An Evaluation on Deep Reinforcement Learning Algorithms for Stock Trading

Yoshiaki Nejime s1260099 Supervised by Xiang Li

Abstract

In algorithmic trading, the main tasks for long-term profitability are to extract appropriate market features and to design a trading strategy. Deep Reinforcement Learning (DRL), which combines deep learning for feature representation and reinforcement learning for optimizing sequential decision-making problems, can handle these tasks from end to end. Several previous studies have reported obtaining profitable DRL agents even for out-of-sample data, but there are known problems: DRL results have a large variance, and simulated past trading results are prone to over-fitting. This study incorporates a stationary feature transformation considering out-of-sample data and time series cross-validation method for quantitative evaluation. We also compare DRL algorithms developed based on different concepts by performing an algorithmic stock trading. The experimental results show that while each agent is profitable, it does not surpass the holding stock strategy. asd adfdad fadsfadf asdf asdfasdfadsfa fad fad as ad adf ad fasdf asdf adsf asdf asdf adsf ad fasdfasdf asdfasd fasdf sa fasd fasd fas asd fasd fasd fasdf as asdf fae

1 Introduction

Algorithmic trading automatically executes trade orders according to predetermined mathematical rules. In recent years, algorithmic trading has dominated most volume and is an interesting topic for investors. The key tasks in algorithmic trading are the market features and the trading strategy design. A market features are patterns such as a trending market where prices move in one direction or a range market where prices move within a certain range. A trading strategy is a decision-making process, such as whether to buy or sell and how much to bet on a given market feature.

The approaches to solve these tasks are statistical, supervised machine learning (ML), and reinforcement learning (RL). The statistical approach is a conventional method and focuses on means, variances, and autocorrelations. The best known is the mean reversion strategy. For example, the strategy trades on the expectation that when the gap between prices and moving averages becomes large, the price reverts to the average. In supervised ML approach, the motivation is to gain an edge through extracting valuable and hidden market features. It is used to predict the next return or volatility given the recent period price information. With the emergence of powerful ML models such as LightGBM [1] and long short-term memory (LSTM) [2], the accuracy has become higher. However, in the aforementioned approach, the trading strategy requires to be developed separately. The RL approach directly designs trading strategies while extracting appropriate market representations. RL is a framework for optimizing policy to maximize the cumulative reward in environments which follow a Markov decision process (MDP). This means that the investment and learning objectives are the same to increase assets for a long-term. In recent years, with the arrival of deep reinforcement learning (DRL), the framework has the potential to surpass human performance in game playing [3].

The previous studies applying DRL to algorithmic trading [4], [5], [6] and [7] show that they beat the traditional method benchmarks. But as mentioned in [8], since dataset selection and data partitioning are unique, it may lead to over-fitting of backtesting and overestimating the results. This problem is mainly caused by human bias and is more pronounced for the DRL with large variance in experimental results. In order to suppress this, a tight evaluation method is required. On the other hand, DRL algorithms are not often compared in this field, although their network architecture differences are compared. The DRL has been developed to overcome a variety of challenges, and it is not suitable for every task. Thus, it is helpful to know which algorithms are effective in working on the research.

This study compares the performance of major DRL algorithms, Deep-Q-Network (DQN) [3], Advantage Actor-Critic (A2C) [9], Proximal Policy Optimization (PPO) [10], and Soft Actor-Critic (SAC) [11] on simulating stock trading. We also incorporate a stationary feature transformation for improved generalization and a time series cross-validation method for quantitative evaluation.

The main contribution of our work is to provide robust investment performance of primary DRLs against out-of-sample data in algorithmic stock trading.

2 Preliminaries

In this section, we explain the basics of financial investment and deep reinforcement learning.

2.1 Financial Knowledge

The objective of financial investment is to gain continuous profit by repeatedly buying and selling financial assets. Profits are generated by buying at a lower price and selling at a higher price or the reverse. Although it seems easy to do, the market is dynamic and uncertain, so it is difficult to make a profit in the long term unless you are familiar with finance. We introduce some terminology in financial trading.

Return: The return is the percentage change of the price with $r_t = \frac{p_t - p_{t-1}}{p_{t-1}}$, where r_t is the return and p_t is the price at time t.

