MAPS: Multi-agent Reinforcement Learning-based Portfolio Management System

Jinho Lee[*] , **Raehyun Kim**[*] , **Seok-Won Yi** and **Jaewoo Kang**[†]
Department of Computer Science and Engineering, Korea University
{jinholee, raehyun, seanswyi, kangj}@korea.ac.kr

Abstract

Generating an investment strategy using advanced deep learning methods in stock markets has recently been a topic of interest. Most existing deep learning methods focus on proposing an optimal model or network architecture by maximizing return. However, these models often fail to consider and adapt to the continuously changing market conditions. In this paper, we propose the Multi-Agent reinforcement learning-based Portfolio management System (MAPS). MAPS is a cooperative system in which each agent is an independent "investor" creating its own portfolio. In the training procedure, each agent is guided to act as diversely as possible while maximizing its own return with a carefully designed loss function. As a result, MAPS as a system ends up with a diversified portfolio. Experiment results with 12 years of US market data show that MAPS outperforms most of the baselines in terms of Sharpe ratio. Furthermore, our results show that adding more agents to our system would allow us to get a higher Sharpe ratio by lowering risk with a more diversified portfolio.

1 Introduction

Most trading decisions nowadays are made by algorithmic trading systems. According to the Deutsche Bank report, the share of automated high-frequency trading in the equity market resulted in a total of 50% in the US [Kaya *et al.*, 2016].

Decision-making processes based on data analysis are called quantitative trading strategies. Quantitative trading strategies can be divided into two categories: fundamental [Abarbanell and Bushee, 1997] and technical analysis [Lo *et al.*, 2000; Park and Irwin, 2007]. Fundamental analysis refers to performing analysis based on real-world activity. Therefore, fundamental data analysis is mostly based on financial statements and balance sheets. On the other hand, technical analysis is solely based on technical signals, such as historical price and volume. Technicians believe that profitable patterns can be discovered by analyzing historical movements of prices. Traditional quantitative traders attempt to find profitable strategies by constructing algorithms that best represent their beliefs of the market. Although they provide rational clues and theoretical justification of their logic, traditional quantitative strategies are only able to reflect a part of the entire market dynamics. For instance, the momentum strategy [Jegadeesh and Titman, 1993] assumes that if there exist clear trends, prices will maintain their direction of movement. The mean reversion strategy [Poterba and Summers, 1988] believes that asset prices tend to revert to the average over time. However, it is nontrivial to maintain stable profits under evolving market conditions by leveraging only specific aspects of the financial market.

Inspired by the recent success of deep learning (DL), researchers have put much effort into finding new profitable patterns from several factors. Early approaches using DL in financial applications focused on how to improve the prediction of stock movements. Qin *et al.* proposed hierarchical attention combined with a recurrent neural network (RNN) architecture to improve time series prediction. Besides using traditional signals such as stock chart information, there have been numerous attempts to find profitable patterns from new factors like news and sentiment analysis [Xu and Cohen, 2018], [Ding *et al.*, 2015]. More recently, [Feng *et al.*, 2019] and [Kim *et al.*, 2019] attempted to create more robust predictions by incorporating adversarial training and corporate relation information, respectively.

Forecasting models, such as the ones mentioned above, require explicit supervision in the form of labels. These labels take on various forms depending on the task at hand (e.g. up-down-stationary signals for classification). Despite its simple facade, the defining and design of these labels is nontrivial.

Reinforcement learning (RL) approaches provide us with a more seamless framework for decision making [Bacoyannis *et al.*, 2018]. The advantage of using RL to make trading decisions is that an agent is trained to maximize its long term reward without supervision. Deng *et al.* applied RL with fuzzy learning and recurrent RL. Xiong *et al.* proved RL's effectiveness in asset management.

Although the aforementioned work shows promising results, there still remain many challenges in applying DL to portfolio management. Most existing methods utilizing DL focus on proposing a model which simply maximizes ex-

[*] Equal contribution.

[†] Corresponding author.

pected return without considering risk factors. However, the ultimate goal of portfolio management is to maximize expected return *constrained to a given risk level*, as stated in modern portfolio theory [Markowitz, 1952]. In other words, we must consider risk-adjusted return (e.g. Sharpe ratio) rather than expected return. There has been relatively few work that has considered risk-adjusted return metrics.

