Practical Deep Reinforcement Learning Approach for Stock Trading

Zhuoran Xiong*, **Xiao-Yang Liu***, **Shan Zhong***, **Hongyang (Bruce) Yang**+, **and Anwar Walid**†
*Electrical Engineering, Columbia University,
+Department of Statistics, Columbia University,
†Mathematics of Systems Research Department, Nokia-Bell Labs
Emails: {ZX2214, XL2427, SZ2495, HY2500}@columbia.edu,
anwar.walid@nokia-bell-labs.com

Abstract

Stock trading strategy plays a crucial role in investment companies. However, it is challenging to obtain optimal strategy in the complex and dynamic stock market. We explore the potential of deep reinforcement learning to optimize stock trading strategy and thus maximize investment return. 30 stocks are selected as our trading stocks and their daily prices are used as the training and trading market environment. We train a deep reinforcement learning agent and obtain an adaptive trading strategy. The agent's performance is evaluated and compared with Dow Jones Industrial Average and the traditional min-variance portfolio allocation strategy. The proposed deep reinforcement learning approach is shown to outperform the two baselines in terms of both the Sharpe ratio and cumulative returns.

1 Introduction

Profitable stock trading strategy is vital to investment companies. It is applied to optimize allocation of capital and thus maximize performance, such as expected return. Return maximization is based on estimates of stocks' potential return and risk. However, it is challenging for analysts to take all relavant factors into consideration in complex stock market [1–3].

One traditional approach is performed in two steps as described in [4]. First, the expected returns of the stocks and the covariance matrix of the stock prices are computed. The best portfolio allocation is then found by either maximizing the return for a fixed risk of the portfolio or minimizing the risk for a range of returns. The best trading strategy is then extracted by following the best portfolio allocation. This approach, however, can be very complicated to implement if the manager wants to revise the decisions made at each time step and take, for example, transaction cost into consideration. Another approach to solve the stock trading problem is to model it as a Markov Decision Process (MDP) and use dynamic programming to solve for the optimum strategy. However, the scalability of this model is limited due to the large state spaces when dealing with the stock market [5–8].

Motivated by the above challenges, we explore a deep reinforcement learning algorithm, namely Deep Deterministic Policy Gradient (DDPG) [9], to find the best trading strategy in the complex and dynamic stock market. This algorithm consists of three key components: (i) actor-critic framework [10] that models large state and action spaces; (ii) target network that stabilizes the training process [11]; (iii) experience replay that removes the correlation between samples and increases the usage

NIPS 2018 Workshop on Challenges and Opportunities for AI in Financial Services: the Impact of Fairness, Explainability, Accuracy, and Privacy, Montréal, Canada. This is a non-archival publication - the authors may submit revisions and extensions of this paper to other publication venues.

of data. The efficiency of DDPG algorithm is demonstrated by achieving higher return than the traditional min-variance portfolio allocation method and the Dow Jones Industrial Average[1] (DJIA).

This paper is organized as follows. Section 2 contains statement of our stock trading problem. In Section 3, we drive and specify the main DDPG algorithm. Section 4 describes our data preprocessing and experimental setup, and presents the performance of DDPG algorithm. Section 5 gives our conclusions.

2 Problem Statement

We model the stock trading process as a Markov Decision Process (MDP). We then formulate our trading goal as a maximization problem.

2.1 Problem Formulation for Stock Trading

Considering the stochastic and interactive nature of the trading market, we model the stock trading process as a Markov Decision Process (MDP) as shown in Fig. 1, which is specified as follows:

No external indicators

- State $s = [p, h, b]$: a set that includes the information of the prices of stocks $p \in \mathbb{R}_+^D$, the amount of holdings of stocks $h \in \mathbb{Z}_+^D$, and the remaining balance $b \in \mathbb{R}_+$, where D is the number of stocks that we consider in the market and $\mathbb{Z}_+$ denotes non-negative integer numbers.
- Action a: a set of actions on all D stocks. The available actions of each stock include selling, buying, and holding, which result in decreasing, increasing, and no change of the holdings h, respectively.
- Reward $r(s, a, s')$: the change of the portfolio value when action a is taken at state s and arriving at the new state s'. The portfolio value is the sum of the equities in all held stocks $p^T h$ and balance b.
- Policy $\pi(s)$: the trading strategy of stocks at state s. It is essentially the probability distribution of a at state s.
- Action-value function $Q_\pi(s, a)$: the expected reward achieved by action a at state s following policy π.

