Adversarial Reinforcement Learning for Portfolio Management

Akarsh Zingade Minsu Yeom Sidharth Prasad Sai Sourabh Madur

Abstract

In this paper, we created a virtual stock exchange where multiple agents invest in stocks against each other. Three of the agents implemented their own strategies based on different Reinforcement Learning (RL) algorithms. No historical data was used to train these RL agents. RL agents are given cash only and do not have any stocks at an initialization phase. Trading environments were designed in a way they accommodate realistic factors such as liquidity costs (buy expensive, sell cheap due to little orders that can be matched), transaction costs, and a management fee. We achieved this by introducing a sophisticated framework that admits the complex nature of the trading environment in the real world. We present an approach of which algorithms the RL agents are based on, how we set our environment, and how a virtual stock exchange was constructed. We used Deep Neural Network(DNN) to represent the Actor-Critic network, which takes an input of stock prices and portfolio weights and outputs a vector of portfolio weights. RL agents' action are concretely implemented by placing orders and they do not know how their states would be until the orders are matched. We introduced non-RL agents, one of which, namely a mean-reversion agent, is in favor of our market environment in its strategy, and the other of which, namely a random agent, is to model the irrational behaviors of the real stock market. We end this paper with experimental analysis and future work.

1. Introduction

The total value of stocks traded worldwide in the year 2017 was US$ 77 trillion which comprised of 117% of the world GDP for that year (WorldBank, 2018). Given that approximately 20% of households' and nonprofit organizations' total financial assets are estimated to be invested in equity markets (FRED, 2018), even a slight increase in percentage return would have far reaching effects in practice. The current distribution of the class of machine learning (ML) algorithms applied to finance is skewed towards supervised learning. Due to the nature of this relatively classical ML technique, most of its applications are limited to predicting stock prices based on historical data (Fischer, 2017). However, institutional-level investors require a more sophisticated and flexible approach to incorporate both the indigenous as well as the exogenous factors within a given quantitative investing strategy. Research centered around supervised learning has revealed a few major limitations of not incorporating such factors within the models. For example, the dynamics of transaction costs, which is an indigenous factor, is often not incorporated in modeling whereas in the real world, dedicated professionals are made responsible for reducing transaction costs in practice due to their large effect on performance. An exogenous factor often ignored is non-stationarity, i.e. making the investment strategies *online* - adapting to the changing market conditions.

In this project, we plan to create a virtual stock market wherein agents invest in stocks using their own Reinforcement Learning (RL) and non-RL based strategies. In order to resolve the limitations of existing quantitative approaches, our project structurally accommodates as many realistic factors as possible by leveraging the power of RL. Furthermore, it is desirable to have an end to end integrated framework for training and back-testing the RL based systematic trading strategies prior to deploying it into production, which we attempt to provide.

The markets are constantly changing and often, the market regime may change without any pre-signal which severely hurts the values of investors' assets. In history, capital markets have witnessed almost no single investor who has been able to beat the market consistently. Only a few famous investors such as *Warren Buffet* are known for their insightful investing, having overcome the negative effects of regime changes over a long period of time based on their *qualitative* value-investing philosophy.

One contrasting approach would be a quantitative investing strategy which does not allow human intervention during any investment decision-making process. The strengths of traditional quantitative strategies have made them grow drastically in quality and quantity, leading to them being one of the two major strategies since time being. Currently, quantitative funds are responsible for approximately 25% of the total trades in the US (Zuckerman, 2017). Taking this further, we try to introduce a novel quantitative investment model by introducing multi-agents working with different kinds of trading strategies, imposing a realistic environment with real-world trading rules, and performing industry-level evaluations toward the agents' behavior. In addition, we attempt to learn from scratch using self-play, i.e. we do not use any historical market data for training our RL agents.

2. Background/Related Work

Surprisingly, RL based investing has drawn relatively less attention from researchers and practitioners as an alternative. (Liang, 2014) explores Deep Reinforcement Learning (DRL) in the context of portfolio management wherein the authors explore the influence of different optimizers and network structures on performance of trading agents, utilizing two kinds of DRL algorithms, deep deterministic policy gradient (DDPG) and proximal policy optimization (PPO). One of their significant contribution is to show that the existence of transaction cost translates the problem from a prediction problem (whose global optimized policy can be obtained via a greedy algorithm) into a compute-expensive dynamic programming problem.