In this paper, we propose a cooperative Multi-Agent reinforcement learning-based Portfolio management System (MAPS) inspired by portfolio diversification strategies used in large investment companies. We focus on the fact that investment firms not only diversify assets composing the portfolios, but also the portfolios themselves. Likewise, rather than creating a single optimal strategy, MAPS creates diversified portfolios by distributing assets to each agent.

Each agent in MAPS creates its own portfolio based on the current state of the market. We designed MAPS' loss function to guide our agents to act as diversely as possible while maximizing their own returns. Agents in MAPS can be seen as a group of independent "investors" cooperating to create a diversified portfolio. With multiple agents, MAPS as a system would have a *portfolio of portfolios*.

We believe that no single strategy fits every market condition, so it is integral to diversify our strategies to mitigate risk and achieve higher risk-adjusted returns. Each agent works towards optimizing a portfolio while keeping in mind that the system as a whole would suffer from a lower risk-adjusted return if they were to create portfolios similar to that of other agents. Our contribution can be summarized as follows:

- To the best of our knowledge, this is the first attempt to use cooperative multi-agent reinforcement learning (MARL) in the field of portfolio management. Given raw financial trading data as the state description, our agents maximize risk-adjusted return.
- We devise a new loss function with a diversification penalty term to effectively encourage agents to act as diversely as possible while maximizing their own return. Our experimental results show that the diversification penalty effectively guide our agents to act diversely when creating portfolios.
- We conduct extensive experiments on 12 year's worth of US market data with approximately 3,000 companies. The results show that MAPS effectively improves risk-adjusted returns and the diversification of portfolios. Furthermore, we conduct an ablation study and show that adding more agents to our system results in better Sharpe ratios due to further diversification.

2 Problem Statement

In this section, we first introduce the concept of a Markov decision process (MDP) and define how trading decisions are made in a single-agent case. We then extend the single-agent case into a multi-agent case.

2.1 Single-Agent Reinforcement Learning

Single-agent decision-making problems are usually formulated as MDPs. An MDP is defined as a tuple $< s, a, r >$, where s is a finite set of current states, a is a finite set of actions, and r is a reward. The state transition function is omitted for simplicity, since the state transition is not affected by the agent actions in our work. Considering the stochastic and dynamic nature of the financial market, we model stock trading as an MDP as follows:

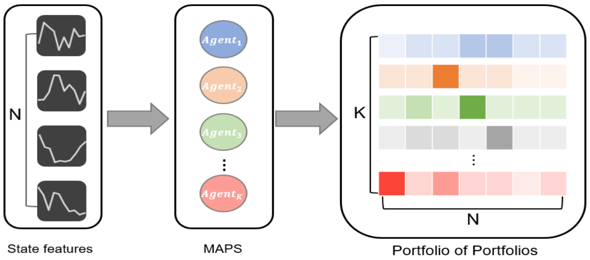

Figure 1: General framework of MAPS.

- State s: a set of features that describes the current state of a stock. In general, different types of information such as historical price movement, trading volume, financial statements, and sentiment scores can be used as the current state. We use a sequence of closing prices of the past f days of a particular company.
- Action a: a set of actions. Our agents can take a long, short, or neutral position.
- Reward $r(s, a)$: a reward based on an agent's action at a current state. In this study, a reward is calculated based on the current action and the next day return of a company.
- Policy $\pi(s, a)$: the trading strategy of an agent. A policy π is essentially a probability distribution over actions given a state s. The goal of an agent is to find the optimal policy which yields maximum cumulative rewards.

2.2 Multi-Agent Reinforcement Learning Extension

The extension of an MDP to the multi-agent case is called a stochastic game which is defined as a tuple $< s, \boldsymbol{a}, \boldsymbol{r} >$. Where s is a finite set of current states and $\boldsymbol{a}$ is a joint action set $\boldsymbol{a} = a_1 \times ... \times a_k$ of K agents. The rewards $\boldsymbol{r} = \{ r_1, ..., r_k \}$ also depend on current state s and joint action $\boldsymbol{a}$ of all agents. Like the single-agent case, the state transition function is omitted in the multi-agent case.

In the fully cooperative MARL, the goal of the agents is to find the optimal joint policy $\boldsymbol{\pi}$ (s, $\boldsymbol{a}$) to maximize the cumulative rewards $\boldsymbol{r}$ of all agents. However, there are two fundamental issues in MARL: the *curse of dimensionality* and the *non-stationarity problem*.

