World Wide Web (2023) 26:3535–3559
https://doi.org/10.1007/s11280-023-01197-5

Neural-network-based parameter tuning for multi-agent simulation using deep reinforcement learning

Masanori Hirano[1] · **Kiyoshi Izumi**[1]

Received: 10 March 2023 / Revised: 27 June 2023 / Accepted: 12 July 2023 /
Published online: 3 August 2023

Abstract
This study proposes a new efficient parameter tuning method for multi-agent simulation (MAS) using deep reinforcement learning. MAS is currently a useful tool for social sciences, but is hard to realize realistic simulations due to its computational burden for parameter tuning. This study proposes efficient parameter tuning to address this issue using deep reinforcement learning methods. To improve compatibility with the tuning task, our proposed method employs actor-critic-based deep reinforcement learning, such as deep deterministic policy gradient (DDPG) and soft actor-critic (SAC). In addition to the customized version of DDPG and SAC for our task, we also propose three additional components to stabilize the learning: an action converter (DDPG only), a redundant full neural network actor, and a seed fixer. For experimental verification, we employ a parameter tuning task in an artificial financial market simulation, comparing our proposed model, its ablations, and the Bayesian estimation-based baseline. The results demonstrate that our model outperforms the baseline in terms of tuning performance, indicating that the additional components of the proposed method are essential. Moreover, the critic of our model works effectively as a surrogate model, that is, as an approximate function of the simulation, which allows the actor to tune the parameters appropriately. We have also found that the SAC-based method exhibits the best and fastest convergence, which we assume is achieved by the high exploration capability of SAC.

Keywords Multi-agent simulation · Parameter tuning · Deep reinforcement learning · Artificial financial markets

This article belongs to the Topical Collection: *Special Issue on Fairness-driven User Behavioral Modelling and Analysis for Online Recommendation*
Guest Editors: Jianxin Li, Guandong Xu, Xiang Ren, and Qing Li.

✉ Masanori Hirano
research@mhirano.jp

Kiyoshi Izumi
izumi@sys.t.u-tokyo.ac.jp

[1] School of Engineering, The University of Tokyo, 7-3-1, Hongo, Bunkyo, Tokyo 113-8656, Japan

1 Introduction

Multi-agent simulation (MAS), which primarily simulates social phenomena by compiling agent behaviors, is widely used in social sciences. For instance, MAS has been utilized in analyses of the COVID-19 pandemic [1], financial markets [2, 3], demographic movements [4], and evacuation or massive pedestrian control planning [5, 6]. Edmonds *et al.* [7] argued that MAS can be used to validate wider possibilities in social sciences. Generally, dominant equations do not exist in social phenomena, and interactions between people are unknown but important; therefore, MAS enables us to understand and analyze meta-phenomena generated by complex micro–macro interactions among agents (people).

Parameter tuning is an essential procedure in MAS. Generally, parameter tuning is used for two reasons. The first is to adjust the parameters of the simulation model such that the simulation results reflect real-world phenomena; this tuning is essential because agents' and environmental parameters cannot be directly observed in the real world. The second reason is to determine the parameter values for optimizing social phenomena or resolving social problems; for instance, because massive pedestrian simulations aim to create a more efficient flow of people, various solutions have been proposed to determine a better flow.

However, parameter tuning is a challenging task because of the large dimensions of the parameters and the computational cost of the simulations. As MAS typically employs many agents in its simulations, the computational cost for each simulation is comparatively high. Moreover, the parameters that should be tuned are also high-dimensional or large, as has been indicated by [8, 9].

A fundamental solution to address this massive computational burden has not yet been proposed, to the best of our knowledge. For instance, to reduce the computational burden, Yamashita *et al.* [9] alternated part of the MAS with neural networks (NNs) to obtain the best solution. Also, Bayesian optimization, such as the tree-structured Parzen estimator (TPE) [10], has frequently been used in the parameter tuning of NNs. However, the proposed solutions failed to fully resolve the dimensional problem because the dimensions of the parameter exploration were not compressed.

Because MAS frequently exhibits chaotic behavior owing to complex agent interactions, the exploration of optimized parameters is difficult. MAS aims to reproduce chaotic phenomena that result from complex interactions. Therefore, it is often difficult to determine optimal parameters from global estimates.

To address these issues, this study attempts to utilize deep reinforcement learning. Deep reinforcement learning has recently been developed to handle high-dimensional problems. For instance, Baker *et al.* [11] showed that it can learn complex hide-and-seek strategies using tools that make the game more complex and high-dimensional. Therefore, we believe that the utilization of deep reinforcement learning for MAS parameter tuning is promising.

Thus, this study proposes a parameter-tuning method that uses reinforcement learning for MAS. As the first step in the introduction, we focus only on low-dimensional parameter tuning. This is because low-dimensional tuning is possible even when using traditional parameter tuning, which enables us to compare and estimate the capability of deep reinforcement learning. In this study, we demonstrate the capability of deep reinforcement learning for MAS parameter tuning.

Consequently, we show that the proposed model is promising. The contributions of this study are as follows:

1. A reinforcement-learning-based parameter tuning model for MAS parameter tuning using some additional components to stabilize learning is proposed.

2. Our model outperforms the baseline model. Furthermore, the proposed additional components are successful and necessary for our proposed model.
3. In our proposed model, actor-critic type reinforcement learning is employed. The results demonstrate that the critic works as a surrogate model of the simulations, leading to the actor being able to learn a better parameter.
4. We propose SAC- and DDPG-based models and confirm that the SAC-based method exhibits the best and fastest convergence owing to its high exploration capability.

2 Related work

Edmonds *et al.* [7] argued that MAS is useful in social sciences, which have complex interactions. MAS aims to imitate the real world by creating an imaginary world using agents on computers. Simulations are beneficial because they enable the exploration of hypothetical situations or the prediction of phenomena under certain conditions, such as new regulations. As mentioned earlier, MAS has been utilized for the analysis of several domains, such as COVID-19 [1], financial markets [2, 3], demographic movement [4], and evacuation or massive pedestrian control planning [5, 6]. The importance of agent-based simulations has been debated, particularly in the context of financial markets [12, 13]. For instance, Lux *et al.* [14] showed that interactions between agents in financial market simulations are necessary to replicate stylized facts therein. Cui *et al.* [15] also showed that the trader model used in artificial financial market simulations required intelligence to replicate stylized facts in financial markets. Furthermore, Mizuta [16] demonstrated that the MAS of a financial market can contribute to the implementation of rules and regulations in actual financial markets. This study used artificial financial markets as an example of the application of the proposed method.

Several practical studies have used artificial financial market simulations. Torii *et al.* [17] revealed how the flow of a price shock was transferred to other stocks using an artificial financial market based on the trader model proposed in [18]. The model proposed in this study is also used as the parameter-tuning target. Mizuta *et al.* [2] tested the effect of tick size on trading shares, that is, the price unit for orders, which led to a discussion on tick size devaluation in the Tokyo Stock Exchange Market. Hirano *et al.* [3] assessed the effect of the regulation of the capital adequacy ratio (CAR), such as the Basel regulatory framework, and observed the risk of market price shocks and depressions due to CAR regulation. Some studies also focused on flush crashes using artificial financial market simulations [19, 20]; as an example of one such platform, Torii *et al.* [21] proposed "Plham" [22]. In this study, we use the "PlhamJ" platform [23], the updated version of "Plham." Meanwhile, other artificial financial market simulators also exist, such as U-MART [24], the Santa Fe artificial stock market [25], and the agent-based interactive discrete event simulation [26].

However, as mentioned in the introduction, MAS, including financial market simulations, has a high computational burden; thus, several workarounds have been proposed. One solution is parallelization of simulations using simulation management software. Murase *et al.* [27] proposed organizing assistants for comprehensive and interactive simulations (OACIS). Murase *et al.* [28] subsequently proposed CARAVAN, a framework for comprehensive simulations of massive parallel machines, to optimize MAS parameters by parameter sampling. These frameworks are mainly aimed at automating the parameter-tuning process and do not address the high computational burden. Another method to reduce the computational burden is introducing NNs. Yamashita *et al.* [9] alternated part of the MAS with NNs to obtain the best solution and reduce the computational burden. The NN operates as an approximate

function of MAS; this approach is known as a surrogate model. The potential of the surrogate MAS model was investigated by Angione *et al.* [29]. In this study, part of our model can be understood as a surrogate model of MAS.