Long / Short Position: The position, which is a specific term for finance derivatives, represents the amount of financial instrument owned. A long position indicates that an investor is holding a purchase with the expectation of an increase in value. This can be profitable if opened when cheap and closed when higher. A short position is the opposite of a long position, which is profitable when it opens at higher prices and closes at lower prices.

Profit & Loss (PnL): PnL is the earnings through the transaction during a period.

Equity: The equity represents the sum of cash, the position's value, and PnL.

2.2 Deep Reinforcement Learning (DRL)

DRL is a method that combines deep learning (DL) [12] for feature representation and reinforcement learning (RL) [13] for optimizing sequential decision-making problems. DL is a technique based on deep neural networks, which are highly representative function approximators. It is the state of the art in fields such as computer vision and natural language processing. In DRL, it is used for state extraction, solving the limitation of tabular RL in which the state space cannot be expanded.

The goal of RL is to obtain a policy to maximize the cumulative reward while the agent interacts repeatedly with the environment following a Markov decision process (MDP), which is denoted by the four tuple $M = (S, A, T, R)$. The interaction is described in this way. At each time step t, the agent receives a state $s_t \in S$ from the environment and takes an action $a_t \in A$ according to the policy $\pi(a|s)$. Based on the action, the environment gives the agent the next state $s_{t+1} \sim T(s_t, a_t)$ and reward $r_{t+1} = R(s_t, a_t, s_{t+1})$.

RL is mainly categorized into two types: value-based and policy-based. The policy-based method defines an objective function with parameterized policy and uses the policy gradient theorem to update the parameters. The value-based method defines and updates an action-value function, also called Q-function, instead of directly learning the policy. The actor-critic method is also a hybrid that uses both. While it takes longer to learn, it stabilizes the learning process.

On-policy and off-policy are important concepts in policy optimization. On-policy is learning to improve a policy directly based on the experience gained with the current policy. Off-policy is learning to use the experience gained from the behavior policy to improve the target policy. The behavioral policy is updated by the target policy at certain intervals. Off-policy tends to have larger variance and slower convergence, but it is more sample efficient because it can use the experience gathered from previous policies for learning.

3 Method

This section first presents the target stock data and feature extraction. Thereafter, we introduce the trading environment formalized in MDP and the DRL agents to be traded. Finally, we describe the experimental methods: time series cross-validation and hyperparameter tuning.

3.1 Dataset

The daily historical price data from 2010 to 2021 provided by Yahoo Finance is used. The data includes opening price, highest price, lowest price, closing price and volume for a day. The stock indexes, the S&P 500 and the Nikkei 225, are targeted. Since stock indices are composed of multiple top companies' stock prices, investing in it means investing in several companies.

3.2 Stationary Features Extraction

Since the price data is non-stationary, the mean and autocovariance are different depending on time. This means that the distributions of training data and test data are not analogous, thus degrading the generalization of the model. Hence, it is necessary to apply a stationary transformation to the prices. The basics of stationary feature extraction are achieved by difference transformation and normalization. We do as show in Table 1.

Besides, the correlation coefficient and the maximal information coefficient (MIC) [14] are used for redundancy elimination. MIC, which utilizes mutual information coefficient, represents a nonlinear relationship such as a parabola or an ellipse. In this reduction, if two features are highly correlated, the one with the lower MIC between the feature and the next return is dropped. In this way, the beneficial features are kept.

Table 1: A specific example of stationary feature extraction, where o_t, h_t, l_t, and c_t are the opening, high, low, and closing prices at time t.

Features	Calculation Formula
Log Return	$\log \frac{c_t}{c_{t-1}}$
True Range	$\frac{\max(\lvert h_t - c_{t-1}\rvert, \lvert l_t - c_{t-1}\rvert, h_t - l_t)}{c_{t-1}}$
Real Body	$\frac{\lvert o_t - c_t\rvert}{c_{t-1}}$
Shadow Range	$\frac{(h_t - l_t) - \lvert o_t - c_t\rvert}{c_{t-1}}$
Candle Value	$\frac{c_t - o_t}{h_t - l_t}$

3.3 Trading Environment

In order to simulate a trading process, the environment is formalized into a MDP as follows.

