Simultaneous Optimization of Discrete and Continuous Parameters Defining a Robot Controller

Takato Ishii[1†], Ryo Ariizumi[1], and Fumitoshi Matsuno[2]

[1]Department of Mechanical Systems Engineering Tokyo University of Agriculture and Technology, Tokyo, Japan
(Tel: +81-42-388-7421; E-mail: ryoariizumi@go.tuat.ac.jp)
[2]Department of Electronics and Information Systems Engineering, Osaka Institute of Technology, Osaka, Japan

Abstract: Reinforcement learning is one of the most powerful optimization methods for designing a robot controller. However, the neural architecture design is often a labor-intensive task performed by experienced and intuitive engineers. Because of this, typical and conventional reinforcement learning problems focus on optimizing only continuous parameters i.e., weights and biases. Therefore, a new method that explores a richer set of architectures and continuous parameters is desirable. Automatic search of neural architecture will not only reduce the design workload but also result in a better-performing controller with less computational burden. In this paper, we explain a novel method to design both the architectures and the continuous parameters simultaneously. Experiments with physical simulations show that the proposed method provides better and more stable results than the conventional method in a walking task.

Keywords: Artificial life, neural architecture search, reinforcement learning, simultaneous optimization

1. INTRODUCTION

Legged robots are expected to be useful in many applications including the exploration of planets [1]. Meanwhile, it is extremely difficult to design the optimal robot controller that can adjust to a highly turbulent real-world environment. Therefore, reinforcement learning (RL) garners much attention due to its effectiveness in this kind of optimization problems. However, there are many constraints when it comes to robot controllers, especially in terms of financial and computational costs. Because of this background, there is a strong demand for a novel method that designs a sub-optimal controller without increasing additional costs.

Many approaches have been taken to optimize the architecture of a neural network. The automatic design of the neural architectures is generally called Neural Architecture Search (NAS). It is a hot topic progressing rapidly in a variety of areas. Sparked by a benchmark study "Neural architecture search with reinforcement learning," [2] more than 1000 new NAS papers have been released over the last few years [3]. However, the mainstream of these attempts is to design networks used for supervised learning, which requires sets of validation data [5] or another network to optimize a target one [6]. There are several approaches that are applicable to RL problems such as pruning [7] and AutoML for Model Compression (AMC) [8]. One problem is that they require a pre-trained network before applying each optimizing algorithm. As a premise, this assumes the network parameters can be finely tuned by the ordinary RL. However, there are possible cases that are difficult to gain a good controller because of the pre-defined network architecture. This means it is necessary to experiment over and over until we can gain a high-performance controller, which is extremely time-consuming.

In this paper, we propose a novel method that optimizes both the architecture and parameters of a neural network during the online training of a robot. To show the effectiveness of the proposed method, we handle a walking task with physical simulation.

This paper is organized as follows. Section 2 describes the problem. Section 3 explains the problems regarding the ordinary RL . Section 4 explains a novel method for the simultaneous optimization of discrete and continuous parameters. The experiments using physical simulations in Section 5 illustrate the effectiveness of our method, comparing the performance with an ordinary RL. Finally, Section 6 summarizes this paper.

Notation: In this paper, $\mathbb{R}$, $\mathbb{Z}$, and $\mathbb{N}$ denote the sets of real numbers, integers, and natural numbers, respectively. For a set $\mathbb{A} \subseteq \mathbb{R}$, let us define $\mathbb{A}_{\geq 0}$ as follows:

$$\mathbb{A}_{\geq 0} = \{a \in \mathbb{A} \mid a \geq 0\} \tag{1}$$

2. MODEL AND PROBLEM DESCRIPTION

As an example, we consider the four-legged robot called Ant [9] shown in Fig. 1. This robot has a trunk called the torso, and four legs attached to it. Each leg consists of two rigid parts and joints. We impose a walking task on this robot. The goal of an agent (robot) is to control eight joints that connect the rigid parts by applying torque and to move in the forward (x) direction as fast as possible. In Fig. 1, the right arrow denotes the forward direction. The details of the training are explained in Section 5.

