown in Figure 2.

Note that since the inputs at each time step has low dimension (4 in our case), we chose to use one layer of transforming network to convert it to higher dimensions (8 in our case) that are in turn fed into the RNN cells. Also, since we used Xavier initialization for the weights and 32-bit floating point numbers, we might hit the limit of numerical precision after few time steps into the RNN. Our solution is to magnify the inputs (log returns) by a certain ratio (500 in our case) to prevent falling into the pitfalls of numerical precision.

The scoring network consists of two feed-forward layers and another bias denoting the 'cash', i.e. resources reserved in the allocation, was concatenated to produce a tensor of one more dimension. A distribution for allocation was then produced by a softmax in the end as the action (portfolio) to take in the next time step.

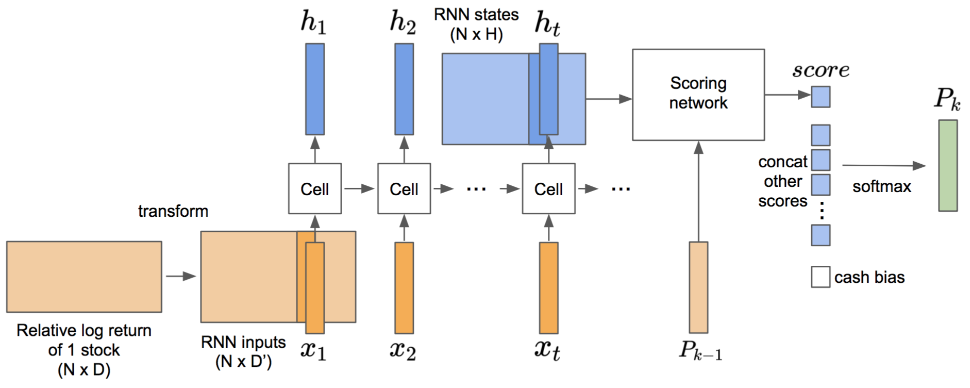


Figure 2. Architecture diagram of RNN model.

4.4. Evaluation

We use the trained model with varying hyperparameters. The models are evaluated on the test set via two metrics. First is the total accumulated return, defined as the product of the returns at all time steps. Second is the Sharpe Ratio [11], defined as the ratio of expected return over the variance of return, indicating how risky is our policy.

5. Benchmarks

5.1. Oracle

The oracle of this problem is set as the agent would know the true stock prices at the next time step. Thus, this agent can straightforwardly maximize the return of current time step in 6 by varying portfolio P_k. Here we apply a greedy method to obtain the maximal return for each time step. This is a simplified version of global maximum for a specific time period, since we don't consider the potential influence on transaction costs for portfolios later than the next time step. However, they are good enough to give us an idea of how well the agent can possibly perform. Specifically, we use cvxpy package to obtain the oracle.

5.2. Baseline

The baseline of this problem is chosen to be the well-known Eigen-portfolio used in tons of financial literature. For time period $[kT, (k+1)T]$, denote expected return for different stocks are $E[r^k] = (E[r_1^k], \ldots, E[r_L^k])$. The eigen-portfolio idea is as follows: given a portfolio P^k, we hope to maximize $E[P^k \cdot r^k]$. Denote $cov(r^k)$ is the covariance of Return for L stocks, then the normalized eigenvector with largest eigenvalue of $cov(r^k)$ is called eigen-portfolio, which has been empirically verified to have the largest correlation with the market.

In order to estimate covariance matrix $cov(r^k)$, we use the weighted average based on the historical data. Here we choose the exponential weights, that is, for data from $[(k-p)T, (k-p+1)T]$, the weight is $\exp(-(p\mu))/C$, where μ is a model parameter and C is the denominator for normalization.

$$\begin{aligned}\hat{E}(r^k) &= \sum_{p=1}^{P} \frac{\exp(-(p\mu))}{C} r^{k-p} \\ c\hat{o}v(r^k) &= \sum_{p=1}^{P} \frac{\exp(-(p\mu))}{C} \times (r^{k-p} - \hat{E}((r^{k-p}))(r^{k-p} - \hat{E}((r^{k-p}))^T\end{aligned} \tag{10}$$

μ is selected by using the precedent time step as validation data. After getting eigen-porfolio, we use (6) to calculate return with transaction costs.

5.3. Best Stock

Best Stock is a benchmark method where the agent knows each stock's growth rate between day one and the last day of consideration, which is $(S_i^{final} - S_i^1)/S_i^1$. The agent just invests all its assets on one stock with the highest such growth rate, and hold it during the whole time period. In this case, each time step's return equals to the growth rate of one best stock.