With each additional agent, the joint action space exponentially grows. For example, if one agent can take three total actions (i.e. Long, Neutral, and Short), having ten agents would lead to a total of 3^{10} actions. As a result, it becomes more and more difficult to find the optimal joint policy in MARL as the number of agents increases.

In addition, portfolios are comprised of various companies, and an action is typically taken for each company. Consider-

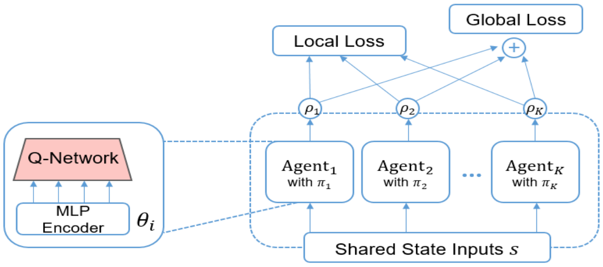

Figure 2: Agent networks and reward strcutrue of MAPS.

ing the combination of all actions for all companies causes the corresponding action space to become exponentially large.

Furthermore, in MARL it is also difficult to find the optimal policy of an agent because all agents learn in conjunction, and consequently the optimal policy of an agent changes as the policy of the other agents change.

Therefore, addressing the proper way to handle the curse of dimensionality problem and designing an appropriate reward structure are central problems in MARL. In the next section, we introduce how we handle these problems and consequently how we can effectively guide the agents to act differently from one another while maximizing their own returns.

3 Methods

3.1 MAPS Architecture

In this section, we describe the overall architecture of MAPS which is illustrated in Figure 2. In MAPS, all of our K agents are trained via Deep Q-learning [Mnih *et al.*, 2015]. Each agent consists of an MLP encoder and Q-network, with structures varying from agent to agent. The input to each agent is a shared state s which is a vector of length f. Each s consists of the normalized closing price sequences of the past f days. The output vector of each agent, ρ, is a vector of length 3 with each element representing the expected long term reward of actions Long, Neutral, and Short, respectively, given the current state s. Therefore, an MLP encoder maps raw state features provided from the environment (i.e. the closing price sequence of a company) into an action value.

To handle the curse of dimensionality and the non-stationarity problem mentioned in the previous section, we use following methods. First, when calculating the action values of agent i the other agents' actions are ignored. Doing so limits the possible number of total actions to three (i.e. Long, Neutral, and Short). Second, each agent maintains two MLP network parameter sets, θ and θ^*. The network parameter set θ is used when performing the gradient step to minimize loss, and the target network parameter set θ^* is simply a copy of θ, and is updated periodically to handle the non-stationarity problem due to the changes of policies of other agents during training. We also adopted experience replay [Mnih *et al.*, 2015] to reduce correlation between subsequent episodes.

The overall training procedure is as follows. For each iteration, the episode for each agent is sampled using an ϵ-greedy policy [Sutton and Barto, 2018] from a training data set of size $N \times T$, stored in a memory buffer of size $K \times M$, where N, T, and M each indicates the number of companies, the number of days, and the size of the memory buffer of each agent. Then, a batch of size $K \times \beta$ is sampled from the memory buffer to calculate the loss. Finally, the gradient step is performed to minimize loss with respect to the parameters θ. θ is copied to θ^* after every C iterations ($C \in \mathbb{Z}$).

3.2 Shared State Memory Buffer

The first step in our training procedure is to sample an episode e_i^m from the training data and to store it in the memory buffer. Unlike the single-agent case, the memory buffer is a $K \times M$ matrix where K is the number of agents.

An episode e_i^m is a tuple defined as e_i^m = $< s_c^t$, $a_{i,c}^t$, $r_{i,c}^t$, $s_c^{t+1} >$, where i and m denote the index of an agent and the column index of the memory buffer. s_c^t and $a_{i,c}^t$ each refer to the current state of company c at time t and the action chosen by the ϵ-greedy policy of agent i given current state s_c^t, and $r_{i,c}^t$ and s_c^{t+1} each refer to the immediate reward received by agent i and the subsequent state of company c at time $t+1$. Note that there is no subscript index for the agents in s_c^t and s_c^{t+1}. This means that the same input state is stored in the same column in the memory buffer.