2.1. Controller

The proposed method learns a controller that computes action a_t from state s_t. The state s_t consists of the angle and angular velocity of each joint, the z (vertical) coordinate of the torso, each velocity of the torso in the xyz-direction, and the quaternions representing the torso's posture. The action a_t is the torque input applied

† Takato Ishii is the presenter of this paper.

to the eight joints. Let $s_t \in \mathcal{S} \subset \mathbb{R}^{N_s}$ and $a_t \in \mathcal{A} \subset \mathbb{R}^{N_a}$ denote the robot's state and action at time $t \in \mathbb{Z}_{\geq 0}$, respectively. Here, N_s and N_a are the dimensions of the state and action, respectively, and $\mathcal{S}$ and $\mathcal{A}$ are the sets of all possible values for the state and action, respectively. We use a multilayer perceptron (MLP) as a static controller, and its parameters will be trained during the optimization process.

The forward propagation of the MLP in hidden layers runs as follows. Let $n^\lambda \in \mathbb{N}$ denote the number of nodes in layer λ. The weight and bias parameters can be described as

$$\zeta^\lambda = \begin{bmatrix} \zeta^\lambda_{1,1} & \cdots & \zeta^\lambda_{n^\lambda,1} \\ \vdots & \ddots & \vdots \\ \zeta^\lambda_{1,n^{\lambda-1}} & \cdots & \zeta^\lambda_{n^\lambda,n^{\lambda-1}} \end{bmatrix} \in \mathbb{R}^{n^{\lambda-1} \times n^\lambda}, \quad (2)$$

$$\xi^\lambda = \begin{bmatrix} \xi^\lambda_1, & \cdots, & \xi^\lambda_{n^\lambda} \end{bmatrix}^\top \in \mathbb{R}^{n^\lambda \times 1}, \quad (3)$$

respectively, and the activation function is $f^\lambda : \mathbb{R}^{n^\lambda \times 1} \to \mathbb{R}^{n^\lambda \times 1}$. Let the output of layer $\lambda - 1$ be denoted as $x^{\lambda-1} = [x^{\lambda-1}_1, \ldots, x^{\lambda-1}_{n^{\lambda-1}}]^\top$, then the output of layer λ is

$$x^\lambda = f^\lambda(\zeta^\lambda x^{\lambda-1} + \xi^\lambda) \in \mathbb{R}^{n^\lambda \times 1} \quad (4)$$

Now, we define a tuple of natural numbers that contains the number of nodes $\nu = (n^1, n^2, \ldots, n^\Lambda) \in \mathbb{N}^\Lambda$ and call ν a *discrete parameter*. Besides, let us assume that there are N_ψ parameters (weights and biases) of the controller, which are tuned by an RL algorithm. Let $\psi = \begin{bmatrix} \psi^1, \psi^2, \ldots, \psi^{N_\psi} \end{bmatrix} \in \mathbb{R}^{N_\psi}$ be a vector constructed by arranging these parameters in one dimension. We call the vector ψ a *continuous parameter*. The total number of layers λ and activation function f^λ are included in hyperparameters.

2.2. Simultaneous optimization problem

The problem discussed in this paper can be formulated using stochastic processes (Fig. 3). Let S_t be an $\mathcal{S}$-valued random variable representing the robot's state and A_t an $\mathcal{A}$-valued random variable representing its action at time t. Define the joint probability density functions $p^\nu_0 : \mathcal{S} \to \mathbb{R}_{\geq 0}$ and $p^\nu : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to \mathbb{R}_{\geq 0}$ for the initial states and state transitions, respectively. The reward function is denoted by $r^\nu : \mathcal{S} \times \mathcal{A} \times \mathcal{S}' \to \mathbb{R}$, and