State: The state consists of the features, long and short position PnL and available funds. The agent observes the state for the latest 25 days. This value is decided by the hyperparameter tuning in Section 3.6 before the experiment.

Action: The action is a discrete set $A = \{-1, 0, 1\}$. Each value represents a short position, no position, and a long position. Note that if the current action and the next action are the same, the agent never trades. Also, if a trade is completed, it always costs 0.1% of the transaction amount.

Reward Function: The reward function is defined as the logarithmic return on assets $r_t = \log e_t - \log e_{t-1}$, where e_t is the equity. Since the logarithmic return is additive, different from the simple return, the objectives of cumulative reward maximization and asset maximization are matched.

3.4 DRL Algorithms

Four DRL algorithms, DQN, A2C, PPO, and SAC, are used as agents for algorithmic trading. These are implemented based on Ray RLlib[1], an open source library for RL. The concept of each DRL is described below.

- **DQN** is value-based method and off-policy learning. It is a known starting point for DRL and establishes a standard method for use of a deep neural network as a function approximator for RL.

- **A2C** is an actor-critic method and on-policy learning. It enables learning by multiple environments to cover inefficient search, which is a disadvantage of on-policy.

- **PPO** is an actor-critic method and on-policy learning. This is inspired by Trust Region Policy Optimization (TRPO) [15]. TRPO solves the conditional optimization to minimize the loss subject to the constraint where the KL divergence between the policy distribution before and after the update is less than a threshold. The purpose is to stabilize the learning process. PPO simplifies the implementation and improves the performance of TRPO.

- **SAC** is an actor-critic method and off-policy learning. It is characterized by the addition of a policy entropy term. This addition facilitates a more diverse search and reduces the sensitivity to hyperparameters.

Moreover, the network model connects two blocks which include fully connected layers to prevent overfitting. The blocks pass through the batch normalization layer [16] first, and then the full connected layer, activation function, and dropout layer [17]. The number of units is 256 and 128, respectively.

3.5 Time Series Cross-validation

We employ a time series cross-validation method to ensure robust results. In cross-validation, the data is divided into training, validation, and testing, and the training and validation are swapped multiple times to boost the model's generalization. In case of cross-validation against time series data, it is not an appropriate validation because it includes future information due to causality and correlation.

In order to prevent information leakage, a time series cross-validation is conducted by dividing the data as shown in Figure 1. Figure 1 shows the training start year as 2010, representing a training period of 5 years, a validation period of 2 years, and a testing period of 1 year. Besides, this partitioning is slided every year for five times. Therefore, the result of the cross-validation becomes the average of the five results for the test data.

3.6 Hyperparameter Tuning

As discussed in [18], hyperparameter setting is quite significant in DRL. The longer the computational time to obtain the experimental results, the more time-consuming the hyperparameter tuning becomes. Bayesian Optimization [19] is used to automatically find the hyperparameters which maximize the validation cumulative reward. Table 2 shows the hyperparameters to be tuned and the relevant algorithms. The hyperparameters not found in the table use the default settings of RLlib.

[1] https://github.com/ray-project/ray

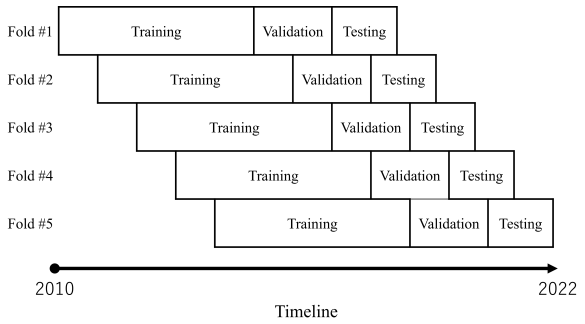


Figure 1: Time series cross-validation for a single stock index

Table 2: The target hyperparameter to be tuned

Parameter	Description	Algorithm
lr	Decide how much to update the learning.	ALL
γ	Decide how concerned about long-term rewards.	ALL
n-step	Decide how many steps to consider in prediction.	DQN, SAC
λ	Decide the trade-off between bias and variance.	A2C, PPO