This study employed reinforcement learning to reduce the computational burden of MAS parameter tuning. In the context of reinforcement learning, numerous research and development initiatives have been undertaken. Q-learning is a well-known off-policy reinforcement learning method based on the Q-table [30]. Learning theory, that is, the temporal difference, was proposed in [31]. Tesauro proposed a method for applying temporal-difference learning to backgammon [32]. SARSA, state–action–reward–state–action, is another example of a simple reinforcement learning method proposed in [33]. There are numerous models of reinforcement learning. The most significant discovery in reinforcement learning is its combination with NNs. After deep reinforcement learning was invented, Mnih *et al.* [34] demonstrated that deep Q-learning (Deep Q-Network, DQN) can outperform humans using the Atari learning environment [35]. These neural-network-based models are known as deep reinforcement learning and were also utilized in this study. These improvements are possible owing to the invention of NNs and deep learning such as convolutional NNs for image classification [36]. Consequently, several reinforcement learning methods using deep learning have been developed. As an extension of DQN, double DQN (DDQN) was proposed in [37], which uses two networks and has better performance than DQN. These two networks were also used in the proposed method. Moreover, the dueling DDQN was proposed using a new state value function to improve learning performance [38]. As an extension of Q-learning, the asynchronous advantage actor-critic [39] method uses deep learning asynchronously. This is based on the actor-critic method proposed in [40], which is used in the deep reinforcement learning of our proposed model. Subsequently, because the asynchronous method was not important, the advantage actor-critic (A2C) method was proposed [41]. Hessel proposed RainBow [42] as a method that combined the aforementioned methods. Ape-X DQN [43], which used prioritized experience replay, and R2D2 [44], which used long short-term memory [45], were proposed as outperforming methods. Silver *et al.* proposed a reinforcement learning algorithm through self-playing, which achieved excellent performance in some games, such as chess, shogi, and Go [46]; this algorithm is based on their previous study, which is well known as "Alpha Go Zero" [47]. In this study, we used the deep deterministic policy gradient (DDPG) [48] and the soft actor-critic (SAC) [49, 50], an actor-critic reinforcement learning for continuous actions.

3 Proposed methods

In the proposed method, we employed actor-critic-based deep reinforcement learning methods with continuous action spaces, such as DDPG [48] and SAC [49, 50]. We selected these for the following reasons. First, Figure 1 shows the typical parameter tuning and our proposed schemes. At the tuning trial in the usual parameter tuning scheme, the parameter tuner provides a parameter set for the parameter-tuning task. Subsequently, the simulator runs using the parameter set and returns the results, and the analyzer (objective function) returns the final objective value. Finally, the parameter tuner receives feedback and attempts to modify the parameter set to maximize the objective value. To consider this scheme down to the reinforcement learning scheme, the actor-critic framework exhibits a good fit, as illustrated in Figure 1. Second, the typical parameter-tuning tasks and parameter sets frequently include continuous values; therefore, support of continuous values is required. The illustration in

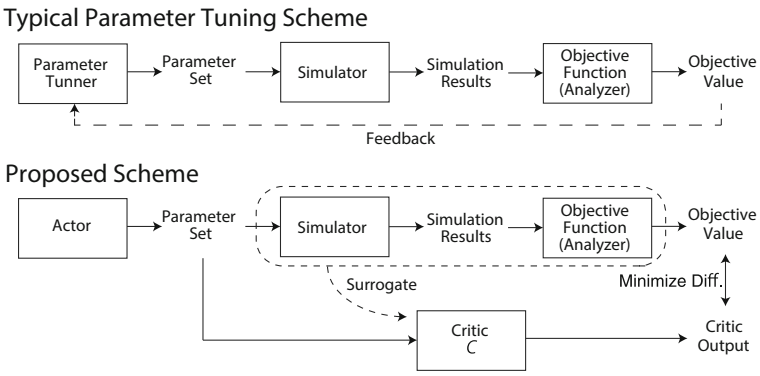

Figure 1 Parameter tuning scheme

Figure 1 represents the basic outline of our methods; however, the details are not exactly the same as illustrated and explained hereafter.

The use of either DDPG or SAC with minor customization to adapt DDPG to our task failed. Thus, we proposed three special components to enable their application to MAS parameter tuning. The three components are as follows:

1. Action converter (AC)
2. Redundant full neural network actor (FNNA)
3. Seed fixer (SF)

An AC is employed to enforce action bounds; in SAC, action bounds are realized by the squashing function for Gaussian sampling. Therefore, the AC is only used for DDPG. The patterns of applicable components are listed in Table 1 and all the models are listed in Table 2.

We first explain the base models except for the baseline, that is, DDPG and SAC with small customization, for MAS parameter tuning and thereafter describe the three components.

3.1 Customized DDPG

A customized DDPG functions similar to the original DDPG; however, the input/output is different because of the task settings.

First, the actor in our model (A in Figure 2) does not accept any states. Typically, the actor accepts the current state to calculate the best policy based on the current state. However, the parameter-tuning task does not include any representation of the current state; moreover, the output of our actor is a parameter set, because the parameter set is similar to an action. Thus, our actor does not accept any state, and it only generates a policy for the parameter

Table 1 Models and additional components

Base Model	Additional Components		
	AC	FNNA	SF
Customized DDPG	Applicable	Applicable	Applicable
Customized SAC	N/A	Applicable	Applicable

Table 2 All models (Some models will not work properly in the following experiments)

Model Name	Base Model	Additional Components		
		AC	FNNA	SF
Baseline	TPE: Optuna	-	-	-
DDPG + AC + FNNA + SF	Customized DDPG	✓	✓	✓
DDPG + FNNA + SF	Customized DDPG	-	✓	✓
DDPG + AC + SF	Customized DDPG	✓	-	✓
DDPG + AC + FNNA	Customized DDPG	✓	✓	-
DDPG + AC	Customized DDPG	✓	-	-
DDPG + FNNA	Customized DDPG	-	✓	-
DDPG + SF	Customized DDPG	-	-	✓
SAC + FNNA + SF	Customized SAC	-	✓	✓
SAC + SF	Customized SAC	-	-	✓
SAC + FNNA	Customized SAC	-	✓	-
SAC	Customized SAC	-	-	✓

sets as follows: $P_i = A()$, where P_i denotes the ith trial parameter set for the simulator. If the parameter is N-dimensional, $P_i = \left(p_{1,i}, p_{2,i}, \cdots, p_{N,i}\right)$, and A denotes the actor.

Second, the critic also differs from the original DDPG. A typical critic of DDPG accepts the state and action, but our critic only accepts a parameter set (action) similar to our actor. Moreover, although the usual prediction target of the critic is the Q-value (the sum of the discounted future returns), the prediction target of the proposed critic is only an objective value of parameter tuning, as shown in Figure 2; this is because parameter tuning is a one-shot trial and not a Markov process. In typical reinforcement learning tasks, every action is supposed to have semi-permanent effects, and the value of each action must be evaluated based on the future expected returns. However, the selection of one parameter does not affect the future results of parameter-tuning tasks; thus, consideration of the discounted future expected returns is not required. The loss function for our critic is defined as follows:

$$L_C = MSE[o_i, C(P_i)] = (o_i - C(P_i))^2, \tag{1}$$

where o_i denotes the objective value of the ith trial from the simulation results, C denotes our critic, P_i denotes the ith trial parameter set given by the actor, and $C(P_i)$ denotes the critic output for the given parameter set P_i.

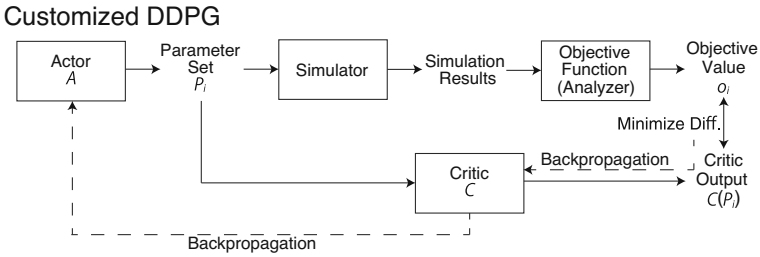

Figure 2 Model outline of customized DDPG

Therefore, actor loss is defined as,

$$L_A = -C(A()) \tag{2}$$

because our actor learns to maximize the objective value estimated by our critic. Thus, by minimizing L_C and L_A alternately, our critic pursues the correct evaluation for a given parameter set and our actor pursues a better parameter set to obtain higher objective values. These procedures are similar to those used for the original DDPG.

Similar to the original DDPG, we employed the Ornstein–Uhlenbeck process [51] as an exploration noise, replay buffer [52], and soft-target update.