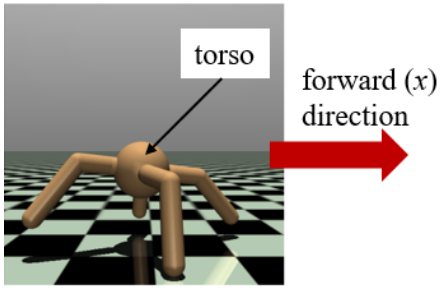


Fig. 1: Appearance of the quadruped robot [11].

random variable representing the reward at time $t \geq 1$ is denoted by $R_t \equiv r^\nu(S_{t-1}, A_{t-1}, S_t)$. The calculations for obtaining the next state s_{t+1} and reward r_{t+1} from state s_t and action a_t are modeled as a black box. Although we use deterministic policies as controllers in this paper, more generally, we assume that stochastic policies are used. Namely, let policy $\pi^\psi : \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ be a joint probability density function. The transitions of the state, action, and reward for a robot whose discrete parameter is ν can be modeled by a tuple $(\mathcal{S},\ \mathcal{A},\ p^\nu_0,\ p^\nu,\ r^\nu)$. This transition model is a Markov decision process (MDP). Let $E^{\nu,\psi}[X]$ denote the expected value of a random variable X, when using the discrete parameter ν and the continuous parameter ψ.

We consider sampling a sequence of states and actions that follow this MDP and policy π^ψ during the period from time 0 to a predetermined time T. This period is considered an *episode*. We call the sum of the rewards sampled in an episode the cumulative reward and write its expected value as

$$J(\nu, \psi) \equiv E^{\nu,\psi}\left[\sum_{t=1}^{T} R_t\right] \quad (5)$$

The expected cumulative reward J is assumed to be partially differentiable concerning each element of ψ. The simultaneous optimization problem discussed in this article can be formulated as

$$\begin{aligned} \max_{\nu,\psi} \quad & J(\nu,\ \psi) \\ \text{s.t.} \quad & \nu \in \mathbb{N}^\Lambda,\ \psi \in \mathbb{R}^{N_\psi} \end{aligned} \quad (6)$$

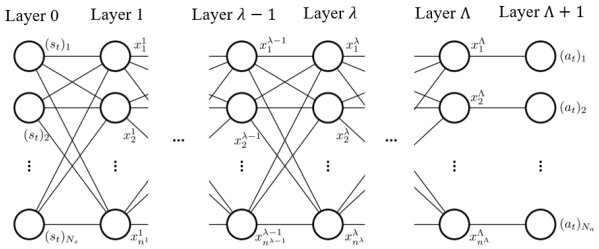


Fig. 2: MLP used as the robot controller [12].

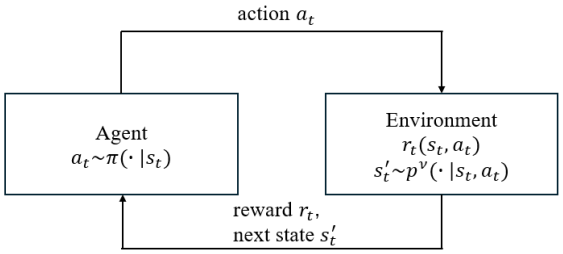


Fig. 3: Framework of the optimization problem as a stochastic process subject to parameters $\nu,\ \psi$.

3. PROBLEMS WITH THE ORDINARY RL

There are two major problems with the ordinary RL that focuses on optimizing only the continuous parameter ψ. In this section, we will point out the downsides of the ordinary RL and explain why the improved one is demanded. To discuss the performance of a robot later in this paper, let us define a value function. The T-step performance evaluation is repeated for L episodes for predefined T and L. Let $\mathscr{R}^l$ denote the cumulative reward $\sum_{t=1}^{T} R_t$ sampled in the lth episode. Then, using the averaged value of all episodes, the value function $\mathscr{R}_{\mathrm{eval}}$ is calculated as