An action $a_{i,c}^t$ and reward $r_{i,c}^t$ are defined as follows.

$$a_{i,c}^t = 1 - \text{argmax}\{\rho_{i,c}^t\} \tag{1}$$

$$r_{i,c}^t = a_{i,c}^t \times R_c^t \tag{2}$$

where $\rho_{i,c}^t$ and R_c^t refer to the output vector of agent i given input s_c^t and the daily return of company c between time t and time $t+1$ represented in percentage. Therefore, the value 1, 0, or -1 is assigned to action $a_{i,c}^t$ for Long, Neutral, or Short actions, respectively.

The next step is to sample random batches of size $K \times \boldsymbol{\beta}$ from the memory buffer to calculate loss. To formulate the procedure, we define a sampled batch as a $K \times \boldsymbol{\beta}$ matrix and define a vector r of length $\boldsymbol{\beta}$. At every iteration, a random integer value in the range $[0, \boldsymbol{\beta})$ is sampled and assigned to vector r. Then the element at the ith row and bth column in the batch matrix is assigned as: $h_i^b \leftarrow e_i^{r_b}$, where r_b indicates bth element in vector r.

The intuition behind this sampling method is to share the same input state sequence among agents in each batch. Since vector r is re-sampled every iteration rather than by each agent, the same column index sequence is sampled from the memory buffer for each agent. Consequently, as shown in Figure 2, every agent is trained using the same input sequence and we can therefore guide the agents to act different from each other despite being given identical input sequences.

3.3 Loss Function

As previously mentioned, our goal is to guide the agents in MAPS to act as diversely as possible while maximizing their own rewards. To achieve these two contradicting goals, we design our loss function to have two components, namely a *local loss*, and a *global loss*. The *local loss* of each agent is calculated based only on the reward and action value of a particular agent. We first define $LLoss_i^b$ of agent i calculated using a single episode at the ith row and bth column in the batch matrix.

$$LLoss_i^b = \left[Q(s_i, a_i; \theta_i) - r + \gamma \max_{a_i'} Q(s_i', a_i'; \theta_i^*)\right]^2 \quad (3)$$

where s_i, a_i, r_i, s_i', and a_i' each indicate the current state, current action, immediate reward, next state, and next action, respectively, and Q refers to the action-value function. These values are obtained from episode h_i^b in the batch matrix. Note that while choosing action a_i' given state s_i', the target network of agent i parameterized by θ_i^* is used to avoid the moving target problem. We get *local loss* by summing up the $LLoss_i^b$ over batch size $\boldsymbol{\beta}$ as follows:

$$LLoss_i = \sum_{b=1}^{\beta} LLoss_i^b \quad (4)$$

However, it is not possible an agent to be aware of the actions of other agents with the local reward alone. Therefore, the *global loss* provides additional guidance to our agents. We define the *positional confidence* score of agent i for company c calculated using a single episode at the ith row and bth column of the batch matrix as follows:

$$\eta_{i,c}^b = \rho_{i,c}[\text{Long}] - \rho_{i,c}[\text{Short}] \quad (5)$$

where $\rho_{i,c}$ is the output vector of agent i given input s_c. Since the elements of $\rho_{i,c}$ each represent the actions of agent i, respectively, $\eta_{i,c}^b$ represents the ith agent's confidence of how much company c's price will rise at the subsequent time step. By concatenating the calculated positional confidence scores, we get a positional confidence vector of agent i:

$$\eta_i = \left[\eta_i^1, ..., \eta_i^\beta\right]^T \quad (5)$$

We penalize similar behavior among the agents by minimizing the correlation of positional confidence vectors between agents. Formally, the *global loss* can be expressed as:

$$GLoss_i = \sum_{i=1, i \neq j}^{K} \left[\text{Corr}(\eta_i, \eta_j^*)\right]^2 \quad (6)$$

Note that while creating a positional confidence vector of agent j for a agent i, we use the target network parameterized by θ_j^* to mitigate the effect of the non-stationarity problem.

Finally, our total loss is a weighted sum of the *local loss* and the *global loss*.

$$Loss_i = (1 - \lambda)LLoss_i + \lambda GLoss_i \quad (7)$$

where λ is a hyperparameter with a value within $[0, 1]$. The training procedure is summarized in **Algorithm 1**. The value of *maxiter*, $\boldsymbol{\beta}$ and C are 400,000, 128, and 1000, respectively.