5.4. Uniform Buy And Hold

We use another benchmark, Uniform Buy And Hold (UBAH), to show how well the agent can perform without any non-trivial strategies. In UBAH, the agent distributes its assets uniformly among all studied stocks. We hope this method can provide us a bottom line for the agent's performance.

6. Experiments

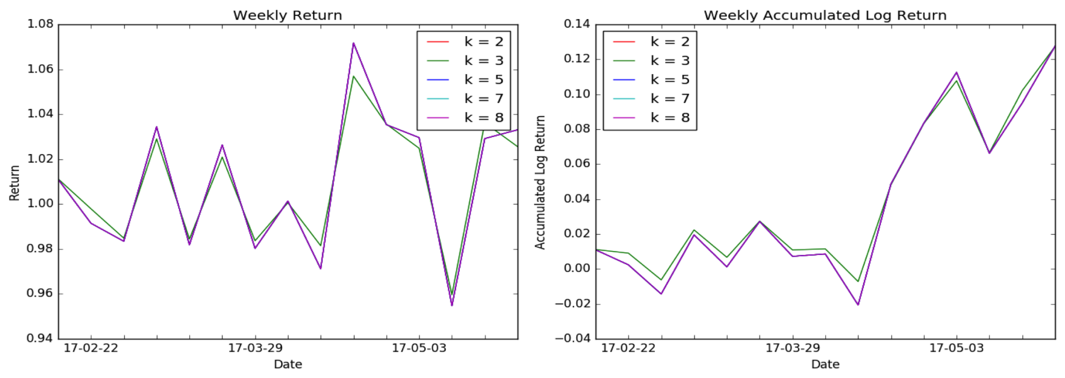
We conducted extensive analysis of the models based on various hyperparameters such as the number of days to look back, the size of CNN kernels and RNN cell hidden size. As stated above, we evaluate the performances based on the total return (TR) and Sharpe Ratio (SR).

The training, validation and testing data date are $[11/05/2014, 02/08/2017]$, $[02/15/2017, 05/24/2017]$, $[05/31/2017, 10/18/2017]$ separately. We did model sensitivity analysis on validation data and compared different model performances on testing data.

6.1. Sensitivity Analysis

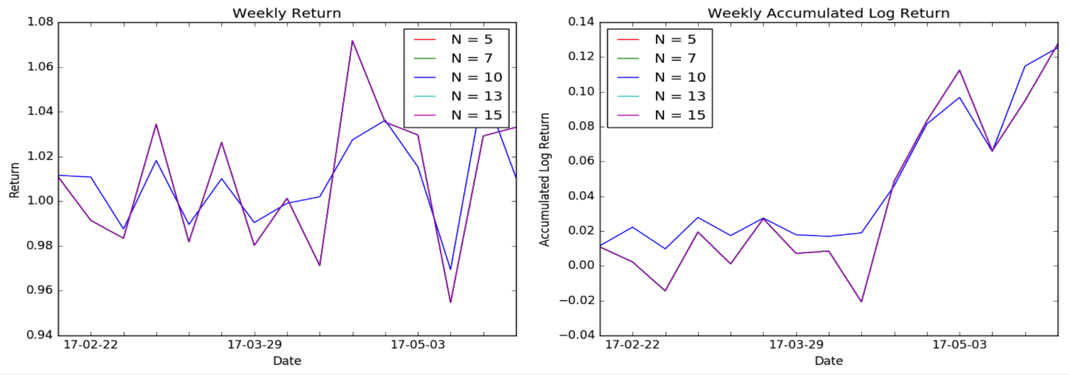
6.1.1 CNN

The weekly return and accumulated return of CNN model under different kernelsize k with $N = 10$ and different N with $k = 3$ is shown in Figures 3 and 4. We can find that varying kernel size k and N does not influence much on total return, while sharpe ratio could be different with different k and N. The best hyperparameters we choose here is $N = 10, k = 3$, which achieves best sharpe ratio on validation data.



k	TR	SR
2	1.14	0.29
3	1.14	0.34
5	1.14	0.29
7	1.14	0.29
8	1.14	0.29

Figure 3. The effect of k (convolutional kernel size) on CNN performance.