DRL techniques are proven to have their own advantages over classical RL. They approximate strategy or value functions using non-linear approximations (which is often based on neural networks). This not only provides us with the flexibility of designing and experimenting with different network structures but also prevents the so called 'curse of dimensionality', enabling large-scale portfolio management.

Deep Deterministic Policy Gradient (DDPG) is a combination of Q-learning and policy gradient and succeeds in using a neural network as its function Wapproximator based on Deterministic Policy Gradient Algorithms (Liang, 2014). Proximal Policy Optimization (PPO) is a policy gradient method based on Trust Region Policy Optimization (TRPO) (Silver, 2014).

3. Approach: a virtual stock market with multiple agents

We developed a framework that accommodates the complexity of stock market work flows. The framework consists of the following three components:

- **Agents:** These are abstraction of the portfolio managers in the real world. An RL agent takes an input $S = (P_t, w_t)$ and returns an output $A = (w_{t+1})$, where P_t is a vector of asset prices p_t, relative to p_{t-1}. Assets consist of cash and N stocks, making it a total of $N + 1$ instruments. w_t is a vector of asset weights at time step t and w_{t+1} is a vector of asset weights at time step $t + 1$. A constraint is imposed on the weight vectors such that its elements sum up to one. The RL agent is based on the *(Adversarial) Policy Gradient (PG)* (Liang, 2018) algorithm. For a purpose of comparison, two other RL algorithms were implemented on which an RL agent is based: Deterministic Policy Gradient algorithms (Silver, 2014) and Proximal Policy Optimization(PPO) (Schulman, 2017a). In addition, there exist other non-RL agents. Further details on the kinds of agents is described in the following subsections.

- **Environment:** This component manages the general RL algorithm flow. It comprises of the *step* function which (1) allows agents to take an action, i.e. place an order at a given time step, (2) calculates rewards, and (3) updates (P, w_t) and passes it onto the next step, starting from $t = 0$ until the end of an epoch. The environment facilitates information flow between the components by providing and API for queries such as stock prices between the *Agents* and the *Exchange*.

- **Exchange:** This represents a virtual stock market comprising of the *Order Book*. Specifically, it implements order-driven markets (Ananth Madhavan & Wagner, 2016), wherein trades are executed directly with the virtual exchange without intermediate dealers. The agents specify a tuple consisting of the bid price and order size when they want to buy stocks and analogously, a tuple consisting of the ask price and order size when they wish to sell stocks. This implies that a bid-ask spread is naturally introduced by the agents' own actions, i.e., while deciding their target portfolio weights.

Our framework, comprising of the three components described above is distinguished from the ones in literature, for the latter typically rely on historical prices at which trades are matched without introducing the bid-ask spread feature, which is in reality an important factor to to be considered in equity trading.

3.1. Methodology

Initialization and agent setting: We start from an initial state which can be viewed as the Initial Public Offering (IPO) of various companies, wherein there is no history of trades to learn from. To be precise, our implementation admits an arbitrary number of agents, assets, and initial amounts of cash. Our environment is initialized with five stocks with different initial offering prices P_i and volume V_i for $i = 1, 2, \ldots, 5$. We consider cash as an investable instrument and this is of importance, which we will discuss later in **3.2. Implementation details and 5.1 Future work**. We then setup five trading agents, the first three of which are RL agents, the fourth a random agent, and the fifth a mean-reversion agent, contending against each other. The RL agents may be one of three kinds: PO agent, DDPG agent, or PPO agent(PO stands for (Adversarial) Policy Gradient, DDPG Deep Deterministic Policy Gradient,and PPO Proximal Policy Optimization. Their details can be found in sections **3.3 Algorithms**). All RL agents and a random agent start with $10,000$ amount of cash only *(zero stock)*, a mean-reversion agent begins with just a few stocks (less than 10 percent of the total value) due to a technical reason with the time constraint. Each of agent is governed by their own strategy. We have the random agent which pulls (injects) liquidity from (into) the market, in order to model the real world setting as closely as possible. We expect our RL agents to learn these irrational behaviors and the best way to exploit them for their own benefit.