$$\mathscr{R}_{\mathrm{eval}} = \frac{1}{L}\sum_{l=1}^{L}\mathscr{R}^l \tag{7}$$

Besides, when we refer to "eliminate nodes," it means to update the specific parts of layers, weights, and biases involving the eliminated nodes, and to obtain the new parameters ν, ψ. For example, "to eliminate a node in layer λ at mth index" specifically means the following process. First, we eliminate x_m^λ from $[x_1^\lambda, \ldots, x_n^\lambda]^\top$. This causes the discrete parameter ν to be updated as $n^\lambda \leftarrow n^\lambda - 1$ in layer λ. Then, mth column of weight ζ^λ, mth element of bias ξ^λ, and mth row of weight $\zeta^{\lambda+1}$ are also removed. After replacing $\zeta^\lambda, \zeta^{\lambda+1}, \xi^\lambda$, and output x^λ by the new values, we reconstruct the vector ψ by arranging them again in one dimension. Note that in this process, the activation function is unchanged and the same one is still used which is specified as a hyperparameter.

We conducted some experiments to show the limit of the ordinary RL. As we mentioned earlier, there are two problems.

First, the quality of training is highly dependent on the fixed values of the model architecture. Figure 4 shows the comparison of performance stability among robots trained with several different architectures. This experiment was organized as follows. First, we prepared some models with different architectures shown in the horizontal axis of Fig. 4. As for hyperparameters, we used the same values in simulation experiments in Section 5. Then, for each setting of architecture (discrete parameter ν), we applied the procedure below.

1. Train only continuous parameter ψ for predefined total time steps T_{total}.
2. For trained models, evaluate performance by calculating $\mathscr{R}_{\mathrm{eval}}$ ten times and averaging them.
3. By iterating the above processes N times in total, obtain a set of data $\{\mathscr{R}^1_{\mathrm{eval}}, \mathscr{R}^2_{\mathrm{eval}}, \ldots, \mathscr{R}^N_{\mathrm{eval}}\}$ and make a violin plot.

In this experiment, we used $T_{\mathrm{total}} = 2,000,000$ and $N = 30$, where these values are shared in common of all settings of the discrete parameter ν. We used PPO [4] as an algorithm to optimize the continuous parameter ψ. As shown in Fig. 4, performance varies from model to model. Moreover, its stability seems to be dependent on the architecture to some extent. We assume that the architecture of the controller that yields stable and high performance is also dependent on the training algorithm and tasks imposed. Therefore, it is very hard and time-consuming for a human engineer to discover the optimum value from the vast search space.

Second, as for a trained model, there possibly exist some nodes that can be eliminated due to their uselessness. To make sure of this, we organized the second experiment following the procedure below.

1. Prepare a model trained with fixed discrete parameter $\nu = (24, 32)$ for total time steps 2,000,000, optimizing only continuous parameter ψ.
2. For the trained model, evaluate performance by calculating $\mathscr{R}_{\mathrm{eval}}$ ten times and averaging them.
3. Eliminate a node one by one in each layer from the original model.
4. Evaluate the model obtained by process 3 exactly in the same way as process 2.

Figure 5 shows the correspondence of an eliminated node index and the evaluation value. From this figure, for example, we can see the difference of importance among nodes by looking at averaged cumulative reward for index 3, 6, 14, etc. in layer 1 and index 1, 5, 20, 21, etc. in layer 2. Also, elimination in layer 2 has less effect on performance than that of layer 1. It is desirable to design a controller that is as simple as possible since we want to curtail its computational burden with application to the real world in mind. This experiment indicates that there is room for customizing the controller to alleviate the computational burden of its current architecture $\nu = (24, 32)$.

The proposed method can address both of these problems mentioned above, as we explain in the next section.