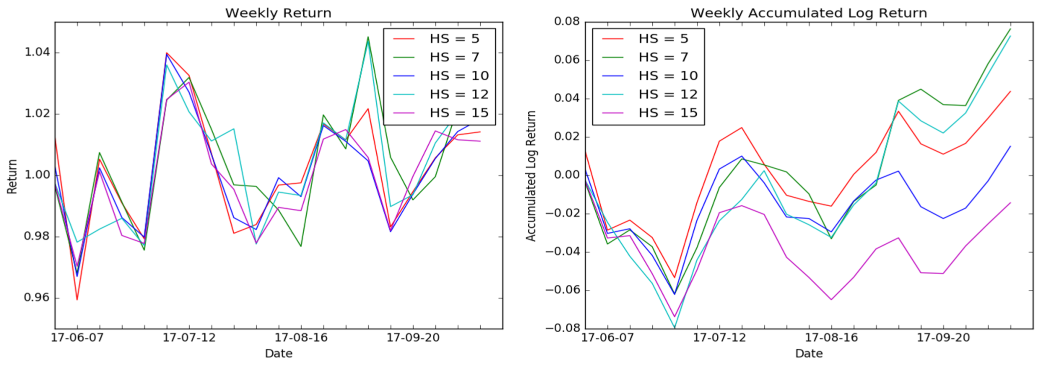


N	TR	SR
5	1.14	0.29
7	1.14	0.29
10	1.13	0.44
13	1.14	0.29
15	1.14	0.29

Figure 4. The effect of N (number of days looking back) on CNN performance.

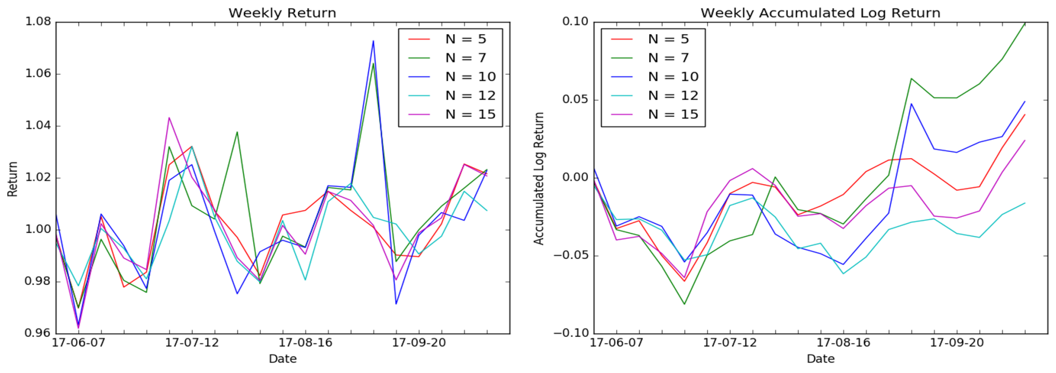
6.1.2 RNN

The weekly return and accumulated return of our RNN model under different kernel sizes are shown in Figures 5 and 6. Based on the total return and sharper ratio, we find that HS and N's influences are not smooth. In this way, we choose our best hyperparameters only by selecting ones with highest total return and sharpe ratio (HS = 7, N = 7).



HS	TR	SR
5	1.04	0.13
7	1.08	0.21
10	1.02	0.05
12	1.08	0.2
15	0.99	-0.04

Figure 5. The effect of HS (hidden size of RNN cell) on RNN performance.



N	TR	SR
5	1.04	0.13
7	1.1	0.23
10	1.05	0.12
12	0.98	-0.05
15	1.02	0.08

Figure 6. The effect of N (number of days looking back) on RNN performance.

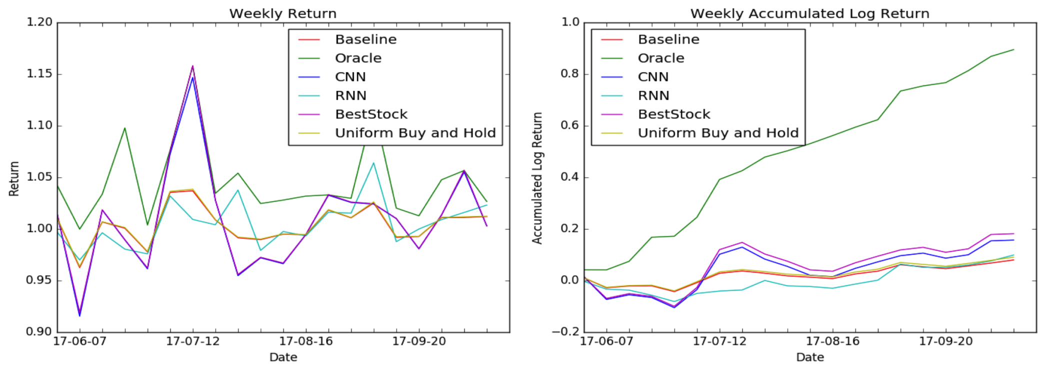
6.2. Model Comparison and Analysis

Figure 7 shows the performances of all our models along with the established benchmarks. These models can be divided into two categories: category one using future data—Oracle and Best Stock—and category two only using historical data: Baseline, CNN, RNN and Uniform Buy and Hold.

For category One, portfolio management is trivial because we have known the tendency of the stock prices. There is no surprise that Oracle is powerful and outperforms all other models.