The stock exchange: The stock exchange would receive the buy/sell orders from each agent and maintain an order book. Agents would place/remove their buy/sell orders to/from the exchange, which will be responsible for simulating the order-book dynamics, leading to price movements. The trading agents observation would be the current price of each of the stock listed in the exchange. In other words, the environment's action is to output the stock prices after all agents have taken their turns to place their orders in a random order with equal and fair probability. The agents' action is to buy/sell/hold each of the stocks in the exchange, and in what volume.

To summarize, the exchange is initialized with different values for each of the stocks. All the agents start trading using their own strategies to maximize the (risk-adjusted) returns essentially learning their optimal policies.

3.2. Implementation details

Deep Reinforcement Learning(DRL) structure: The Actor Critic network is represented by Deep Neural Networks (DNN). The Network takes the input $S = (P_t, w_t)$ where $P_t = [P_{Si,t}]$ represents the relative prices of the stocks, and w_t represents the current portfolio vector of the agent. We reserve $i = 0$ as cash because cash is one of the assets agents trade. The other Investment attraction over the six assets (one cash, five stocks) are ranked by an weight to the corresponding asset, i.e., the more attractive the asset is, the higher the weight is. This ranked-weight is an output - we call it w_{t+1} - of the DNN, which is also an action that the corresponding RL agent takes. This w_{t+1} represents the new portfolio that the agent believes is good for $t + 1$. This action is implemented by placing an order directly to the virtual stock market. When we train the DNN, we set learning rate to 10^{-3} and used Adam Optimizer. TensorFlow is chosen as a tool to implement them.

RL algorithm selected, rationale and additional achievement: Adversarial Policy Gradient(PG) was implemented and successfully incorporated within our framework. PG was our foremost intended result out of other RL algorithms such as DDPG and PPO. This preference was due to the complicated nature of the stock market. Arguably, every news in the world may have an impact on the stock market, and (Liang, 2018) also took this factor into account in their suggestion of PG algorithm; adding noise $\sim N(0, 0.002)$ to stock price. Agreeing with their rationale, and taking that even further by utilizing our own framework, we successfully and naturally added *transaction costs* into trading process. One way of achieving it was to add positive noise to a bid price, i.e., a price at which an agent is willing to *buy*, to make a trade "buy more expensive" one. Likewise, a negative noise was added to an ask price, i.e., a price at which an agent is willing to *sell*, to make a trade "sell cheaper" one. Specifically,

$$bid \text{ price (to buy) = market price } + Uniform(0,1)$$

$$ask \text{ price (to sell) = market price } - Uniform(0,1)$$

This reflects the lower liquidity of our virtual market and we call this *transaction costs*. In order to trade and make some profits, agents should overcome this inherent constraints. In implementation, we rounded the price to one decimal point to model a *tick-size* convention of exchanges in the real world. In our proposal, we originally mentioned only to reflect the dynamics of transaction costs, which are fixed fees that are charged every time a trade is matched. However, we not only implemented the transaction costs but also created a more realistic trading framework.

Other costs: A trading fee is set from 0 up to 1 percent of traded amount. Performances varied with different trading fees. Management fee was not explicitly deducted from cash of a portfolio, but was used within a reward function.

Reward function: By the nature of stock investment, all investment strategies share, at least to the some degree, a common goal: maximizing profits. This simplifies a reward function. We selected a relative return to the market as a reward function r:

$$r = (pv_t - pv_{t-1}) - (mv_t - mv_(t-1)) - c,$$

where pv_t, pv_{t-1} is the portfolio value at time = t and $t-1$, respectively. Likewise, mv_t, mv_{t-1} is the market value at time = t and $t-1$. c is the management fee, which can be thought of profit that the asset management company running a portfolio makes. The rational behind this selection is that a portfolio manager who manages to outperform the market should be rewarded, and beating the market is a very hard goal to achieve.

Limit: We implemented PPO in our framework as well. However, we did not have the time to conduct experiments thoroughly.