Moreover, in the case of a simulation failure, a penalty for the objective value was applied because the objective value could not be calculated. However, if the parameter is set inappropriately, the gradient cannot be calculated because all simulations will fail, and only penalties are applied, which have no gradient. Thus, if the parameter is set to this value through learning, the new parameter is selected again at random.

3.2 Customized SAC

Based on the original SAC [49, 50], we developed a customized SAC. Similar to the customized DDPG, the most important customizations are the input and output. However, unlike DDPG, SAC has a slightly more complex architecture than DDPG.

SAC is similar to DDPG in terms of supporting continuous action spaces. However, unlike DDPG, SAC employs a stochastic policy and an entropy term of the policy in its objective function, which enables high exploration capability. Figure 3 shows the outline of the implementation of the customized SAC.

First, in our actor, the action is generated using the reparameterization trick as follows:

$$\mu, \sigma = A(), \tag{3}$$

$$P_i \sim \mathcal{N}(\mu, \sigma), \tag{4}$$

where $A()$ denotes the SAC actor NN that generates $\mathbb{R}^{2\times N}$. $\mathcal{N}(\mu, \sigma)$ is a Gaussian distribution, whose mean and variance are $\mu \in \mathbb{R}^N$ and $\sigma^2 \in \mathbb{R}^N$, respectively . $P_i = \left(p_{1,i}, p_{2,i}, \cdots, p_{N,i}\right)$ are the N-dimensional simulation parameters for the i th iteration.

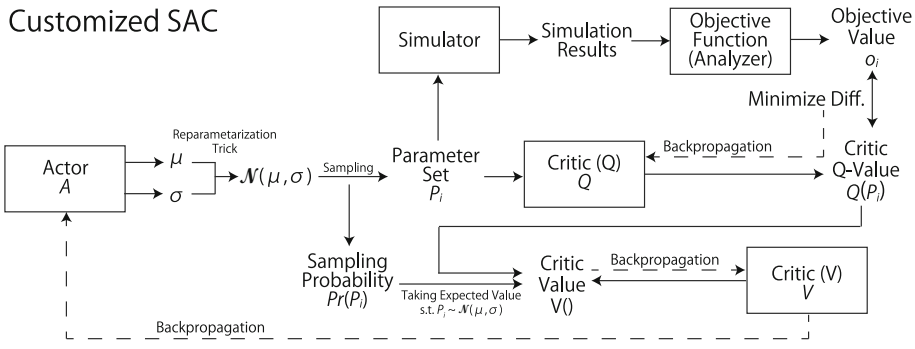

Figure 3 Model outline of customized SAC

In our SAC critic, unlike the customized DDPG, the model comprises two networks: $Q(P_i)$ and $V()$. $V()$ does not accept any inputs and the target value is theoretically defined as follows:

$$V() = \mathbb{E}_{P_i \sim \mathcal{N}(\mu,\sigma)} \left[Q(P_i) - \alpha \ln \{\Pr(P_i)\} \right], \tag{5}$$

where $\Pr(P_i)$ denotes the probability that P_i is sampled from $\mathcal{N}(\mu, \sigma)$ in (4), and α is a hyperparameter which has been set to 0.2 in our experiments. In the usual reinforcement learning method, $Q(P_i)$ and $V()$ denote the action value and state value function, respectively. However, in our study, no such state existed. Thus, if we interpret them, $Q(P_i)$ and $V()$ are considered the parameter value and exploration value function, respectively, because the second term in $V()$ denotes policy entropy. Moreover, although P_i appears on the right side of (5), $V()$ does not depend on P_i because it evaluates the current state in the original theory of reinforcement learning, and no state is available in this task.

The loss functions of the critic are defined as follows:

$$L_Q = MSE[o_i, Q(P_i)] = (o_i - Q(P_i))^2, \tag{6}$$

$$L_V = MSE[Q(P_i) - \alpha \ln \{\Pr(P_i)\}, V()] \tag{7}$$

$$= [Q(P_i) - \alpha \ln \{\Pr(P_i)\} - V()]^2 . \tag{8}$$

In our experiments, we employed target networks. Therefore, $Q(P_i)$ in (7) and (8) is the fixed network and no gradient is passed.

However, the loss function of the actor is obtained as follows:

$$L_A = -V() = -\mathbb{E}_{P_i \sim \mathcal{N}(A())} \left[Q(P_i) - \alpha \ln \{\Pr(P_i)\} \right]. \tag{9}$$

Here, backpropagation is realized using the aforementioned reparameterization trick for some samples.

In the original study on SAC [49, 50], squashing of the action space (enforcing action bounds in the original context) was also employed. In our experiments, the parameters had positive value restrictions. Therefore, we formulated a squashed Gaussian policy. In our model, we employed the function $f(x) = \ln(1 + \exp(x))$. Thus, the Gaussian policy is converted to,

$$\ln \{\Pr(P_i)\} = \ln \{\Pr_{\mathcal{N}}(\mathbf{u})\} + \sum_j P_{i,j} - \sum_j u_j, \tag{10}$$

where $P_i = \ln(1 + \exp(\mathbf{u}))$, $\mathbf{u} \sim \mathcal{N}(\mu, \sigma)$ and $\Pr_{\mathcal{N}}(\mathbf{u})$ denotes the sampling probability. This definition differs from that in (4) because of the use of action squashing.

In addition, in a customized SAC, the replay buffer [52], soft target update, and penalty for invalid simulations are employed.

3.3 Action converter (AC)

We propose an AC to introduce parameter restrictions into a customized DDPG. The AC converts the output of the actor into a restricted parameter space, like action squashing in the SAC algorithm. In our study, we mainly used this for a non-negative constraint parameter using the function $f(x) = \ln(1 + \exp(x))$ as in SAC.

This is similar to the aforementioned enforcing action bounds in SAC [49, 50]; however, only the aim (enforcing action bounds) is common to our AC; notably, SAC is not a deterministic reinforcement learning. Although SAC squashes its action probability, DDPG does

not have an action probability but only employs exploring noise. Thus, DDPG requires action space squashing instead of probability squashing to bind the action space.

Thus, we assumed that a more relaxed squashing than that of SAC should be applied as an AC. Excessive changes should be avoided, particularly in areas that do not require constraints, to avoid disturbing DDPG exploration. Thus, for the non-negative constraint, we employed $f(x) = \ln(1 + \exp(x))$. This is because $\frac{\partial f(x)}{\partial x} = \frac{e^x}{1+e^x} = 1 - \frac{1}{1+e^x}$, which is almost one when x is sufficiently large.

However, it is also possible to consider other functions. In addition, we did not discuss and examine the AC for other parameter constraints.

A practical application of AC is to the constraint parameters of the actor output. For instance, suppose that the parameter space is two-dimensional, and all parameters have non-negative constraints. In this case, $P_i = \left(p_{1,i}, p_{2,i}\right)$, and we define $p^{\star}_{1,i} = \ln(1 + \exp(p_{1,i}))$ and $p^{\star}_{2,i} = \ln(1 + \exp(p_{2,i}))$. Subsequently, the final parameter set $P^{\star}_i = \left(p^{\star}_{1,i}, p^{\star}_{2,i}\right)$ is obtained and used instead of P_i.

Note that AC is applied only to the customized DDPG; it is not necessary for SAC because it has enforcing action bounds.

3.4 Redundant full neural network actor (FNNA)

For a simple implementation of our models, that is, the customized DDPG and SAC, the actor is only required to return the gradient-enabled parameters like the left panel of Figure 4. Thus, the minimum implementation is for the actor to have only N parameters when the dimension of the parameter set is N because the actors in our models do not accept an input.

However, we propose an actor with a redundant NN called a redundant full NN actor (FNNA). As the right panel of Figure 4 shows, the FNNA architecture contains a multi-layer perceptron (MLP), and the actor accepts a dummy vector all of whose components are one.

Although this architecture seems redundant and meaningless, it is similar to the lottery ticket hypothesis [53]. We assume that FNNA worked better than the minimum implementation of our models because NN performs similar to collective intelligence or learning and are stabilized according to the lottery ticket hypothesis.

Moreover, when we employed SAC and FNNA, the FNN was also applied to the V-net ($V()$).

3.5 Seed fixer (SF)

MAS frequently exhibits various behaviors depending on random seeds. Random variables are used in many MAS routines to realize the probabilistic decisions of agents. However, because of the complex interactions between agents, slight state differences cause chaotically significant differences. Thus, differences in random variables could cause significant differences, as illustrated in the left panel of Figure 5.

The effects of random seeds are significant in learning. A critic was introduced to learn the relationship between the simulation parameter sets and simulation results. The actor was trained using the gradients obtained from the critic. The smoothness of the critic gradient, that is, the smooth response of the critic outputs to the input parameter, is necessary. Thus, the difference in the objective values caused by small changes in the parameter set must be greater than the effect of the random seeds.