The dynamics of the stock market is described as follows. We use subscript to denote time t, and the available actions on stock d are

- Selling: k ($k \in [1, h[d]]$, where $d = 1, ..., D$) shares can be sold from the current holdings, where k must be an integer. In this case, $h_{t+1} = h_t - k$.
- Holding: $k = 0$ and it leads to no change in h_t.
- Buying: k shares can be bought and it leads to $h_{t+1} = h_t + k$. In this case $a_t[d] = -k$ is a negative integer.

It should be noted that all bought stocks should not result in a negative balance on the portfolio value. That is, without loss of generality we assume that selling orders are made on the first d_1 stocks and the buying orders are made on the last d_2 ones, and that a_t should satisfy $p_t[1 : d_1]^T a_t[1 : d_1] + b_t + p_t[D - d_2 : D]^T a_t[D - d_2 : D] \geq 0$. The remaining balance is updated as $b_{t+1} = b_t + p_t^T a_t$. Fig. 1 illustrates this process. As defined above, the portfolio value consists of the balance and sum of the equities in all held stocks. At time t, an action is taken, and based on the executed action and the updates of stock prices, the portfolio values change from "portfolio value 0" to "portfolio value 1", "portfolio value 2", or "portfolio value 3" at time $(t + 1)$.

Before being exposed to the environment, p_0 is set to the stock prices at time 0 and b_0 is the initial fund available for trading. The h and $Q_\pi(s, a)$ are initialized as 0, and $\pi(s)$ is uniformly distributed among all actions for any state. Then, $Q_\pi(s_t, a_t)$ is learned through interacting with the external environment.

[1]The Dow Jones Industrial Average is a stock market index that shows how 30 large, publicly owned companies based in the United States have traded during a standard trading session in the stock market.

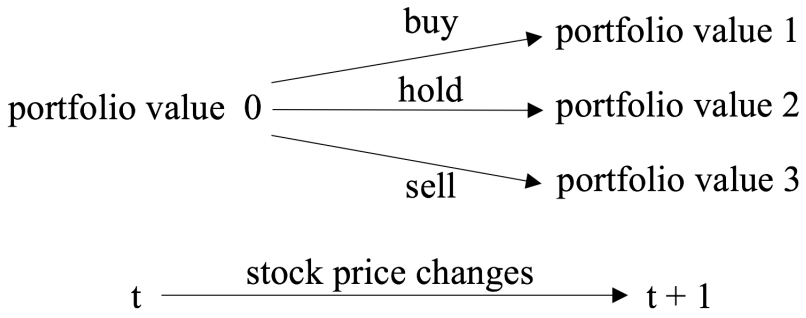


Figure 1: One starting portfolio value with three actions leading to three possible portfolio values where actions have probabilities that sum up to one. Note that "hold" can lead to different portfolio values if the stock prices change.

According to Bellman Equation, the expected reward of taking action a_t is calculated by taking the expectation of the rewards $r(s_t, a_t, s_{t+1})$, plus the expected reward in the next state s_{t+1}. Based on the assumption that the returns are discounted by a factor of γ, we have

Typical equation →

$$Q_\pi(s_t, a_t) = \mathbb{E}_{s_{t+1}}[r(s_t, a_t, s_{t+1}) + \gamma \mathbb{E}_{a_{t+1}\sim\pi(s_{t+1})}[Q_\pi(s_{t+1}, a_{t+1})]]. \quad (1)$$

2.2 Trading Goal as Return Maximization

The goal is to design a trading strategy that maximizes the investment return at a target time t_f in the future, i.e., $p_{t_f}^T h_t + b_{t_f}$, which is also equivalent to $\sum_{t=1}^{t_f-1} r(s_t, a_t, s_{t+1})$. Due to the Markov property of the model, the problem can be boiled down to optimizing the policy that maximizes the function $Q_\pi(s_t, a_t)$. This problem is very hard because the action-value function is unknown to the policy maker and has to be learned via interacting with the environment. Hence in this paper, we employ the deep reinforcement learning approach to solve this problem.