3.3. Algorithms

Proximal Policy Optimization(PPO): (Schulman, 2017a) is a Policy Gradient method. PPO is essentially based on Trust Region Policy Optimization(TRPO) (Schulman, 2017b). The main improvement over TRPO is the introduction of clipped surrogate objective. The clipped surrogate objective can be written as:

$$L^{CLIP}(\theta) = \mathbf{E}[\mathbf{min}(\mathbf{r}(\theta)\mathbf{A}, \mathbf{clip}(\mathbf{r}(\theta), \mathbf{1}-\epsilon, \mathbf{1}+\epsilon)\mathbf{A})]$$

The whole PPO algorithm can be summarised as follows :

Algorithm 1 PPO

```
Randomly initialize actor $\mu : S \rightarrow \mathbb{R}^{m+1}$ and $\sigma : S \rightarrow diag(\sigma_1, \sigma_2, \dots, \sigma_{m+1})$
for $i = 1$ to M do
  Run policy $\pi_\theta \sim N(\mu(s), \sigma(s))$ for $T$ time steps and collect $(s_t, a_t, r_t)$
  Estimate advantages $\hat{A}_t = \sum_{t'>t} \gamma^{t'} r_{t'} - V(s_t)$
  Update old policy $\pi_{old} \leftarrow \pi_\theta$
  for $j = 1$ to N do
    Update actor policy by policy gradient:
      $$\sum_i \nabla_\theta L_i^{CLIP}(\theta)$$
    Update critic by:
      $$\sum_i \nabla_\theta L_i^{CLIP}(\theta)$$
    Update actor policy by policy gradient:
      $$\nabla L(\phi) = -\sum_{t=1}^{T} \nabla \hat{A}_t^2$$
  end for
end for
```

Deep Deterministic Policy Gradient(DDPG): DDPG uses a Neural Network as its function approximator which is based on Deterministic Policy Gradient algorithms (Silver, 2014). It combines Q-learning and Policy Gradients. DDPG uses four networks- online actor, online critic, target actor and target critic. The online critic evaluates the action provided by the actor, and updates the online actor. The target actor and target critic are used to update the online critic. It is described as follows :

Algorithm 2 DDPG

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

Adversarial Policy Gradient(PG): The work in (Liang, 2018) found that the state-of-the-art algorithms DDPG and PPO performed poorly in learning to manage the portfolio in United States and China Stock Market data. We implement the adversarial training proposed in that paper. It is described as follows :

Algorithm 3 Adversarial Policy Gradient (PG)

```
Randomly initialize actor $\mu(s|\theta^{\mu})$
Initialize replay buffer $R$
for $i = 1$ to M do
    Receive initial observation state $s_1$
    Add noise into the price data
    for $t = 1$ to T do
        Select action $\omega_t = \mu(s_t|\theta^{\mu})$
        Execute action $w_t$ and observe $r_t, s_{t+1}$ and $\omega'_t$
        Save transition $(s_t, \omega_t, \omega'_t)$ in $R$
    end for
    Update actor policy by policy gradient:
        $\nabla_{\theta^{\mu}} J =$
        $\nabla_{\theta^{\mu}} \frac{1}{N} \sum_{t=1}^{T} \left( log(\omega_{\mathbf{t}} \cdot \mathbf{y_t} - \mu \sum_{\mathbf{i=1}}^{\mathbf{m}} |\omega_{\mathbf{i,t}} - \omega'_{\mathbf{i,t-1}}|) \right)$
end for
```

4. Experiment results

4.1. Data

We do not use an external data source, as described in Section 3, our testing-framework is similar to *Alpha Go Zero* (Silver, 2017) , where we let the agents play against each other and learn from experience.

4.2. RL Results

By setting the first three agents as RL agents, we tested the performance of our algorithms with different transaction costs.

4.2.1. PG agent, transaction cost(tc)=0%

By setting a transaction cost to zero, we started with three PG agents and two non-RL agent. As can be seen in 1, the best performer was Agent 3, which is a PG agent, by about 15% (1,500 unit-money). RL agents' action, or portfolio weights, varied from 8% to 29% at the last time step, implying their active actions taken. Surprisingly, a random agent was the second best, followed by the other two RL agents. Since we did not impose any transaction cost, the stock prices went up during this period, a mean-reversion agent seems not to have many trading opportunities during this period. This makes sense because the agent takes advantage of 'buy the low' and 'sell the high', but the market did not go in that way.