4. FRAMEWORK FOR UPDATING THE PARAMETERS

In this section, we propose a simultaneous optimization method for discrete and continuous parameters. We call the process of updating the parameters ν, ψ as *gener-*

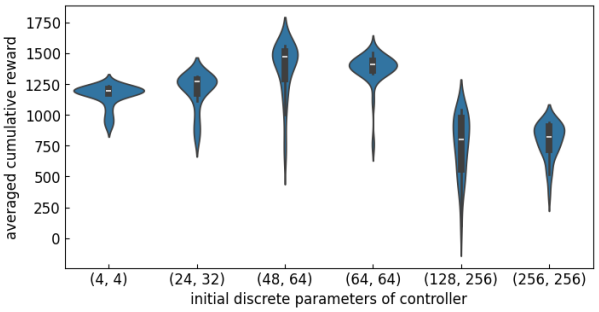
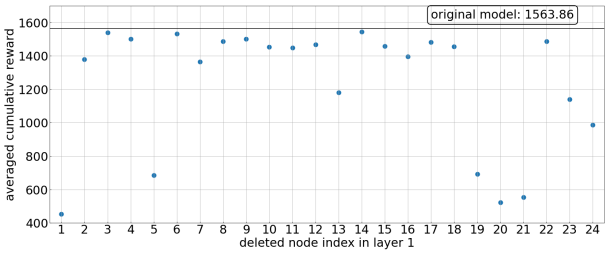
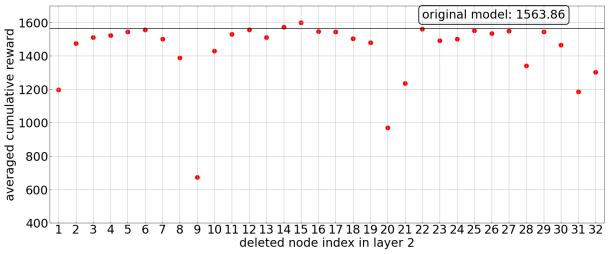


Fig. 4: Comparison of performance stability among robots trained with different initial discrete parameters.



(a)



(b)

Fig. 5: Fluctuation of performance when deleting just one node from a trained model in each layer and index.

ation, and denote the number of generations by the index k. Note that unlike the parameter ψ, which is updated at every generation, the discrete parameter ν is only updated at certain generations.

The continuous parameter ψ is tunable in RL algorithms. Among several RL optimization methods, the selected method is denoted as Algorithm $\mathbb{A}$, which is one of the hyperparameters.

The procedure for updating the parameters in the proposed method is as follows. First, we create M robots with parameters (ν^k, ψ^k) at kth generation. Then, Algorithm $\mathbb{A}$ is applied to the setting $(\nu^k, \psi^{k,j})$ in each $j \in \{1, 2, \ldots, M\}$ to obtain a new continuous parameter $\psi^{k+1,j}$. Using the average value of these continuous parameters, the continuous control parameter for the next generation is calculated as

$$\psi^{k+1'} = \frac{1}{M} \sum_{j=1}^{M} \psi^{k,j} \tag{8}$$

Furthermore, the performance of the configuration $(\nu^k, \psi^{k+1'})$ is evaluated by physical simulation or other methods. The T-step performance evaluation is repeated for L episodes for predefined T and L. Let $\mathscr{R}^{k,l}$ denote the cumulative reward $\sum_{t=1}^{T} R_t$ sampled in the lth episode. Its mean regarding episodes and individuals is computed as

$$b^k = \frac{1}{M} \sum_{j=1}^{M} \left(\frac{1}{L} \sum_{l=1}^{L} \mathscr{R}^{k,l} \right) \tag{9}$$

We now provide a criterion for determining whether the discrete parameter ν^k at the kth generation should be updated at that point. The setting of this criterion comes from the idea that the best timing for changing the neural architecture is when the performance reaches a plateau. The mean of the cumulative reward b^k evaluates the performance of (ν^k, ψ^k). We define g^k as the value obtained by smoothing this evaluation value with the exponential moving average (EMA). In the case of $\nu^{k-1} \neq \nu^k$, however, the EMA is not used, and the raw evaluated value is used. In other words, g^k is