3 A Deep Reinforcement Learning Approach

We employ a DDPG algorithm to maximize the investment return. DDPG is an improved version of Deterministic Policy Gradient (DPG) algorithm [12]. DPG combines the frameworks of both Q-learning [13] and policy gradient [14]. Compared with DPG, DDPG uses neural networks as function approximator. The DDPG algorithm in this section is specified for the MDP model of the stock trading market.

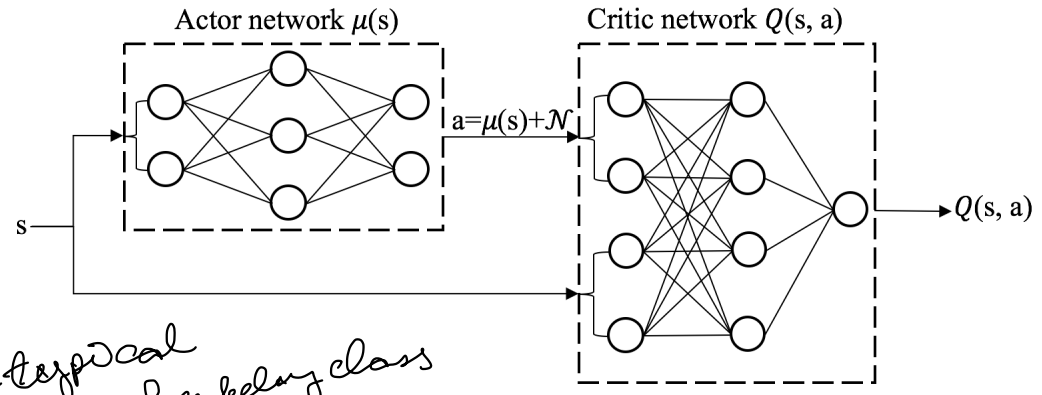
The Q-learning is essentially a method to learn the environment. Instead of using the expectation of $Q(s_{t+1}, a_{t+1})$ to update $Q(s_t, a_t)$, Q-learning uses greedy action a_{t+1} that maximizes $Q(s_{t+1}, a_{t+1})$ for state s_{s+1}, i.e.,

$$Q_\pi(s_t, a_t) = \mathbb{E}_{s_{t+1}}[r(s_t, a_t, s_{t+1}) + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1})]. \quad (2)$$

With Deep Q-network (DQN), which adopts neural networks to perform function approximation, the states are encoded in value function. The DQN approach, however, is intractable for this problem due to the large size of the action spaces. Since the feasible trading actions for each stock is in a discrete set, and considering the number of total stocks, the sizes of action spaces grow exponentially, leading to the "curse of dimensionality" [15]. Hence, the DDPG algorithm is proposed to deterministically map states to actions to address this issue.

As shown in Fig. 2, DDPG maintains an actor network and a critic network. The actor network $\mu(s|\theta^\mu)$ maps states to actions where θ^μ is the set of actor network parameters, and the critic network $Q(s, a|\theta^Q)$ outputs the value of action under that state, where θ^Q is the set of critic network parameters. To explore better actions, a noise is added to the output of the actor network, which is sampled from a random process $\mathcal{N}$.

Similar to DQN, DDPG uses an experience replay buffer R to store transitions and update the model, and can effectively reduce the correlation between experience samples. Target actor network



Looks like a typical Actor-Critic from Berkeley class

Figure 2: Learning network achitecture.

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with random weight θ^Q and θ^μ;
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$;
Initialize replay buffer R;
for episode= 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration;
  Receive initial observation state s_1;
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise;
    Execute action a_t and observe reward r_t and state s_{t+1};
    Store 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}) from R;
    Set $y_i = r_i + \gamma Q'(s_{t+1}, \mu'(s_{i+1}|\theta^{\mu'}|\theta^{Q'}))$;
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$;
    Update the actor policy by using the sampled policy gradient:

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

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'},$$
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}.$$

  end for
end for

Q' and μ' are created by copying the actor and critic networks respectively, so that they provide consistent temporal difference backups. Both networks are updated iteratively. At each time, the DDPG agent takes an action a_t on s_t, and then receives a reward based on s_{t+1}. The transition (s_t, a_t, s_{t+1}, r_t) is then stored in replay buffer R. The N sample transitions are drawn from R and $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'}, \theta^{Q'})), i = 1, \cdots, N$, is calculated. The critic network is then updated by minimizing the expected difference $L(\theta^Q)$ between outputs of the target critic network Q' and the critic network Q, i.e,

$$L(\theta^Q) = \mathbb{E}_{s_t, a_t, r_t, s_{t+1} \sim \text{buffer}}[(r_t + \gamma Q'(s_{t+1}, \mu(s_{t+1}|\theta^\mu)|\theta^{Q'}) - Q(s_t, a_t|\theta^Q))^2]. \quad (3)$$

The parameters θ^μ of the actor network are then as follows:

$$\nabla_{\theta^\mu} J \approx \mathbb{E}_{s_t, a_t, r_t, s_{t+1} \sim \text{buffer}}[\nabla_{\theta^\mu} Q(s_t, \mu(s_t|\theta^\mu)|\theta^Q)] \quad (4)$$
$$= \mathbb{E}_{s_t, a_t, r_t, s_{t+1} \sim \text{buffer}}[\nabla_a Q(s_t, \mu(s_t)|\theta^Q)\nabla_{\theta^\mu}\mu(s_t|\theta^\mu)]. \quad (5)$$

After the critic network and the actor network are updated by the transitions from the experience buffer, the target actor network and the target critic network are updated as follows:

$$\theta^{Q'} \leftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q'}, \tag{6}$$

$$\theta^{\mu'} \leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'}, \tag{7}$$

where τ denotes learning rate. The detailed algorithm is summerized in Algorithm 1.

4 Performance Evaluations

We evaluate the performance of the DDPG algorithm in Alg. 1. The result demonstrates that the proposed method with the DDPG agent achieves higher return than the Dow Jones Industrial Average and the traditional min-variance portfolio allocation strategy [16, 17].

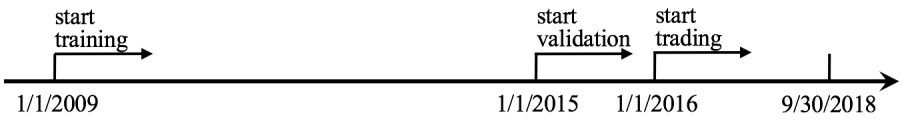


Figure 3: Data spliting.

4.1 Data Preprocessing

We track and select Dow Jones 30 stocks of 1/1/2016 as our trading stocks, and use historical daily prices from 01/01/2009 to 09/30/2018 to train the agent and test the performance. The dataset is downloaded from Compustat database accessed through Wharton Research Data Services (WRDS) [18].

Our experiment consists of three stages, namely training, validation and trading. In the training stage, Alg. 1 generates a well-trained trading agent. The validation stage is then carried out for key parameters adjustment such as learning rate, number of episodes, etc. Finally in the trading stage, we evaluate the profitability of the proposed scheme. The whole dataset is split into three parts for these purposes, as shown in Fig. 3. Data from 01/01/2009 to 12/31/2014 are used for training, and the data from 01/01/2015 to 01/01/2016 are used for validation. We train our agent on both training and validation data to make full use of available data. Finally, we test our agent's performance on trading data, which is from 01/01/2016 to 09/30/2018. To better exploit the trading data, we continue training our agent while in the trading stage as this will improve the agent to better adapt the market dynamics.

4.2 Experimental Setting and Results of Stock Trading

We build the environment by setting 30 stocks data as a vector of daily stock prices over which the DDPG agent is trained. To update the learning rate and number of episodes, the agent is validated on validation data. Finally, we run our agent on trading data and compare performance with Dow Jones Industrial Average (DJIA) and the min-variance portfolio allocation strategy.

Four metrics are used to evaluate our results: final portfolio value, annualized return, annualized standard error and the Sharpe ratio. Final portfolio value reflects portfolio value at the end of trading stage. Annualized return indicates the direct return of the portfolio per year. Annualized standard error shows the robustness of our model. The Sharpe ratio combines the return and risk together to give such evaluation [19].