Figure 4 FNNA blue print

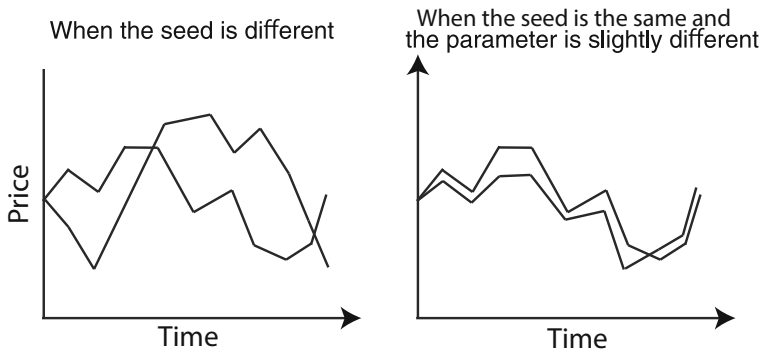

Figure 5 Blueprint of the seed fixer. Plots represent each path of financial market simulation. The left panel illustrates the images when the simulation seed is different. The right panel illustrates the images when the simulation seed is the same, and the simulation parameter is slightly different

Although a larger number of simulations generally eases the effect of random seeds, it is insufficient to erase their effects for critic gradients. Moreover, it is difficult to increase the number of simulations when the computational cost is large.

The SF provides one solution and fixes the seeds used in each set of simulations, as illustrated in the right panel of Figure 5. The objective values obtained through the simulations illustrated in Figure 1 are calculated by the mean of M trials for stabilization. Thus, the SF fixes M random seeds in the simulations; this implies that the set of random seeds for M simulations is always the same.

Introducing an SF eliminates the effect of random seeds and smoothens the gradient of the critic. Although simulations can exhibit chaotic behavior, they are improved.

However, this component has a trade-off, in that the learning results are biased. As the random seeds are fixed to only M patterns, and M is generally excessively small to cover all possible states, the results are biased by these M patterns. Thus, there is a tradeoff between the gradient smoothness of the critic and the possibility of bias. Although employing a larger M could be a solution, it could also increase the computational burden. Fortunately, in our experiments, $M = 100$ did not appear to cause a significant bias.

4 Experiments

4.1 Task setting

We employed an artificial financial market simulation as a MAS for the experiments. Recently, particularly after the 2007–2008 financial crisis, financial market simulations have gained focus. For instance, Bookstaber [54] argued that MAS for financial markets was important for avoiding future crashes caused by the complex interaction of financial market factors. Mizuta [16] argued that MAS could contribute to the discussion of regulations in financial markets.

An artificial financial market simulation seems an ideal task for testing our method. It is because statistical indicators are used in actual financial markets and can be used for tuning targets. It enables us to set a reasonable tuning target.

Our experiments employ the stylized financial market simulation proposed in [17], which is widely used as an artificial financial market simulation. This simulation is available via PlhamJ, a platform for large-scale, high-frequency artificial financial markets (Java version) [23].

Moreover, our proposed method can be applied to other simulations as long as Key Performance Indicators (KPIs) for the output are defined. Therefore, although this study only tests our method on an artificial financial market simulation, it is also applicable to other MAS tasks and simulation models.

4.1.1 Simulation

Only 100 stylized trader agents based on [17] and one continuous double-auction market exist in the simulation. At time t, the stylized trader agent i determines its trading actions using the following criteria: fundamentals, chartists (trends), and noise. Initially, the agents calculate the following three factors:

- Fundamental factor:
$$F_t^i = \frac{1}{\tau^{*i}} \ln \left\{ \frac{p_t^*}{p_t} \right\}. \tag{11}$$
where τ^{*i} denotes the mean reversion-time constant of agent i, p_t^* denotes the fundamental price at time t, and p_t denotes the price at time t.
- Chartist factor:
$$C_t^i = \frac{1}{\tau^i} \sum_{j=1}^{\tau^i} r_{(t-j)} = \frac{1}{\tau^i} \sum_{j=1}^{\tau^i} \ln \frac{p_{(t-j)}}{p_{(t-j-1)}}, \tag{12}$$
where τ^i denotes the time window size of agent i and r_t denotes the logarithmic return at time t.
- Noise factor:
$$N_t^i \sim \mathcal{N}(0, \sigma). \tag{13}$$
denotes that N_t^i obeys a normal distribution with zero mean and variance σ^2.

Subsequently, the agents calculate the weighted averages of these three factors as follows:

$$\widehat{r_t^i} = \frac{1}{w_F^i + w_C^i + w_N^i} \left(w_F^i F_t^i + w_C^i C_t^i + w_N^i N_t^i \right), \tag{14}$$

where w_F^i, w_C^i, and w_N^i denote the weights of agent i for each factor.

Next, the expected price of agent i is calculated using the following equation:

$$\widehat{p_t^i} = p_t \exp\left(\widehat{r_t^i} \tau^i \right). \tag{15}$$

Subsequently, using a fixed margin of $k^i \in [0, 0.1]$, we determine the actual order prices using the following rules:

- If $\widehat{p_t^i} > p_t$, agent i places a bid (buy order) at the following price:
$$\min \left\{ \widehat{p_t^i}(1 - k^i), p_t^{\text{ask}} \right\}. \tag{16}$$
- If $\widehat{p_t^i} < p_t$, agent i places an ask (sell order) at the following price:
$$\max \left\{ \widehat{p_t^i}(1 + k^i), p_t^{\text{bid}} \right\}. \tag{17}$$

Here, p_t^{bid} and p_t^{ask} denote the best bid and ask prices, respectively.

The parameters employed for this type of trader are as follows: $p_t^* = 300$, $w_N^i \sim Ex(1.0)$, $\sigma = 0.001$, $\tau^* \in [50, 100]$, and $\tau \in [100, 200]$, which were mainly determined based on [17]. Here, $Ex(\lambda)$ indicates an exponential distribution with an expected value of λ. The values of w_F, w_C in $w_F^i \sim Ex(w_F)$ and $w_C^i \sim Ex(w_C)$ were the tuning targets in this experiment.

4.1.2 Objective value

The objective values are the skewness and kurtosis of the log-returns of the market prices. The values are calculated as follows:

$$x_t := \ln\left\{\frac{p_t}{p_{t-1}}\right\}, \tag{18}$$

$$\text{skewness} = \frac{1}{T}\sum_{t=1}^{T}\frac{(x_t - \overline{x})^3}{(x_t - \overline{x})^{2\frac{3}{2}}}, \tag{19}$$

$$\text{kurtosis} = \frac{1}{T}\sum_{t=1}^{T}\frac{(x_t - \overline{x})^4}{(x_t - \overline{x})^{2^2}}, \tag{20}$$

where T denotes the total number of simulation steps, which was 10,000 in this study.

According to the results in [55], we set the targets of skewness and kurtosis to 0.0 and 6.0, respectively; these values are approximates as these values differ across financial markets. However, this was not important because we focused only on the tuning efficiency of the proposed method in this study.

Subsequently, the total mean square error (MSE) of both the skewness and kurtosis of the targets was calculated. Thus,

$$\text{MSE}[(\text{skewness}, \text{kurtosis}), (0.0, 6.0)] = (\text{skewness} - 0.0)^2 + (\text{kurtosis} - 6.0)^2. \tag{21}$$

If the simulations fail, the MSE is replaced by the penalty, which was set to $1,000$ for this study. Simulation failures could occur because the price increased to infinity owing to inappropriate parameters.

Then, the final objective values were calculated as the negative MSE because the objective values are maximized.

In the tuning phase, all simulations were performed 100 times and the mean of the objective values was used as the final objective value to stabilize these objective values. Moreover, in the final evaluation phase after tuning, we performed 1,000 simulations to evaluate the performance of the tuners.

4.2 Models

As mentioned in the task settings, our parameter-tuning task aims to obtain a better w_F and w_C to provide a better objective value. As per the baseline model representing the typical parameter-tuning scheme shown in Figure 1, we employed a TPE (Optuna). In the following section, we explain this as well as the detailed settings of our proposed models.

4.2.1 Baseline model: tree-structured parzen estimator (Optuna)

We employed a TPE [10] as a baseline model for MAS parameter tuning. This estimator is frequently used for parameter tuning in machine learning and is known as Optuna [56]. TPE is a Bayesian optimization method that pursues the best parameter set for a higher objective value as a black box optimization problem.