$$g^k = \begin{cases} b^k & (k = 0 \vee (k \geq 1 \wedge \nu^{k-1} \neq \nu^k)) \\ \beta b^k + (1 - \beta)\, g^{k-1} & (\text{otherwise}) \end{cases} \tag{10}$$

with a constant β $(0 \leq \beta \leq 1)$. Furthermore, let a natural number $N_{\text{sight}} \geq 2$ be defined as a hyperparameter and assume that the condition

$$\begin{aligned} &(k \geq N_{\text{sight}} - 1) \\ &\wedge (\nu^{k-N_{\text{sight}}+1} = \nu^{k-N_{\text{sight}}+2} = \cdots = \nu^k) \end{aligned} \tag{11}$$

is satisfied. Let δ^k be defined as the slope of a line fit to the data

$$\{(1, g^{k-N_{\text{sight}}+1}), (2, g^{k-N_{\text{sight}}+2}), \ldots, (N_{\text{sight}}, g^k)\} \tag{12}$$

by the least-squares method. If the value δ^k is smaller than a certain threshold $\delta_{\min} \geq 0$, the discrete parameter is updated from ν^k to ν^{k+1}. The proposed framework is shown as pseudocode in Algorithm 1.

5. SIMULTANEOUS OPTIMIZATION OF DISCRETE AND CONTINUOUS PARAMETERS

First, we explain our strategy adopted to optimize the discrete parameter. As we discussed in Section 4, we can estimate the number of nodes that yields the best performance is relatively small. Therefore, we adopted a strategy to decrease the number of nodes monotonously inspired by neural development in the human brain. It is shown that the number of synapses in the human brain gradually declines [13], after a rapid growth around the newborn period. Considering this biological point of view, we assume that one of the relevant strategies is to set a relatively complicated model at first, and then eliminate "less important" nodes during training.

One of the possible factors that causes unstable learning results as we see in Fig. 4 may lie in over-training. In MLP learning, dropout is a well-known method for its effectiveness in preventing over-learning [14]. This is a regularization method that allows us to train a large network by temporarily removing some nodes at random. From a different point of view, it indicates that the temporary elimination of nodes causes other nodes to supplement the roles of the deleted ones. If the causes of the unstable results of learning can be attributed to the over-sized parameters, we may be able to gain the same effect in our case. However, dropout cannot be applied directly because we want to downsize the parameter irreversibly. Moreover, it is not an efficient way to select

Algorithm 1 Simultaneous optimization of discrete and continuous parameters

Inputs: The maximum number of generations K, the number of individuals M, the number of episodes for the performance evaluation L, initial values $(\nu^0,\ \psi^0)$, the parameters $(\delta_{\min},\ \beta,\ N_{\text{sight}})$, Algorithm $\mathbb{A}$, and the environment for the performance evaluation.

Outputs: trained $\nu^K,\ \psi^K$

for $k \leftarrow 0$ to $K-1$ **do**
  for $j \leftarrow 1$ to M **do**
    $\psi^{k+1,j} \leftarrow$ (The outputs of applying Algorithm $\mathbb{A}$ to the configuration (ν^k, ψ^k))
  end for
  $\psi^{k+1'} \leftarrow$ (Right-hand side of the Eq.(8))
  for $j \leftarrow 1$ to M **do**
    for $l \leftarrow 1$ to L **do**
      $\mathscr{R}^{k,j,l} \leftarrow$ (The result of the performance evaluation)
    end for
  end for
  if (Condition (11) holds) and $\delta^k < \delta_{\min}$ **then**
    $(\nu^{k+1}, \psi^{k+1}) \leftarrow$ (The output obtained by the proposed method with the inputs of the results of the performance evaluation)
  else
    $(\nu^{k+1}, \psi^{k+1}) \leftarrow (\nu^k, \psi^{k+1'})$
  end if
end for

nodes at random. This is because there are different impacts on performance depending on the selected node as we see in Fig. 5. We assume that the "important nodes" can be judged from the outputs of nodes in hidden layers. Prioritizing nodes enables us to obtain a smaller size network without affecting the stability of learning.