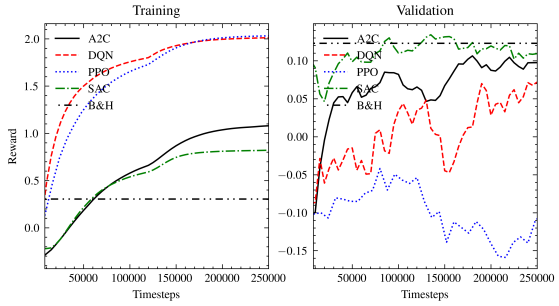
4 Results

In this section, we evaluate each DRL algorithm by simulating stock trading of the Nikkei 225 and S&P 500 in two ways: learning curve and investment performance. Note that we only focus on the experimental results of the hyperparameter with the highest validation cumulative reward within the 30 trials by Bayesian Optimization. One trial indicates running a time series cross-validation against a stock index. Our code is available on GitHub and is reproducible [2].

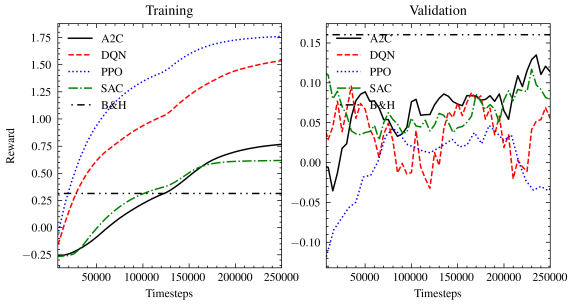
4.1 Learning Curve

Figure 2 shows the learning curve of the cumulative reward on the training and validation data. These values are the average of the five results generated by the time series cross-validation. As a baseline, we also add the buy-and-hold (B&H) strategy, which is to buy and hold stocks at the beginning of trading.

In the training data, all the algorithms are successful in learning since the cumulative reward increases steadily. On the other hand, no algorithms exceed the B&H strategy, although the trend is slowly increasing in the validation data. The PPO and DQN have higher training rewards, while their validation rewards are lower than A2C and SAC.



(a) Nikkei 225



(b) S&P 500

Figure 2: Learning curve on training and validation data.

Table 3 shows the computation time taken for one trial in which the results of time series cross-validation are obtained. These values are the time allocated for two CPU cores with no GPU. Deep learning often uses the GPU to accelerate the computation, but since DRL needs to interact with the environment running on the CPU [3], data communication is often the bottleneck. For our task, the CPU alone is faster because the batch size and network architecture are not huge. Using RLlib, which is excellent for parallel computation, these values can be reduced to the value calculated by dividing the number of CPU cores by two. Also, our execution machine is equipped with an Intel Core i7-9700F @ 3.00GHz. It takes about 3 days to learn the respective DRLs for both stock indices. A2C is the fastest to learn. SAC is the slowest and takes about six times longer than A2C.

4.2 Investment Performance

We present the investment performance on the test data in Table 3. Each metric is calculated from the transition of equity generated by simulating the best agent in the validation as below.

[2] https://github.com/napnel/UoA-thesis-s1260099

[3] There are also environments running on the GPU.

Table 3: Average computation time per trial

Algorithm	Computation Time [min]
DQN	54.1
A2C	**18.0**
PPO	49.1
SAC	108.4

- **Cumulative Return (CR)** is the percentage profit and loss from the initial equity to the final equity.
- **Maximum Drawdown (MDD)** is the percentage loss from the peak to the bottom equity.
- **Sharpe Ratio (SR)** is the return adjusted for risk.

$$SR = \frac{\mathbb{E}[e_{0:T}]}{\sqrt{Var[e_{0:T}]}} \tag{1}$$

where $e_{0:T}$ is the equity from the beginning to the end of the trading.

Table 4: Investment performance on test data, where ES=F denotes the S&P 500 in Yahoo Finance.

Algorithm	Index	CR [%]	MDD [%]	SR
DQN	N225	4.37	-9.08	**0.05**
	ES=F	1.33	**-9.19**	0.03
A2C	N225	4.65	**-8.25**	0.04
	ES=F	3.92	-10.73	0.03
PPO	N225	-11.94	-16.41	-0.05
	ES=F	3.73	-13.57	0.02
SAC	N225	2.22	-11.66	0.02
	ES=F	0.11	-13.02	-0.03
B&H	N225	**6.46**	-12.65	0.04
	ES=F	**10.28**	-12.07	**0.08**

In the DRL algorithms, DQN and A2C are profitable with low risk. PPO has the worst values in all metrics despite obtaining the best training rewards. Overall, the results using the Nikkei 225 are better than the S&P 500. Our method is not able to beat the B&H strategy quantitatively.