4.2.2 Our customized DDPG

All schemes and learning theories are explained in Section 3. Here, we describe the settings in detail, including the model parameters in the customized DDPG used in our experiments.

- Actor (FNNA): four-layered MLP with 100-dimension hidden layers returning two-dimension output. Each layer, except for the last layer, uses layer normalization and ReLU activations. This implies that a 100-dimensional dummy vector filled with ones is accepted as the input.
- Actor (not FNNA): always returns only two gradient-enabled parameters.
- Critic: four-layered MLP with 100-dimension hidden layers accepting two-dimensional input (parameter set) and retuning one-dimensional output (estimated objective value). Each layer, except for the last layer, uses layer normalization and ReLU activations.
- Soft target update ratio of DDPG: 0.1
- Actor learning rate: 10^{-4}
- Critic learning rate: 10^{-3}
- Batch size: 100
- Buffer size of experience replay: 1,000

4.2.3 Our customized SAC

All schemes and learning theories are explained in Section 3. Here, we explain the settings in detail, including the model parameters in the customized SAC used in our experiments.

- Actor (FNNA): three-layered common MLP with 100-dimension hidden layers and the final linear layers returning two-dimension outputs for μ and σ, respectively. All linear layers use layer normalization and each layer in the three-layered common MLP uses ReLU activations. This implies that a 100-dimensional dummy vector filled with ones is accepted as the input.
- Actor (not FNNA): always returns two two-dimensional gradient-enabled parameters for μ and σ, respectively.
- Q-net ($V(P_i)$): four-layered MLP with 100-dimension hidden layers accepting two-dimensional input (parameter set) and retuning one-dimensional output (estimated objective value). Each layer, except for the last layer, uses layer normalization and ReLU activations.
- V-net ($V()$ with FNNA): four-layered MLP with 100-dimension hidden layers accepting a 100-dimensional dummy vector filled with ones and retuning one-dimensional output ($V()$). Each layer, except for the last layer, uses layer normalization and ReLU activations.
- V-net ($V()$ without FNNA): always returns one-dimensional gradient-enabled parameters for $V()$.
- Soft target update ratio of SAC: 0.1
- Actor learning rate: 10^{-4}

- Critic learning rate: 10^{-3}
- Batch size: 100
- Buffer size of experience replay: 1,000

4.3 Evaluation

For fair evaluation, the number of simulations available for training was limited to 100,000 at all. This implies that each epoch of training performed 100 simulations; thus, 1,000 epochs were performed.

After the training phase, the final evaluation test was performed with 1,000 simulations using the best-performing model during training (based on objective values for training simulations).

5 Results

Table 3 summarizes the results of each of the ten experiments. This table lists the loss (negative objective value), kurtosis, and skewness of the tuning results of all models, as well as the mean (and standard deviation) and median values.

The loss is defined by (21), and a smaller loss indicates better parameter tuning. Moreover, as mentioned earlier, the loss is the negative value of the objective values because the actor-critic framework generally maximizes the objective value. In our settings, MSE was employed for this loss; thus, the loss was always non-negative.

According to the results in Table 3, the SAC + FNNA + SF setup exhibits the best performance in terms of loss. However, SAC + FNNA + SF shows not only the best mean of the loss but also the best deviation and median of the loss compared with those of the baseline and other models.

The SAC-based model achieved the best results, but the DDPG-based model outperformed the baseline. However, SAC-based models appear to be more stable than DDPG-based models.

Comparing all the results of the DDPG-based models with those of other DDPG-based models missing the specified components, it is thus confirmed that all three components (AC, FNNA, and SF) are essential for our proposed DDPG-based model, even if one of the components is missing, performance degrades significantly. In particular, AC is necessary to tune because learning does not work completely if the AC is missing. When the AC is missing, the non-negative constraints of the parameters in our task are not always satisfied, thus causing an invalid simulation. An invalid simulation always returns a fixed penalty, which leads to the missing parameter-tuning gradient. Thus, at least in our task, the AC is a necessary feature for DDPG-based models to run the tuner practically.

Comparing the effects of each component of the DDPG-based models, AC > FNNA > SF was observed in the sequence of larger effects. The effect of the SF seems comparatively small, but if it is missing, our DDPG-based models do not exhibit better performance than the baseline.

In contrast, for SAC-based models, only the SF is a necessary component. FNNA performs better only when the SAC-based model employs the SF. In contrast to DDPG-based models, the AC is not applicable. However, the dynamics of the effects of the additional components are also different from those of DDPG-based models.

Table 3 Evaluation Results (statistic among 10 experiments). SAC + FNNA + SF shows the best results and is statistically different (significance level of 5%) from others except for SAC + SF

Model	Loss ($-o_i$)		Kurtosis		Skewness	
	Mean	Median	Mean	Median	Mean	Median
Baseline (TPE: Optuna)	3.410830 ± 5.993605	0.558568	6.629670 ± 0.437744	6.407023	0.006460 ± 0.009737	0.006179
DDPG + AC + FNNA + SF	0.727502 ± 0.560671	0.489755	6.288674 ± 0.092182	6.158437	0.007488 ± 0.004775	0.004005
DDPG + FNNA + SF	Cannot be tuned at all					
DDPG + AC + SF	17911.68 ± 21985.85	4942.005	26.35092 ± 23.80422	7.160722	0.014936 ± 0.037776	0.000424
DDPG + AC + FNNA	1094.991 ± 3252.000	0.652749	8.631269 ± 4.523929	6.892526	0.005324 ± 0.020119	0.010744
DDPG + AC	36871.57 ± 47942.91	20397.35	44.59121 ± 45.65828	7.138884	-0.035963 ± 0.081353	-0.001542
SAC + FNNA + SF	0.194473 ± 0.045963	0.195069	6.245487 ± 0.087551	6.222321	0.006581 ± 0.004671	0.004943
SAC + SF	0.203258 ± 0.045476	0.209067	6.275684 ± 0.050873	6.243806	0.007037 ± 0.004622	0.006978
SAC + FNNA	21.41606 ± 14.70367	25.60599	9.855362 ± 1.985068	9.583112	0.009523 ± 0.008616	0.007500
SAC	17.45062 ± 14.48998	21.85446	9.363488 ± 2.028399	9.176651	0.007901 ± 0.006655	0.008429

Figures 6, 7, 8, 9, 10, 11, 12, 13 and 14 show the losses of all models for each epoch. The evaluation losses are plotted in these figures. The loss values plotted in these figures were calculated using an additional 1,000 simulations only for evaluations. Figure 6 shows the results for the baseline (TPE: Optuna). In this figure, a type of overfitting of the training samples can be observed. In contrast, in Figure 7, the DDPG-based model (DDPG + AC + FNNA + SF) exhibits a slower convergence. Unlike these two models, according to Figure 11, the SAC-based model (SAC + FNNA + SF) exhibits faster and more stable convergence in its loss. In addition, from these figures, we can assume that the SAC-based model outperforms other models. Moreover, as shown in Figures 11 – 14, we can observe that the SF leads to faster convergence.

6 Discussion

Our proposed model exhibited better tuning performance than the baseline model. In our evaluation, the SAC-based models exhibited the best performance, particularly with the SF. As mentioned earlier, our task setting was less dimensional; therefore, the baseline method has several advantages. Although our proposed method exhibited its advantages in higher-dimensional parameter tuning under our assumption, the results demonstrated that our proposed model works well and seems promising.

Moreover, our proposed components of DDPG and one component (SF) of SAC were found to be essential. If one of these three components for DDPG were missing, the DDPG-based models would have performed worse than the baseline. Moreover, if the SF was missing for SAC, the SAC-based models would also have performed worse than the baseline. Faster convergence caused by the SF was also clearly observed as shown in Figures 11 and 13, and Figures 12 and 14. According to these results, the SF is the most important factor for parameter tuning in MAS. In our proposition, we assumed that differences in the random variables could cause significant differences because of the chaotic behavior of MAS as a complex system. Our results demonstrated that this assumption was valid.