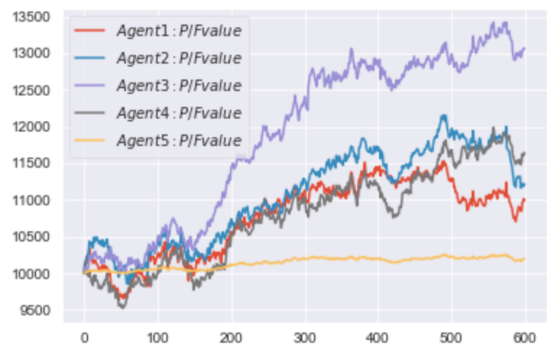

Figure 1. Portfolio value (PG, tc=0%)

4.2.2. DDPG agent, transaction cost(tc)=0%

A DDPG agent was the best performer in this case as well but by lesser degree in 2. A random agent also takes a benefit of no transaction cost imposed and increasing stock prices.

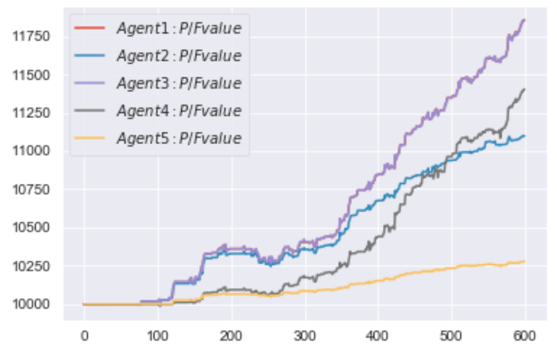

Figure 2. Portfolio value (DDPG, tc=0%)

4.2.3. PG agent, transaction cost(tc)=0.05%

3 shows the most realistic test case with a transaction cost of 0.05%. As expected, our PG agent (Agent 1) outperformed all other agents, followed by another PG agent. RL agents' investment weights in stocks varied from 0% to 37.7%, again reflecting their active decision-making. The performances of both of the non-RL agents are not satisfactory. During this time, stock prices are up. We noticed that agent 3 decided to prefer cash from early time steps to the end.

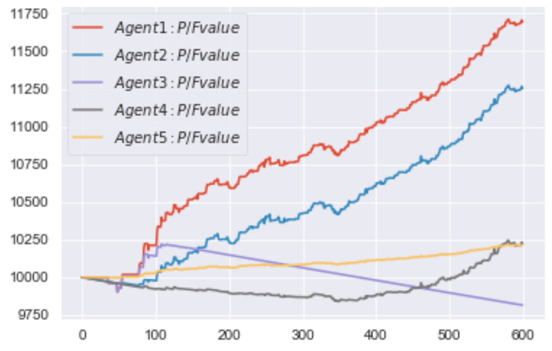

Figure 3. Portfolio value (PG, tc=0.05%)

4.2.4. PG AGENT, TRANSACTION COST(TC)=1%

4 showed a dramatic case. We set a huge transaction cost of 1 percent. In this case, four stock prices were up; 5%, 25.7%, 29.8%, 43.5%. The other remained at the same price. Still, two PG agents managed to preserve their cash. This result is surprising if it's compared to 5. A random agent's asset almost went to zero, and so did another RL agent.

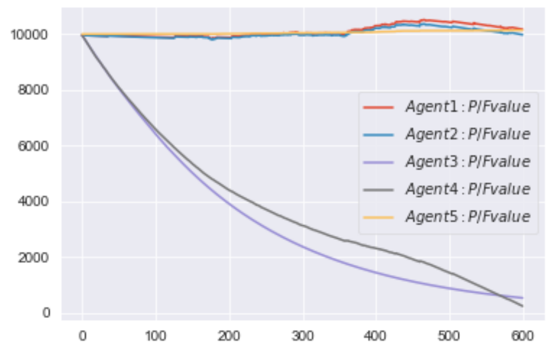

Figure 4. Portfolio value (PG, tc=1%)

4.2.5. DDPG AGENT, TRANSACTION COST(TC)=1%

DDPG agents in 5 did not perform well when transaction cost was set to 1%. All, but a mean-reversion agent, lost money with all stock prices going *up* from 29.6% up to 57.3% . This emphasizes how large transaction cost could be, and therefore, a sophisticated model should incorporate the dynamics of this cost within the model as our framework does.