Algorithm 1 Training algorithm

```
for $maxiter$ do
    Store experience $e_i^m$ in the memory buffer.
    Sample batch of size $K \times \boldsymbol{\beta}$ from the memory buffer.
    Calculate $Loss_i$ for each agent.
    Perform gradient descent to minimize $Loss_i$ w.r.t. the parameters $\theta_i$ for each agent.
    Copy $\theta_i^* \leftarrow \theta_i$ at every $C$ iterations for each agent.
end for
```

	Period	N	#Data
Training	2000-2004	1534	1876082
Validation	2004-2006	1651	779272
Test	2006-2018	2061	6019248

Table 1: The statistics of dataset

3.4 Portfolio of Portfolios

When training is finished, each of our agents is expected to output an action value. We create a final portfolio vector $\boldsymbol{\alpha^t}$ at time t by summing the portfolio vectors of each agent. The portfolio vector of agent i at time t (i.e. α_i^t) is a vector of length N, which satisfies $\sum_{c=1}^{N} |\alpha_{i,c}^t| = 1$, where $\alpha_{i,c}^t$ represents the cth element in the vector α_i^t. Thus, each $\alpha_{i,c}^t$ represents the weight assigned to company c at time t by agent i. We use the positional confidence score $\eta_{i,c}$ to create the portfolio vector of agent i at time t as follows:

$$\alpha_{i,c}^t = \frac{\eta_{i,c}^t}{\sum_{c=1}^{N} |\eta_{i,c}^t|} \quad (8)$$

Note that superscript t is added to $\eta_{i,c}$ (i.e. $\eta_{i,c}^t$) since the test is proceeded on a test set size of $N \times T$, not on the batch. The final portfolio vector $\boldsymbol{\alpha^t}$ is calculated as follows.

$$\boldsymbol{\alpha_c^t} = \frac{\sum_{i=1}^{K} \alpha_{i,c}^t}{K} \quad (9)$$

where $\boldsymbol{\alpha_c^t}$ represents the cth element in vector $\boldsymbol{\alpha^t}$. Finally, the portfolio vector $\boldsymbol{\alpha^t}$ is normalized to satisfy $\sum_{c=1}^{N} |\boldsymbol{\alpha_c^t}| = 1.0$.

4 Experiments

4.1 Experimental Settings

Dataset We collected roughly 18 year's worth of daily closing price data of approximately 3,000 US companies. Specifically, we used the list of companies from the Russell 3000 index.

We divided our dataset into training set validation set and test set. Detailed statistics of our dataset are summarized in Table 1. The validation set is used to optimize the hyperparameters.

States & Hyperparameters Among many possible candidates, we gave our agents raw historical closing prices as state description features. However, it is worth noting that our framework is not restricted to certain types of state features, and other kinds of features such as technical indicators or sentiment scores can also be used. We expect further diversification to occur if various sources of information were to be provided to MAPS, and leave this as an open question for future work.

MAPS@*k* is our proposed model with k agents in the system. k is an arbitrary hyperparameter and we choose among the values [4, 8, 16] for our experiments to show the effect of using different numbers of agents.

To explain the structure of the MLP encoder, we take **MAPS@4** as an example. An MLP of size $[32, 16]$ represents agent #1, and each subsequent agent has an extra layer with double the hidden units. For example, agent #2 would be an MLP of size $[64, 32, 16]$. **MAPS@8** and **MAPS@16** are simply structures where this pattern is repeated two and four times, respectively.

Batch normalization [Ioffe and Szegedy, 2015] is used after every layer except the final layer and Adam optimizer [Kingma and Ba, 2014] was used with a learning rate of 0.00001 to train our models. The value of λ was empirically chosen as 0.8 based on the validation set.

Evaluation Metric We measure profitability of methods with Return and Sharpe ratio.

- **Return** We calculated the daily return of our portfolio $\boldsymbol{\alpha^t}$ as follows:

$$R_t^{\alpha} = 100 \times \sum_{c=1}^{N} \left(\frac{p_c^{t+1} - p_c^{t}}{p_c^{t}} \right) \cdot \boldsymbol{\alpha_c^t} \qquad (5.1)$$

where p_t^c denotes the closing price of stock c at time t.