According to the results, SAC-based models outperformed DDPG-based models. The first possible reason is that SAC is not deterministic. Because of the deterministic policy of DDPG, DDPG-based models only achieved parameter tuning by continuous exploration. The action

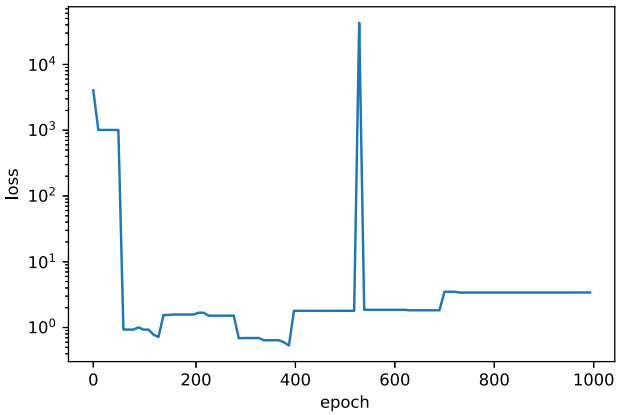

Figure 6 Evaluation loss of the baseline model (TPE: Optuna)

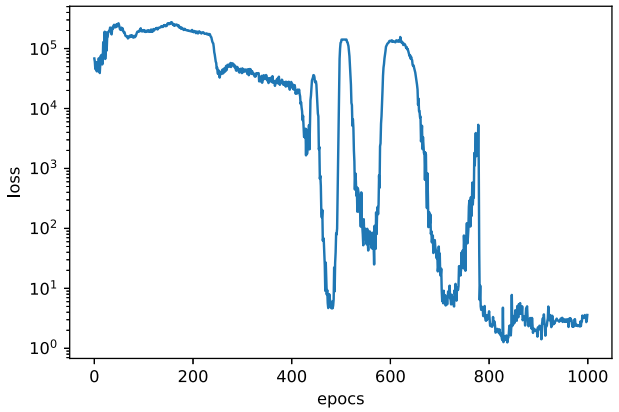

Figure 7 Evaluation loss of DDPG + AC + FNNA + SF

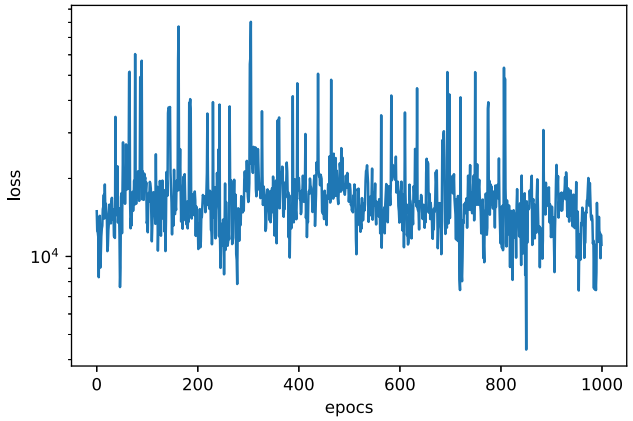

Figure 8 Evaluation loss of DDPG + AC + SF

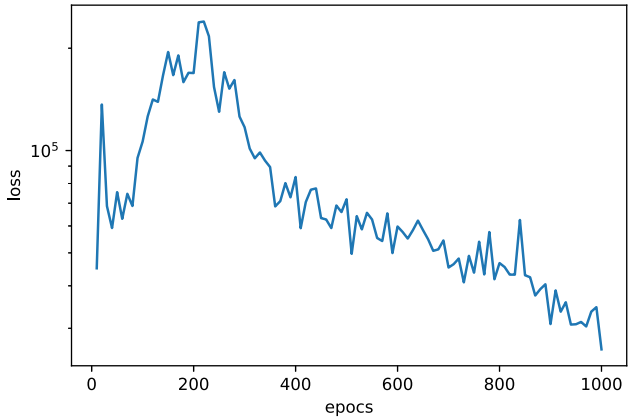

Figure 9 Evaluation loss of DDPG + AC + FNNA

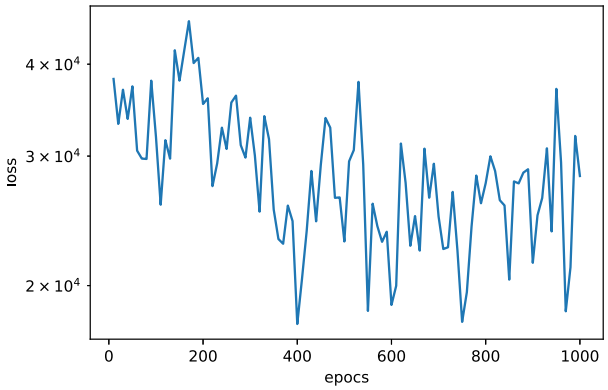

Figure 10 Evaluation loss of DDPG + AC

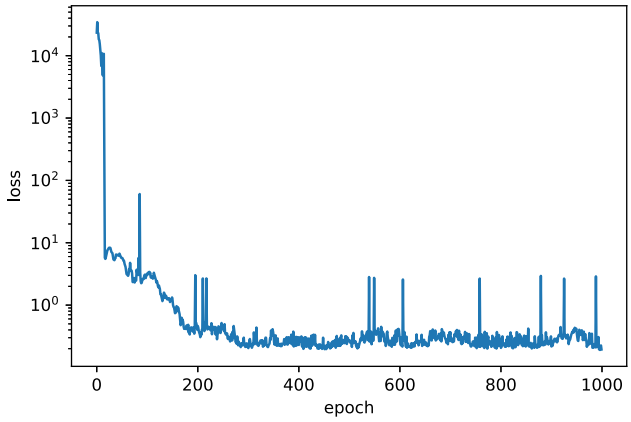

Figure 11 Evaluation loss of SAC + FNNA + SF

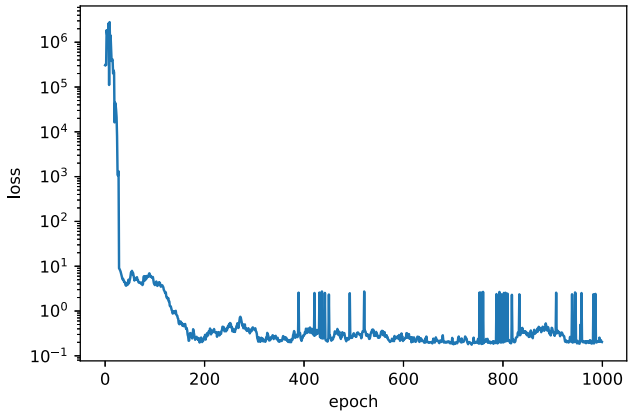

Figure 12 Evaluation loss of SAC + SF

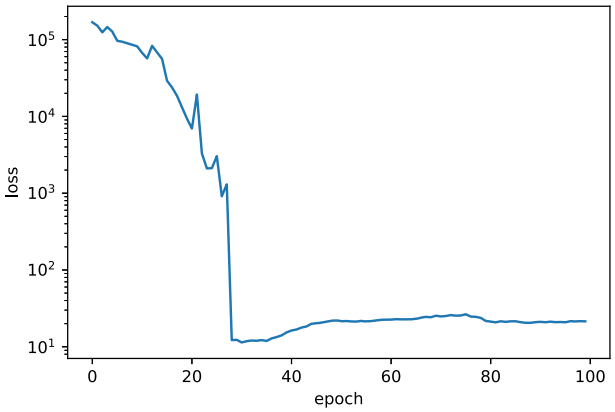

Figure 13 Evaluation loss of SAC + FNNA

space, that is, the parameter space in this task, is extremely broad; thus, it is difficult to find a sweet spot through continuous exploration. However, SAC enables stochastic exploration, which causes a difference in results. The second possible reason is the SAC policy entropy. Because of the policy entropy term (the second term in (5)), SAC has a rich and wide exploration capability. Thus, compared to DDPG, SAC has a higher exploration capability, which led to better results in our task.

The fact that our actor-critic-based methods could tune the parameters of the simulation implies that the critic works as an approximate function of the simulations and analysis in our model. As explained and illustrated in Figure 1, our actor did not receive feedback directly from the simulation output. During the actor learning procedure, the actor received feedback on the evaluation score of the critic for the output parameter of the actor. This implies that the critic works sufficiently as a simulation approximation, and that the simulations are unnecessary for actor learning. This further implies that the critic successfully works as an end-to-end surrogate model. Owing to the approximation by the critic, the actor successfully obtained a gradient for parameter tuning.

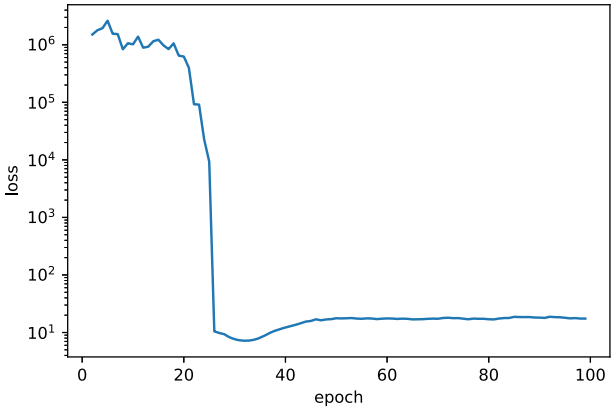

Figure 14 Evaluation loss of SAC

In terms of the evaluation, our experiments should also be updated for a more accurate evaluation. Our experimental task simply tuned kurtosis and skewness; however, this is far from practical. Therefore, the task settings should be updated for more practical tuning.