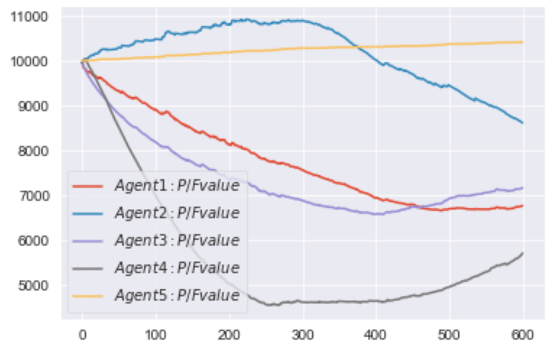

Figure 5. Portfolio value (DDPG, tc=1 %)

4.3. Non-RL agents

We test the exchange by introducing two types of Non-RL agents(agents which use conventional trading strategies) These are (i)Random agents and (ii) Mean reversion agents. Random agents place a buy/sell order randomly with equal probability and at a price drawn from the distribution

$$Buy/Sell\ price \sim Stock\ price + \mathcal{N}(\mu, \sigma^2)$$

where μ is set to 1 and σ set to 5. Mean reversion agents place a buy/sell order whenever the stock price goes beyond one standard deviation of the stock price history for a defined window length. The trading strategy is as follows:

Algorithm 4 Mean Reversion Trading Strategy

```
(i) Get the stock price history for the past $L$ rounds of trading, where $L$ is the window length.
(ii) Calculate the mean $\mu$ and variance $\sigma$ of this stock price history.
(iii) Calculate the z-score, which is defined as
    $$z\_score = \frac{stock\_price - \mu}{\sigma}$$
where $\mu$ and $\sigma$ are calculated in step (ii)
(iv) Now,
if $z\_score > 1$ then
    Sell the stock at the current stock price.
else if $z\_score < -1$ then
    Buy the stock at the current stock
else
    Do not take any action.
end if
```

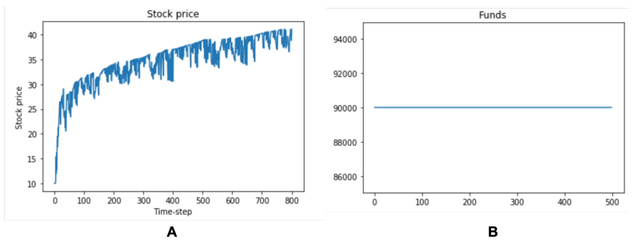

Figure 6. **A.** Stock price movement and **B.** Average Funds , when 4 random agents trade in the exchange.

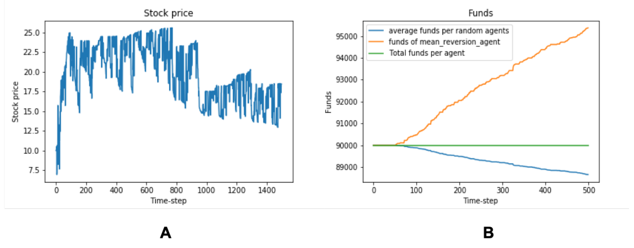

Figure 7. **A.** Stock price movement and **B.** Funds , when 4 random agents and one mean reversion agent trade in the exchange.

4.3.1. Testcase 1: Random agents

The first test case we consider is when we have four random agents which are allowed to trade(Figure 6).
As the randomness parameter μ is set to a positive value, the trend in stock price is always upward(Figure 6 **A**). This happens as each of the agent wants to sell at a price larger than than what it had bought the stock resulting in a bullish market.
Another point to note is that the total funds i.e the sum of the funds of all the agents remains the same throughout the trading period. This is expected as the stock market is a zero sum game.