In Fig. 4, we can see that the DDPG strategy significantly outperforms Dow Jones Industrial Average and the min-variance portfolio allocation. As can be seen from Table 1, the DDPG strategy achieves annualized return 22.24%, which is much higher than Dow Jones Industrial Average's 16.40% and min-variance portfolio allocation's 15.93%. The sharpe ratio of the DDPG strategy is also much higher, indicating that the DDPG strategy beats both Dow Jones Industrial Average and min-variance portfolio allocation in balancing risk and return. Therefore, the result demonstrates that the proposed DDPG strategy can effectively develop a trading strategy that outperforms the benchmark Dow Jones Industrial Average and the traditional min-variance portfolio allocation method.

Validation is same as test dataset in FinRL paper

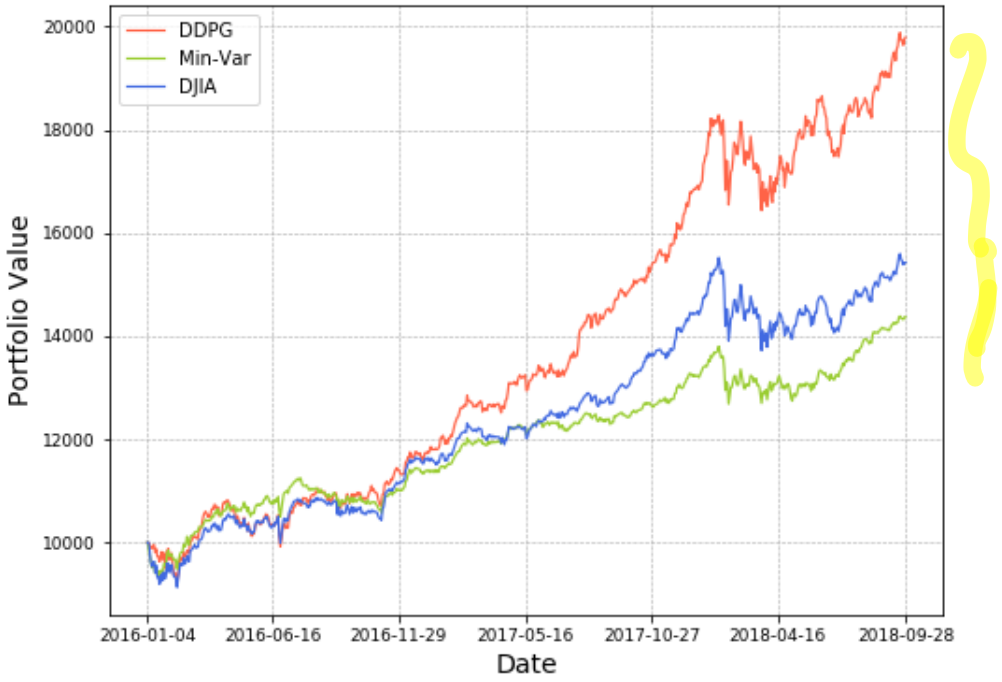


Figure 4: Portfolio value curves of our DDPG scheme, the min-variance portfolio allocation strategy, and the Dow Jones Industrial Average. (Initial portfolio value $\$10,000$).

Table 1: Trading Performance.

	DDPG (ours)	Min-Variance	DJIA
Initial Portfolio Value	$\mathbf{10,000}$	$10,000$	$10,000$
Final Portfolio Value	$\mathbf{19,791}$	$14,369$	$15,428$
Annualized Return	$\mathbf{25.87}\%$	15.93%	16.40%
Annualized Std. Error	$\mathbf{13.62}\%$	9.97%	11.70%
Sharpe Ratio	$\mathbf{1.79}$	1.45	1.27

5 Conclusion

In this paper, we have explored the potential of training Deep Deterministic Policy Gradient (DDPG) agent to learn stock trading strategy. Results show that our trained agent outperforms the Dow Jones Industrial Average and min-variance portfolio allocation method in accumulated return. The comparison on Sharpe ratios indicates that our method is more robust than the others in balancing risk and return.

Future work will be interesting to explore more sophisticated model [20], deal with larger scale data [21], observe intelligent behaviors [22], and incorporate prediction schemes [23].