Considering the situation in which our models exhibited potential, we should test them for the task of high-dimensional parameter tunings. In the context of deep reinforcement learning, our proposed method can be assumed to outperform high-dimensional tasks because deep reinforcement learning has outperformed high-dimensional tasks in previous studies. By contrast, in the context of surrogate models, when the parameter dimension is large, the approximation of simulations by the critic may become more complex and require more data for learning approximation. Moreover, for the more dimensional task, evaluation criteria (objective function) should also be enriched more. For tuning a small number of parameters, a low-dimensional KPI is enough. However, if we consider tuning more dimensional tasks, low-dimensional KPI, such as our experiment only using Kurtosis and Skewness, is not enough to set a tuning task without multiple local optima. Therefore, for testing more dimensional parameter tuning tasks, we also try to build better evaluation criteria, and the building is not easy. Thus, it remains unclear whether our reinforcement-learning-based models work well for high-dimensional tasks and need to be addressed in future studies.

Finally, applicability to the other tasks is also a possible future work. The only requirement of our proposed method is the KPI of outputs. Although building a KPI for tuning is difficult as we discussed above, if the KPI exists, our method seems applicable for any simulation parameter tuning tasks. However, if KPI is inappropriate, the tuning will fail. Therefore, also on simulations of other fields, both construction of KPIs and the experiments of our method are necessary.

7 Conclusion

This study proposed a method for tuning the simulation parameters for MAS using a customized reinforcement learning method. In our proposed method, actor-critic-type reinforcement learning methods such as DDPG and SAC were modified for MAS parameter tuning. Moreover, we proposed three additional components: AC, FNNA, and SF. For the experiments, we employed an artificial financial market simulation for the tuning task. The objective function in tuning is the negative MSE between the target and simulations such that the skewness and kurtosis are close to realistic values. In our experiments, we compared our proposed method with a baseline known as TPE (Optuna), which is based on Bayesian estimation. The results demonstrated that the proposed method outperformed the baseline method. In particular, our SAC-based models outperformed other models, including the baseline. These results indicate that the proposed method is promising. Moreover, it was also indicated that AC, FNNA, and SF for DDPG-based models and SF for SAC-based models were essential components. Interestingly, the results demonstrated that the critic of our proposed model worked well as a surrogate model for the simulations. Subsequently, owing to the critic, the actor could be assumed to learn better parameters. Based on these results, we conclude that the proposed model is promising. In future work, we plan to address the learning stability or the evaluation of other tasks, such as high-dimensional parameter-tuning tasks, in which the method based on reinforcement learning can fully demonstrate its advantages.

Acknowledgements This work was supported by JSPS KAKENHI Grant Number JP 21J20074 (Grant-in-Aid for JSPS Fellows).

Author Contributions M.H. made the conception or design and the softwear, conducted experiments and analysis, and wrote the main manuscript text. All authors reviewed the manuscript.

Funding Open access funding provided by The University of Tokyo. This work was supported by JSPS KAKENHI Grant Number JP 21J20074 (Grant-in-Aid for JSPS Fellows).

Availability of data and materials No data is used in this study.

Declarations

Ethical Approval Not applicable

Competing interests The authors declare no conflicts of interest.

References

1. Kurahashi, S.: Estimating Effectiveness of Preventing Measures for 2019 Novel Coronavirus Diseases (COVID-19). Proceeding of 2020 9th Int. Congress Adv. Appl. Inf. 487–492 (2020). https://doi.org/10.1109/IIAI-AAI50415.2020.00103
2. Mizuta, T., Kosugi, S., Kusumoto, T., Matsumoto, W., Izumi, K., Yagi, I., Yoshimura, S.: Effects of Price Regulations and Dark Pools on Financial Market Stability: An Investigation by Multiagent Simulations. Intell. Syst. Account. Finance Manag. **23**(1–2), 97–120 (2016). https://doi.org/10.1002/isaf.1374
3. Hirano, M., Izumi, K., Shimada, T., Matsushima, H., Sakaji, H.: Impact Analysis of Financial Regulation on Multi-Asset Markets Using Artificial Market Simulations. J. Risk Financial Manag. **13**(4), 75 (2020). https://doi.org/10.3390/jrfm13040075
4. Sajjad, M., Singh, K., Paik, E., Ahn, C.W.: A data-driven approach for agent-based modeling: Simulating the dynamics of family formation. J. Art. Soc. Soc. Simul. 19(1), 9 (2016). https://doi.org/10.18564/jasss.2988
5. Nonaka, Y., Onishi, M., Yamashita, T., Okada, T., Shimada, A., Taniguchi, R.I.: Walking velocity model for accurate and massive pedestrian simulator. IEEJ Trans. Electron. Inf. Syst. **133**(9), 1779–1786 (2013). https://doi.org/10.1541/ieejeiss.133.1779
6. Shigenaka, S., Onishi, M., Yamashita, T., Noda, I.: Estimation of LargeScale Pedestrian Movement Using Data Assimilation. IEICE Trans. Inf. Syst. D. J. 101(9), 1286–1294 (2018). https://doi.org/10.14923/transinfj.2017SAP0014
7. Moss, S., Edmonds, B.: Towards Good Social Science. J. Art. Soc. Social Simul. 8(4), 13 (2005). http://jasss.soc.surrey.ac.uk/8/4/13.html
8. Matsushima, H., Uchitane, T., Tsuji, J., Yamashita, T., Ito, N., Noda, I.: Applying Design of Experiment based Significant Parameter Search and Reducing Number of Experiment to Analysis of Evacuation Simulation. Trans. Japanese Society Art. Intell. **31**(6), 1–9 (2016). https://doi.org/10.1527/TJSAI.AG-E
9. Yamashita, Y., Shigenaka, S., Oba, D., Onishi, M.: Estimation of Large-scale Multi Agent Simulation Results Using Neural Networks [in Japanese]. In: 39th Japanese Special Interest Group on Society andArtificial Intelligence (SIG-SAI), p. 05 (2020). https://doi.org/10.11517/JSAISIGTWO.2020.SAI-039_05
10. Ozaki, Y., Tanigaki, Y., Watanabe, S., Onishi, M.: Multiobjective treestructured parzen estimator for computationally expensive optimization problems. In: Proceedings of 2020 Genetic and Evolutionary Computation Conference, pp. 533–541 (2020). https://doi.org/10.1145/3377930.3389817