5 Discussion and Conclusion

In this study we evaluated the performance of algorithmic stock trading via a major DRL algorithm. To deal with two problems: DRL experimental results with large variance and over-fitting through backtesting, we incorporated stationary feature extraction, time series cross-validation, and hyperparameter tuning with Bayesian optimization.

As we observed in Section 4, our method did not outperform the B&H strategy in validation and testing, although it did successfully learn the training data. But it seems to bring a slight generalization ability since there is no sign of over-fitting behavior. Meanwhile, we expected to find a difference between the results of the on-policy and off-policy algorithms, however no such relationship was revealed.

Our method does not show excellent results, but there are a few improvements to be considered. The first is that stationary feature extraction is not always better because it takes the difference, which removes the correlation between the time series. Second, It is also worth to replace the network architecture such as convolutional neural network or long short-term memory to obtain the features across time series. The last one is to change the reward function. In Figure 2, it seems that the agent tries to learn the B&H strategy. An investor is interested in outperforming the strategy. Therefore, it may be useful to define a reward relative to the strategy.

Finally, we present the limitations of this research. In DRL, there is a problem caused by the difference between real and simulated environments. For algorithmic trading, it is not considering the different desired execution prices and market impact. In addition, our environment can only perform market order without limit order for desired execution price and stop order for closing the position when a certain condition is hit.

References

[1] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," Advances in neural information processing systems, vol. 30, pp. 3146–3154, 2017.

[2] S. Hochreiter and J. Schmidhuber, "Long short-term memory," Neural computation, vol. 9, no. 8, pp. 1735–1780, 1997.

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," arXiv preprint arXiv:1312.5602, 2013.

[4] Y. Deng, F. Bao, Y. Kong, Z. Ren, and Q. Dai, "Deep direct reinforcement learning for financial signal representation and trading," IEEE transactions on neural networks and learning systems, vol. 28, no. 3, pp. 653–664, 2016.

[5] Y. Li, W. Zheng, and Z. Zheng, "Deep robust reinforcement learning for practical algorithmic trading," IEEE Access, vol. 7, pp. 108014–108022, 2019.

[6] Z. Zhang, S. Zohren, and S. Roberts, "Deep reinforcement learning for trading," The Journal of Financial Data Science, vol. 2, no. 2, pp. 25–40, 2020.

[7] T. Théate and D. Ernst, "An application of deep reinforcement learning to algorithmic trading," Expert Systems with Applications, vol. 173, pp. 114632, 2021.

[8] M. L. De Prado, Advances in financial machine learning, John Wiley & Sons, 2018.

[9] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," International conference on machine learning, pp. 1928–1937, PMLR, 2016.

[10] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," arXiv preprint arXiv:1707.06347, 2017.

[11] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, et al., "Soft actor-critic algorithms and applications," arXiv preprint arXiv:1812.05905, 2018.

[12] I. Goodfellow, Y. Bengio, and A. Courville, Deep learning, MIT press, 2016.

[13] R. S. Sutton and A. G. Barto, Reinforcement learning: An introduction, MIT press, 2018.

[14] D. N. Reshef, Y. A. Reshef, H. K. Finucane, S. R. Grossman, G. McVean, P. J. Turnbaugh, E. S. Lander, M. Mitzenmacher, and P. C. Sabeti, "Detecting novel associations in large data sets," science, vol. 334, no. 6062, pp. 1518–1524, 2011.

[15] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," International conference on machine learning, pp. 1889–1897, PMLR, 2015.

[16] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," International conference on machine learning, pp. 448–456, PMLR, 2015.

[17] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," Journal of Machine Learning Research, vol. 15, no. 56, pp. 1929–1958, 2014.

[18] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep reinforcement learning that matters," Proceedings of the AAAI conference on artificial intelligence, vol. 32, 2018.

[19] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," Advances in neural information processing systems, vol. 25, 2012.