For category Two, we can see that CNN outperforms all others in the end, showing the effectiveness of our reinforcement learning policy and the potential that useful information can be extracted from historical stock prices to predict the future. Note that CNN has poorer performance than baseline at the beginning, and later becomes better, which is coherent with online learning update while our policy gets better during the reinforcement learning as time continues.

However, the stock data is noisy and many outside factors other than historical prices can influence the price trend, which limits the ability of our model. Another noticeable thing is that RNN does not outperform CNN and Uniform Buy and Hold, which is against our previous expectation, because RNN can be asymptotically considered as depending on infinite history while CNN only depends on finite history and Uniform Buy and Hold does not pay attention to price tendency. This could be due to that the RNN cell we used is not complex enough and the price data is highly noisy which impacts RNN learning.



Model	TR	SR
BL	1.08	0.24
OC	2.45	1.22
CNN	1.17	0.19
RNN	1.10	0.23
BS	1.20	0.21
UBAH	1.09	0.26

Figure 7. Comparison of all models and benchmarks.

7. Conclusion

In this project, we have formulated the portfolio management problem into a deep RL problem with DPG. We successfully implemented the models in TensorFlow and trained on historical data from S&P 500.

At the end the test results of our models only surpassed the baseline by a slight margin. It is possible that the results are caused by the following reasons: 1. Deep learning models tend to have high variance, which makes them hard to train on financial data with high noise. 2. Our data might be insufficient. Some work [6] train their models on crytocurrency data taken every 30 min. Our model was trained on stock prices every week. 3. The network structures require more refinement, e.g. the softmax operation at the end might be problematic because it changes a possibly linear scale in the scores to a exponential scale, or the cash bias should be a parameter to be learned instead of setting to 0 for cash. 4. More useful outside information other than historical stock prices can be integrated into networks such as news and fiscal policies.

To improve this project, one way is to train on a larger dataset (preferably with cryptocurrency data) and also take into account the interactions between assets. Another way is to integrate other inputs such as news sentiment into the network to help make better decisions. Also, the network structure is something we can work on. We could change how we calculated action based on scores as discussed above. We could also try to use other innovative architectures such as actor-critic [10] to help learning. To reduce the variance of the models, mechanisms such as attention [12] and dropout [13][14] can be applied as well. Deep learning in finance is very promising since it might be able to capture some information is an extremely complicated environment.

References

[1] *Modern Invest Theory*, Robert A. Haugen, *Prentice Hall*, 1986

[2] *Online portfolio selection: A survey*, Bin Li and Steven CH Hoi, *ACM Computing Serveys (CSUR)*, 46(3):35, 2014

[3] *Nonparametric Kernel-based Sequential Investment Strategies*, Laszlo Gyorfi, Gabor Lugosi, and Frederic Udina, *Mathematical Finance*, 16(2):337357, 2006

[4] *Deep learning for finance: deep portfolios*, J. B. Heaton, N. G. Polson, and Jan Hendrik Witte, *Applied Stochastic Models in Business and Industry*, 2016, ISSN 1526-4025. doi: 10.1002/ASMB.2209

[5] *Forecasting S&P 500 index using artificial neural networks and design of experiments*, Seyed Taghi Akhavan Niaki and Saeid Hoseinzade, *Journal of Industrial Engineering International*, 9(1):1, 2013. ISSN 2251-712X. doi: 10.1186/2251-712X-9-1

[6] *Deep direct rein- forcement learning for financial signal representation and trading*, Yue Deng, Feng Bao, Youyong Kong, Zhiquan Ren, and Qionghai Dai, *IEEE transactions on neural networks and learning systems*, 28(3):653664, 2017

[7] *A Deep Reinforcement Learning Framework for the Financial Portfolio Management Problem*, Zhengyao Jiang, Dixing Xu, Jinjun Liang, *arXiv:1706.10059v2*

[8] https://github.com/wassname/rl-portfolio-management

[9] *Deterministic Policy Gradient Algorithms*, Silver et al, *arXiv:1509.02971*

[10] *Deep Deterministic Policy Gradients*, Lillicrap et al, *arXiv:1509.02971*

[11] *The sharpe ratio*, Sharpe, W. F. (1994), *The Journal of Portfolio Management*, 21(1), 49-58.

[12] *Effective Approaches to Attention-based Neural Machine Translation*, Luong et al, *arXiv:1508.04025*

[13] *Improving deep neural networks for lvcsr using rectified linear units and dropout*, Dahl et al, *IEEE 2013*, DOI: 10.1109/ICASSP.2013.6639346

[14] *Understanding the dropout strategy and analyzing its effectiveness on LVCSR*, Li et al, *IEEE 2013*, DOI: 10.1109/ICASSP.2013.6639144