- **Sharpe Ratio** The annualized Sharpe ratio is used to measure the performance of an investment compared to its risk. The ratio calculates the excess earned return to the risk-free rate per unit of volatility (risk) as follows:

$$\text{Sharpe} = \sqrt{252} \times \frac{\mathrm{E}[R_t^{\alpha} - R_t^{f}]}{\mathrm{std}[R_t^{\alpha} - R_t^{f}]} \qquad (5.2)$$

where R_t^f is daily risk-free rate at time t and 252 is the number of business days in a year.

Baselines We compare MAPS with the Russell 3000, which is one of the major indices, and the following baselines:

- **Momentum (MOM)** is an investment strategy based on the belief that current market trends will continue. We use the simplest version of the strategy: the last 10-day price movements are used as momentum indicators.
- **Mean-Reversion (MR)** strategy works on the assumption that there is a stable underlying trend line and the price of an asset changes randomly around this line. MR believes that asset prices will eventually revert to the long-term mean. The 30-day moving average is used as the mean reversion indicator.
- **MLP, CNN** Among many existing stock movement forecasting methods, we chose these two models as our forecast baselines as they are widely used in stock forecasting [Di Persio and Honchar, 2016]. The MLP model in our experiments consists of five hidden layers with sizes of [256, 128, 64, 32, 16]. The CNN model has four convolutional layers with [16, 16, 32, 32] filters and one fully connected layer of size [32] is used. Max-pooling layers are applied after the second and fourth layers. Batch normalization is applied for both models. Both models have one additional prediction layer with a softmax function and are trained with 3-label (i.e. up, neutral, down) cross-entropy loss.

Period	2006-2012		2012-2018	
Models	Return	Sharpe	Return	Sharpe
MOM	3.938	1.149	-3.223	-1.198
MR	-2.262	-0.899	2.220	0.816
MLP	16.377	1.309	1.744	0.368
CNN	17.036	1.093	-3.294	-0.442
DA-RNN	11.860	3.283	4.309	2.113
MAPS@4	17.955	4.829	4.846	2.121
MAPS@8	22.744	4.751	**6.123**	2.175
MAPS@16	**23.467**	**5.547**	5.567	**2.247**

Table 2: Experimental results on the Russell 3000 companies.

Models	2006-2012	2012-2018
MAPS@4	0.3415	0.3456
MAPS@8	0.4622	0.4424
MAPS@16	0.2318	0.2429

Table 3: Average correlation of daily return of each agents. A smaller correlation indicates more independent actions of agents.

- **DA-RNN** refers to the dual-stage attention-based RNN [Qin *et al.*, 2017]. Is is the state-of-the-art and attention mechanisms are used in each stage to identify relevant input features and select relevant encoder hidden states. As the DA-RNN model was originally designed to forecast time series signals, we also trained our model to predict future prices of the assets with mean squared error. A portfolio is created based on the expected return of the predicted asset prices.

4.2 Results

Performance analysis Our experiment results are summarized in Table 2, and Figure 3 illustrates a comparison of cumulative wealth based on the portfolios created by each model. MAPS outperformed all baselines in terms of both annualized return and Sharpe ratio. Some interesting findings are as follows:

- The performance of traditional strategies like MOM and MR vary based on market conditions and generally perform poorly. As these strategies use a single rule leveraging only certain aspects of market dynamics, their performance is not robust as the market evolves.
- Forecast-based methods show better performance than traditional approaches in terms of annualized return. Naturally, the performance of forecast-based methods heavily relies on the prediction accuracy of the model. The MLP and CNN perform better in general but did not always outperform the traditional methods. Only DA-RNN performed consistently better in both annualized return and Sharpe ratio for both testing periods.
- MAPS outperformed all baselines in our experiments. **It is worth noting** that MAPS shows a better Sharpe

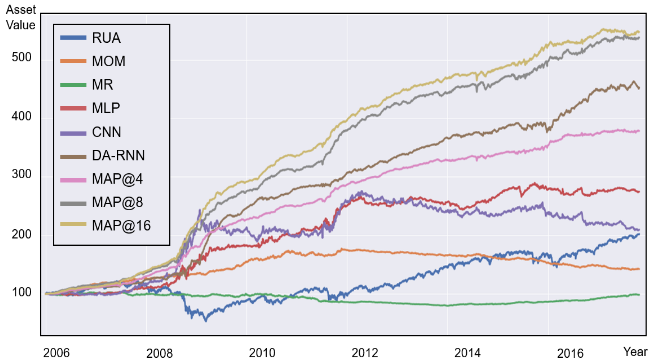

Figure 3: Comparison of cumulative wealth of different models (RUA indicates the Russell 3000 index). All asset values are set to 100 at the beginning of the test period.

ratio even when the return is similar. This proves the effectiveness of diversification with multiple agents in the perspective of risk-adjusted return. We further observe that MAPS with more agents obtains a better Sharpe ratio.