Based on the assumption above, we propose the *node ranking method* (NRM). Prior to the elimination, this method requires *ranking*, which is yielded at every generation as follows.

1. For each individual $j \in \{1, \ldots, M\}$, calculate the output $x^{k,j,\lambda}$ in layer λ. Let U be the number of samples obtained in this process.
2. Average the output in each layer in the number of individuals M, and define $H^{k,\lambda}$ as follows.

$$H^{k,j,\lambda} = \begin{bmatrix} x_1^{k,j,\lambda} & x_2^{k,j,\lambda} & \cdots & x_U^{k,j,\lambda} \end{bmatrix}^\top \quad (13)$$

$$H^{k,\lambda} = \frac{1}{M}\sum_{j=1}^{M} H^{k,j,\lambda} \quad (14)$$

3. For the matrix $H^{k,\lambda}$, let $\overline{H_m^{k,\lambda}}$ be the average of mth row:

$$\overline{H_m^{k,\lambda}} = \frac{1}{U}\sum_{u=1}^{U} H_{u,m}^{k,\lambda} \quad (15)$$

Then, define

$$h_m^{k,\lambda} = \sqrt{\frac{\sum_{u=1}^{U}(H_{u,m}^{k,\lambda} - \overline{H_m^{k,\lambda}})^2}{U}}, \quad (16)$$

which is the standard deviation of mth row of $H^{k,\lambda}$.

4. Define a set of index m as $\mathcal{M}^{k,\lambda}$ as follows:

$$\mathcal{M}^{k,\lambda} = \{m |\ (h_m^{k,\lambda} \text{ holds the condition (a)})\} \quad (17)$$

where the condition (a) is described as follows:
(a) For each element $h_m^{k,\lambda} \in \{h_1^{k,\lambda}, h_2^{k,\lambda}, \ldots, h_{n^\lambda}^{k,\lambda}\}$ of the vector $h^{k,\lambda}$, the value is positioned less than 10 percent of n^λ as a whole.

This set will be updated depending on the discrete parameter as follows.

$$\mathcal{M}^{k,\lambda} = \begin{cases} \mathcal{M}^{k,\lambda} & (k = 0 \vee (k \geq 1 \wedge \nu^{k-1} \neq \nu^k)) \\ \mathcal{M}^{k-1,\lambda} \cup \mathcal{M}^{k,\lambda} & (\text{otherwise}) \end{cases} \quad (18)$$

The set $\mathcal{M}^{k,\lambda}$ can be considered as "elimination candidates" in our method.

5. Let the number for the node m of layer λ to be included in $\mathcal{M}^{k',\lambda}$ $(\hat{k} < k' \leq k)$ be $C^{k,\lambda}(m)$. Let us define a set $\mathcal{R}^{k,\lambda}$, which encompasses all possible pairs of $(m, C^{k,\lambda}(m))$. We call this set as "ranking."

Suppose there is a demand for updating the discrete parameter at kth generation. In this case, we select the elimination target nodes as follows. First, we refer to the ranking $\mathcal{R}^{k,\lambda}$. If $C^{k,\lambda}(m)$ is bigger than a certain threshold $C_{\max}^{\lambda} \in \mathbb{N}$, then the index mth node is eliminated. The pseudocode of the NRM is shown in Algorithm 2.

Here, we give an example to verify the validity of this method. We conducted the following operation using the trained model which we used for showing the Fig. 5. First, we define a condition (b) as follows:

(b) For each element $h_m^{k,\lambda} \in \{h_1^{k,\lambda}, h_2^{k,\lambda}, \ldots$ $\ldots, h_{n^\lambda}^{k,\lambda}\}$ of the vector $h^{k,\lambda}$, the value is positioned *more* than 10 percent of n^λ as a whole.

This is the arranged version of the condition (a). Depending on each condition (a) or (b), we call selected nodes "low-ranking" or "high-ranking" nodes, respectively. The result is shown in Table 1, where we provided the first few selected nodes in each layer.