References

[1] Stelios D. Bekiros, "Fuzzy adaptive decision-making for boundedly rational traders in speculative stock markets," *European Journal of Operational Research*, vol. 202, pp. 285-293, 2010.

[2] Yong Zhang, and Xingyu Yang, "Online portfolio selection strategy based on combining experts' advice," *Computational Economics*, vol. 50, No. 1, pp. 141-159, 2017.

[3] Youngmin Kim, Wonbin Ahn, Kyong Joo Oh, and David Enke, "An intelligent hybrid trading system for discovering trading rules for the futures market using rough sets and genetic algorithms," *Applied Soft Computing*, vol. 55, pp. 127-140, 2017.

[4] Markowitzz, H., "Portfolio selection," *The Journal of Finance*, vol. 7, No. 1, pp. 77-91, 1952.

[5] Dimitri Bertsekas, "Dynamic programming and optimal control," *Athena Scientific,* vol. 1, 1995.

[6] Francesco Bertoluzzoa, and Marco Corazza, "Testing different reinforcement learning configurations for financial trading: introduction and applications," *Procedia Economics and Finance*, vol. 3, pp. 68-77, 2012.

[7] Ralph Neuneier, "Optimum asset allocation using adaptive dynamic programming," *Advances in Neural Information Processing Systems*, vol. 8, 1996.

[8] Ralph Neuneier, "Enhancing Q-Learning for optimal asset allocation," *Advances in Neural Information Processing Systems*, 1997.

[9] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971,* 2015.

[10] Vijay R. Konda and John Tsitsiklis. "Actor-critic algorithms," *Advances in Neural Information Processing Systems*, pp. 1008–1014, 1999.

[11] Volodymyr Mnih, et al, "Human-level control through deep reinforcement learning," *Nature*, pp. 529-533, 2015.

[12] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, Martin Riedmiller, "Deterministic policy gradient algorithms," *International Conference on Machine Learning,* vol. 32, 2014.

[13] Richard S. Sutton and Andrew G. Barto, *Reinforcement learning: an introduction*, MIT Press. 1998.

[14] Richard S. Sutton, et al. "Policy gradient methods for reinforcement learning with function approximation," *Advances in Neural Information Processing Systems*, 2000.

[15] Lucian Buşoniu, Tim de Bruin, Domagoj Tolić, Jens Kober, Ivana Palunko, "Reinforcement learning for control: Performance, stability, and deep approximators," *Annual Reviews in Control,* ISSN 1367-5788, 2018.

[16] "Codes for Min-Variance Portfolio Allocation," http://www.tensorlet.com/.

[17] Hongyang Yang, Xiao-Yang Liu, Qingwei Wu, "A practical machine learning approach for dynamic stock recommendation," *IEEE International Conference On Trust, Security and Privacy in Computing And Communications*, 2018.

[18] Compustat Industrial [daily Data]. Available: Standard Poor's/Compustat [2017]. Retrieved from "Wharton Research Data Service," 2015.

[19] Willia F. Sharpe, "The Sharpe ratio," *The Journal of Portfolio Management*, vol. 1, No. 1, 21, 49-58, 1994.

[20] Lu Wang, Wei Zhang, Xiaofeng He, Hongyuan Zha, "Supervised reinforcement learning with recurrent neural Network for dynamic treatment recommendation," *International Conference on Knowledge Discovery & Data Mining,* pp. 2447-2456, 2018.

[21] Yuri Burda, Harri Edwards, Deepak Pathak, Amos Storkey, Trevor Darrell, Alexei A. Efros, "Large-scale study of curiosity-driven learning," *arXiv:1808.04355,* 2018.

[22] Xiao-Yang Liu, Zihan Ding, Sem Borst, Anwar Walid, "Deep reinforcement learning for intelligent transportation systems," *NeurIPS Workshop on Machine Learning for Intelligent Transportation Systems*, 2018.

[23] Weiju Lu, Xiao-Yang Liu, Qingwei Wu, Yue Sun, Anwar Walid, "Transform-based multilinear dynamical system for tensor time series analysis," *NeurIPS Workshop on Modeling and Decision-Making in the Spatiotemporal Domain*, 2018.