- One unexpected result is that the Sharpe ratio does not scale linearly with the number of agents. We can interpret this with the daily return correlation scores of different MAPS, summarized in Table 3. If our agents act diversely, the correlation of daily return would be small. In table 3 we can find that the average correlation of MAPS@8 is higher than MAPS@4. The agents of MAPS@8 act more similarly to each other than those of MAPS@4, resulting in lower Sharpe ratio despite higher returns. Further improvement in our learning scheme may solve this issue and we leave this for future work.

Effect of *global loss* To investigate the effect of a *global loss*, we compare the learning process of MAPS with and without *global loss*. During our learning process, we calculate the correlation of a positional confidence score η_i between all agents for the entire validation set. We calculate this value every 10,000 training iterations and average for all companies and pairs of agents. As the positional confidence value indicates the type of action taken by the agents, higher correlation means more similar actions among the agents. As we can see in Figure 4., average correlation values of MAPS without *global loss* increase rapidly and converge with much higher values than MAPS with *global loss*. In contrast, the correlations of MAPS trained with *global loss* increased slowly and resulted in having small values. The results verify the effectiveness of *global loss* in making agents act independently.

Case Study To better understand how our agents act differently with identical state features, we illustrate an example portfolio in Figure 5. The black line is a movement of Amazon's stock price in 2016. The colored rectangle at the bottom of the figure describes the actions of our agents. In this case, we have eight agents in MAPS and each line represents which positions were taken by which agent, with each color representing a position. Red, grey, and blue each refer to long, neutral, and short positions taken at a given time. What we can observe here is that the agents in our system make different decisions based on their own understanding of the market. For instance, in the spring of 2014 (the period we outlined with the bright white box), the future movement of Amazon stock price seems volatile and uncertain after a steep fall and several price corrections. Two out of eight agents decided to take long positions betting that the future price would rise, and two agents chose a short position with the opposite prospect. This kind of discrepancy in actions is prevalent throughout the trading process, making our portfolio as a whole sufficiently diversified.

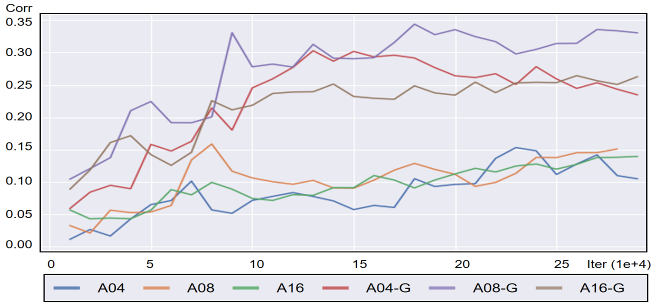

Figure 4: Illustrations of how agents in MAPS choose different actions given same state features.

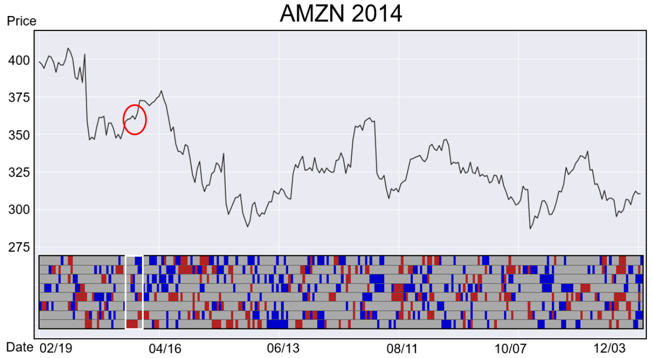

Figure 5: Illustrations of how agents in MAPS choose different actions given same state features.