4.3.2. Testcase 2: Random agents + mean reversion agent

The second testcase we consider is when a mean reversion agent is allowed to trade in addition to the four random agents introduced previously.
In this case, the mean reversion agent takes advantage of the variation in stock prices and makes a profit unlike the random agents, which lose money. This can be seen from Figure 7 where the funds of the mean reversion agent increase once it starts trading(at $t = 50$(window length)) and the funds of the random agents dwindle.

5. Conclusion

We introduced a novel environment where trading agents, starting with only cash, can learn how to trading within the pre-defined trading rules. We implemented three RL agents and two non-RL agents. Our analysis revealed that transaction costs significantly impact on portfolio performance, and it implies that any simulation without appropriately transaction costs would result in disappointing performance in reality. Our framework shows that learning from *'zero'* is also achievable.

5.1. Future work

More agents added: We have three RL agents and two non-RL agents implemented. Given the time constraint of this project, this can't be considered a small number. However, in order to mimic the real stock market, it is desirable to have much more agents. Expected results would be to see more diverse types of investment strategies that RL agents may learn.

A various ways of evaluations: Currently, we have our evaluation as a relative return to the stock market minus costs involved (such as liquidity costs, management fees and transaction fees).

More unconstrained environment: We did not allow agents to any kind of short selling, e.g., borrowing stocks and selling them to buy back at cheaper price later to make some profits. This would lead this environment to an even higher level because, in even in the reality, borrowing stocks engages a complicated process with several entities involved. Despite of this potential complexity, absolute returns would be achievable only by having this feature in the market. Put simply, an agent may earn money even when stock prices fall down, pursuing an 'absolute return' regardless of a direction in which the market heads.

Asset allocation strategy: Our framework and RL agents' strategy are flexible so that cash can be regarded as an investment instrument. The DNN decides a preference of cash to other stocks by specifying cash weight. This has a far-reaching potential to expand it to another, new strategy; asset allocation. In this strategy, portfolio managers run multiple assets, which are, for example, cash, stocks, fixed income, currency and/or commodity. Their roles are not to pick stocks, but allocate a *weight* to each asset. Since our RL agents are able to separate cash from stocks in terms of investment attraction, further work would bring an applicable result.

6. Contributions of each member

We all together made this report complete and showed a possibility that multiple RL agents can compete each other in a virtual stock market. From the extent of our knowledge, this was not achieved before. In this sense, we believe we contributed the RL community to move forward.

Akarsh Zingade: Implemented Deep Neural Network the two RL agents(DDPG, PPO).

Minsu Yeom:: Implemented the one RL agent(PG). Conducted experiments and analysis for RL agents.

Sidharth Prasad: Implemented the environment and exchange.

Sai Sourabh Madur: Implemented the two non-RL agents. Conducted experiments and analysis for non-RL agents.

References

Ananth Madhavan, Jack L. Treynor and Wagner, Wayne H. (eds.). *CFA Level III Volume 6 Trading and Rebalancing, Performance Evaluation, and Global Investment Performance Standards, Chapter 29*. CFA Institute, Charlottesville, Virginia, 2016.

Fischer, Thomas G. Reinforcement learning in financial markets - a survey. In *FAU Discussion Papers in Economics, No. 12)*, 2017.

FRED, Federal Reserve Bank of St. Louis. Households and nonprofit organizations; corporate equities; asset, level [hnoceaq027s]. Technical report, Board of Governors of the Federal Reserve System (US), 2018.

Liang, Zhipeng, et. al. Deterministic policy gradient algorithms. ICML, 2014.

Liang, Zhipeng, et. al. Adversarial deep reinforcement learning in portfolio management. ICML, 2018.

Schulman, John, et al. Proximal policy optimization algorithms. 2017a.

Schulman, John, et al. Trust region policy optimization . 2017b.

Silver, David, et. al. Deterministic policy gradient algorithms. 2014.

Silver, David, et. al. Mastering the game of go without human knowledge. Naature, 2017.

WorldBank. World bank open data, 2018.

Zuckerman, Fletcher. Flush with cash, top quant funds stumble, 2017.