11. Baker, B., Kanitscheider, I., Markov, T., Wu, Y., Powell, G., McGrew, B., Mordatch, I.: Emergent Tool Use From Multi-Agent Autocurricula. In: Proceedings of the International Conference on Learning Representations (2020). https://doi.org/10.48550/arxiv.1909.07528
12. Farmer, J.D., Foley, D.: The economy needs agent-based modelling. Nature **460**(7256), 685–686 (2009). https://doi.org/10.1038/460685a
13. Battiston, S., Farmer, J.D., Flache, A., Garlaschelli, D., Haldane, A.G., Heesterbeek, H., Hommes, C., Jaeger, C., May, R., Scheffer, M.: Complexity theory and financial regulation: Economic policy needs interdisciplinary network analysis and behavioral modeling. Science **351**(6275), 818–819 (2016). https://doi.org/10.1126/science.aad0299
14. Lux, T., Marchesi, M.: Scaling and criticality in a stochastic multi-agent model of a financial market. Nature **397**(6719), 498–500 (1999). https://doi.org/10.1038/17290
15. Cui, W., Brabazon, A.: An agent-based modeling approach to study price impact. In: Proceedings of 2012 IEEE Conference on Computational Intelligence for Financial Engineering and Economics, pp. 241–248 (2012). https://doi.org/10.1109/CIFEr.2012.6327798
16. Mizuta, T.: An agent-based model for designing a financial market that works well. arXiv (2019). https://doi.org/10.48550/arXiv.1906.06000
17. Torii, T., Izumi, K., Yamada, K.: Shock transfer by arbitrage trading: analysis using multi-asset artificial market. Evol. Inst. Econ. Rev. **12**(2), 395–412 (2015). https://doi.org/10.1007/s40844-015-0024-z
18. Chiarella, C., Iori, G.: A simulation analysis of the microstructure of double auction markets. Quantitative Finance **2**(5), 346–353 (2002). https://doi.org/10.1088/1469-7688/2/5/303
19. Leal, S.J., Napoletano, M.: Market stability vs. market resilience: Regulatory policies experiments in an agent-based model with low- and high-frequency trading. J. Econ. Behav. Organ. 157, 15–41 (2019). https://doi.org/10.1016/j.jebo.2017.04.013
20. Paddrik, M., Hayes, R., Todd, A., Yang, S., Beling, P., Scherer, W.: An agent based model of the E-Mini S&P 500 applied to flash crash analysis. In: Proceedings of 2012 IEEE Conference on Computational Intelligence for Financial Engineering and Economics, pp. 257–264 (2012). https://doi.org/10.1109/CIFEr.2012.6327800
21. Torii, T., Kamada, T., Izumi, K., Yamada, K.: Platform Design for Largescale Artificial Market Simulation and Preliminary Evaluation on the K Computer. Art. Life Robotics **22**(3), 301–307 (2017). https://doi.org/10.1007/s10015-017-0368-z
22. Torii, T., Izumi, K., Kamada, T., Yonenoh, H., Fujishima, D., Matsuura, I., Hirano, M., Takahashi, T.: Plham: Platform for Large-scale and Highfrequency Artificial Market (2016). https://github.com/plham/plham
23. Torii, T., Izumi, K., Kamada, T., Yonenoh, H., Fujishima, D., Matsuura, I., Hirano, M., Takahashi, T., Finnerty, P.: PlhamJ (2019). https://github.com/plham/plhamJ
24. Sato, H., Koyama, Y., Kurumatani, K., Shiozawa, Y., Deguchi, H.: U-mart: a test bed for interdisciplinary research into agent-based artificial markets. In: Evolutionary Controversies in Economics, pp. 179–190 (2001). https://doi.org/10.1007/978-4-431-67903-5_13
25. Arthur, W.B., Holland, J.H., LeBaron, B., Palmer, R., Tayler, P.: Asset pricing under endogenous expectations in an artificial stock market. The Economy as an Evolving Complex System II, 15–44 (1997). https://doi.org/10.1201/9780429496639-2
26. Byrd, D., Hybinette, M., Hybinette Balch, T., Morgan, J.: ABIDES: Towards High-Fidelity Multi-Agent Market Simulation. In: Proceedings of the 2020 Conference on Principles of Advanced Discrete Simulation, pp. 11–22 (2020). https://doi.org/10.1145/3384441.3395986
27. Murase, Y., Uchitane, T., Ito, N.: A Tool for Parameter-space Explorations. Phys. Proced. 57(C), 73–76 (2014). https://doi.org/10.1016/J.PHPRO.2014.08.134
28. Murase, Y., Matsushima, H., Noda, I., Kamada, T.: CARAVAN: A Framework for Comprehensive Simulations on Massive Parallel Machines. Massively Multi-Agent Systems II, 130–143 (2019). https://doi.org/10.1007/978-3-030-20937-7_9
29. Angione, C., Silverman, E., Yaneske, E.: Using machine learning as a surrogate model for agent-based simulations. PLOS ONE **17**(2), 0263150 (2022). https://doi.org/10.1371/JOURNAL.PONE.0263150
30. Watkins, C.J.C.H., Dayan, P.: Q-learning. Mach. Learn. **8**(3–4), 279–292 (1992). https://doi.org/10.1007/bf00992698
31. Sutton, R.S.: Learning to predict by the methods of temporal differences. Mach. Learn. **3**(1), 9–44 (1988). https://doi.org/10.1007/BF00115009
32. Tesauro, G.: Temporal Difference Learning and TD-Gammon. Commun. ACM **38**(3), 58–68 (1995). https://doi.org/10.1145/203330.203343
33. Rummery, G.A., Niranjan, M.: On-line Q-learning Using Connectionist Systems. University of Cambridge, Department of Engineering Cambridge, England (1994)

34. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., Hassabis, D.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529–533 (2015). https://doi.org/10.1038/nature14236
35. Bellemare, M.G., Veness, J., Bowling, M.: The Arcade Learning Environment: An Evaluation Platform for General Agents. J. Art. Intell. Res. **47**, 253–279 (2013). https://doi.org/10.1613/jair.3912
36. Krizhevsky, A., Sutskever, I., Hinton, G.E.: ImageNet classification with deep convolutional neural networks. In: Adv. Neural Inf. Process. Syst. 2, 1097–1105 (2012). https://doi.org/10.1145/3065386
37. Van Hasselt, H., Guez, A., Silver, D.: Deep reinforcement learning with double Q-Learning. In: Proceedings of 30th AAAI Conference on Artificial Intelligence, pp. 2094–2100 (2016). https://doi.org/10.1609/aaai.v30i1.10295
38. Wang, Z., Schaul, T., Hessel, M., Van Hasselt, H., Lanctot, M., De Frcitas, N.: Dueling Network Architectures for Deep Reinforcement Learning. In: Proceedings of 33rd International Conference on Machine Learning, pp. 2939–2947 (2016)
39. Fortunato, M., Azar, M.G., Piot, B., Menick, J., Osband, I., Graves, A., Mnih, V., Munos, R., Hassabis, D., Pietquin, O., Blundell, C., Legg, S.: Noisy Netw. Explor. arXiv (2017). https://doi.org/10.48550/arXiv.1706.10295
40. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT press, USA (2018)
41. OpenAI: OpenAI Baselines: ACKTR & A2C (2017). https://openai.com/blog/baselines-acktr-a2c/ Accessed 2019-11-06
42. Hessel, M., Modayil, J., Van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., Silver, D.: Rainbow: Combining improvements in deep reinforcement learning. In: Proceedings of 32nd AAAI Conference on Artificial Intelligence, pp. 3215–3222 (2018). https://doi.org/10.1609/aaai.v32i1.11796
43. Horgan, D., Quan, J., Budden, D., Barth-Maron, G., Hessel, M., van Hasselt, H., Silver, D.: Distributed Prioritized Experience Replay. arXiv (2018). https://doi.org/10.48550/arXiv.1803.00933
44. Kapturowski, S., Ostrovski, G., Quan, J., Munos, R., Dabney, W.: Recurrent Experience Replay in Distributed Reinforcement Learning. In: Proceedings of International Conference on Learning Representations, pp. 1–15 (2019)
45. Hochreiter, S., Schmidhuber, J.: Long Short-Term Memory. Neural Comput. **9**(8), 1735–1780 (1997). https://doi.org/10.1162/neco.1997.9.8.1735
46. Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., Hassabis, D.: A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. Sci. **362**(6419), 1140–1144 (2018). https://doi.org/10.1126/science.aar6404
47. Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., Van Den Driessche, G., Graepel, T., Hassabis, D.: Mastering the game of Go without human knowledge. Nature **550**(7676), 354–359 (2017). https://doi.org/10.1038/nature24270
48. Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, D.: Continuous control with deep reinforcement learning. In: Proceedings of 4th International Conference on Learning Representations (2015). https://doi.org/10.48550/arxiv.1509.02971
49. Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., Levine, S.: Soft Actor-Critic Algorithms and Applications. arXiv (2018). https://doi.org/10.48550/arxiv.1812.05905
50. Haarnoja, T., Zhou, A., Abbeel, P., Levine, S.: Soft Actor-Critic: OffPolicy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. Proc. 35th Int. Conf. Mach. Learn. 2976–2989 (2018). https://doi.org/10.48550/arxiv.1801.01290
51. Uhlenbeck, G.E., Ornstein, L.S.: On the theory of the brownian motion. Physi. Rev. **36**(5), 823 (1930). https://doi.org/10.1103/PhysRev.36.823
52. Wawrzyński, P., Tanwani, A.K.: Autonomous reinforcement learning with experience replay. Neural Netw. **41**, 156–167 (2013). https://doi.org/10.1016/j.neunet.2012.11.007
53. Frankle, J., Carbin, M.: The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks. Proceedings of 7th International Conference on Learning Representations (2018). https://doi.org/10.48550/arxiv.1803.03635
54. Bookstaber, R.M.: The End of Theory: Financial Crises, the Failure of Economics, and the Sweep of Human Interaction. Princeton University Press, USA (2017)
55. Corsi, F.: Measuring and modelling realized volatility: from tick-by-tick to long memory. PhD thesis, Universitá della Svizzera italiana (2005)

56. Akiba, T., Sano, S., Yanase, T., Ohta, T., Koyama, M.: Optuna: A next-generation hyperparameter optimization framework. In: Proceedings of the 25th International Conference on Knowledge Discovery & Data Mining, pp. 2623–2631 (2019). https://doi.org/10.1145/3292500.3330701