5 Conclusion

In this work, we propose MAPS, a cooperative Multi-Agent reinforcement learning-based Portfolio management System. The agents in MAPS act as differently as possible while maximizing their own reward guided by our proposed loss function. Experiments with 12 years of US market data show that MAPS outperforms most of the existing baselines in terms of Sharpe ratio. We also presented the effectiveness of our learning scheme and how our agents' independent actions end up with a diversified portfolio with detailed analysis.

Acknowledgements

This work was supported by the National Research Foundation of Korea (NRF-2017R1A2A1A17069645, NRF-2017M3C4A7065887).

References

[Abarbanell and Bushee, 1997] Jeffrey S Abarbanell and Brian J Bushee. Fundamental analysis, future earnings, and stock prices. *Journal of Accounting Research*, 35(1):1–24, 1997.

[Bacoyannis *et al.*, 2018] Vangelis Bacoyannis, Vacslav Glukhov, Tom Jin, Jonathan Kochems, and Doo Re Song. Idiosyncrasies and challenges of data driven learning in electronic trading. *arXiv preprint arXiv:1811.09549*, 2018.

[Deng *et al.*, 2016] Yue Deng, Feng Bao, Youyong Kong, Zhiquan Ren, and Qionghai Dai. Deep direct reinforcement learning for financial signal representation and trading. *IEEE transactions on neural networks and learning systems*, 28(3):653–664, 2016.

[Di Persio and Honchar, 2016] Luca Di Persio and Oleksandr Honchar. Artificial neural networks architectures for stock price prediction: Comparisons and applications. *International journal of circuits, systems and signal processing*, 10(2016):403–413, 2016.

[Ding *et al.*, 2015] Xiao Ding, Yue Zhang, Ting Liu, and Junwen Duan. Deep learning for event-driven stock prediction. In *Twenty-fourth international joint conference on artificial intelligence*, 2015.

[Feng *et al.*, 2019] Fuli Feng, Huimin Chen, Xiangnan He, Ji Ding, Maosong Sun, and Tat-Seng Chua. Enhancing stock movement prediction with adversarial training. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, pages 5843–5849. AAAI Press, 2019.

[Ioffe and Szegedy, 2015] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[Jegadeesh and Titman, 1993] Narasimhan Jegadeesh and Sheridan Titman. Returns to buying winners and selling losers: Implications for stock market efficiency. *The Journal of finance*, 48(1):65–91, 1993.

[Kaya *et al.*, 2016] Orçun Kaya, Jan Schildbach, and Deutsche Bank Ag. High-frequency trading. *Reaching the limits, Automated trader magazine*, 41:23–27, 2016.

[Kim *et al.*, 2019] Raehyun Kim, Chan Ho So, Minbyul Jeong, Sanghoon Lee, Jinkyu Kim, and Jaewoo Kang. Hats: A hierarchical graph attention network for stock movement prediction. *arXiv preprint arXiv:1908.07999*, 2019.

[Kingma and Ba, 2014] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[Lo *et al.*, 2000] Andrew W Lo, Harry Mamaysky, and Jiang Wang. Foundations of technical analysis: Computational algorithms, statistical inference, and empirical implementation. *The journal of Finance*, 55(4):1705–1765, 2000.

[Markowitz, 1952] Harry Markowitz. Portfolio selection. *The journal of finance*, 7(1):77–91, 1952.

[Mnih *et al.*, 2015] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[Park and Irwin, 2007] Cheolho Park and Scott H Irwin. What do we know about the profitability of technical analysis? *Journal of Economic Surveys*, 21(4):786–826, 2007.

[Poterba and Summers, 1988] James M Poterba and Lawrence H Summers. Mean reversion in stock prices: Evidence and implications. *Journal of financial economics*, 22(1):27–59, 1988.

[Qin *et al.*, 2017] Yao Qin, Dongjin Song, Haifeng Chen, Wei Cheng, Guofei Jiang, and Garrison Cottrell. A dual-stage attention-based recurrent neural network for time series prediction. *arXiv preprint arXiv:1704.02971*, 2017.

[Sutton and Barto, 2018] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. 2018.

[Xiong *et al.*, 2018] Zhuoran Xiong, Xiao-Yang Liu, Shan Zhong, Hongyang Yang, and Anwar Walid. Practical deep reinforcement learning approach for stock trading. *arXiv preprint arXiv:1811.07522*, 2018.

[Xu and Cohen, 2018] Yumo Xu and Shay B Cohen. Stock movement prediction from tweets and historical prices. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1970–1979, 2018.