From Table 1, we can see that the high-ranking nodes largely correspond to nodes whose evaluation values drop when eliminated at Fig. 5. This result indicates that NRM is valid in preventing important nodes from being eliminated and leaving necessary ones. In fact, simulation experiments confirmed the tendency of elimination candidates to converge on certain indexes as generation goes by. In summary, the proposed method is based on the assumption that the update of the ranking across generations is effective in distinguishing important nodes and those that are not.

Algorithm 2 Node Ranking Method

Inputs: The discrete parameter ν^k, the temporary continuous parameter $\psi^{k+1'}$, the ranking $\mathcal{R}^{k,\lambda}$
Outputs: ν^{k+1}, ψ^{k+1}

```
for m ← 1 to n^λ do
    if C^{k,λ}(m) > C^λ_max then
        (ν^{k+1}, ψ^{k+1}) ← (eliminate index mth node and update parameters)
    else
        (ν^{k+1}, ψ^{k+1}) ← (ν^k, ψ^{k+1'})
    end if
end for
```

Table 1: Low and high ranking nodes in each layer

selected condition	layer 1	layer 2
(a), low-ranking nodes	6, 9, 22	12, 13, 26, 27
(b), high-ranking nodes	11, 21, 23	8, 10, 20, 24

Suppose the parameters are updated from $\nu^k, \psi^{k+1'}$ to $\nu^{k+1}, \psi^{k+1} (\nu^k \neq \nu^{k+1})$ by the proposed method. The parameter ψ^{k+1} is learned to match the discrete parameter ν^k, so it is unlikely to match the discrete parameter ν^{k+1}. However, it is expected that this decrease in the evaluation function will be temporary and that the function will quickly increase again with subsequent learning. This is because the discrete parameter ν^k and ν^{k+1} are similar to each other, so the parameter ψ^{k+1} can benefit from transfer learning.

6. SIMULATION EXPERIMENTS

6.1. Task and method configurations

For simulation experiments, we developed an OpenAI Gym-style environment with MuJoCo [15]. Because we consider the walking task, we selected PPO as Algorithm $\mathbb{A}$, which is proven to be effective with similar tasks [12]. We created the simulation environment using Stable-Baselines3 [16] and parallelized it with Microsoft MPI.

The reward configuration is the default of Ant-v2. According to Gym Documentation [11], it is given as a following equation, where $v_x(t)$ is the x velocity of the torso and $a_i(t)$ is the torque of joint i at time step t.

$$r_t = 1.0 + v_x(t) - 0.5 \sum_i a_i(t)^2 \tag{19}$$

In the experiments in this section, we used the hyperparameters shown in Table 2.

6.2. Results of the proposed method

To compare the results between the ordinary RL and the proposed methods, we prepared two parallel environments. One is the case where we optimized only the continuous parameter ψ, fixing the discrete parameter as $\nu = (256, 256)$. The other used the proposed method to update both ν and ψ. An example of the results is shown in Fig. 6. From this figure, we can see the cumulative reward of the proposed method drops right after the discrete parameter is updated. This is because of the mismatch of the continuous parameter and the updated discrete parameter. However, the drop in the cumulative reward is only temporary, and it immediately begins to increase again. We believe this is caused by the effect of transfer learning in the MLP, as we discussed in Section 4. Because of this expected benefit, the temporary drop in the cumulative reward is not a critical issue. In this example, the optimum value of the discrete parameter was $\nu = (139, 147)$ recorded at the 7621th generation.

Table 2: Hyperparameters used for the simulation experiments

parameter	value
The number of maximum generation K	8000
The number of individuals M	3
Algorithm $\mathbb{A}$ (for the continuous parameters)	PPO
Steps in Algorithm $\mathbb{A}$	3000
Episodes in the performance evaluation L	2
Steps in the performance evaluation T	500
Sight of EMA N_{sight}	500
Coefficient of EMA β	$\frac{2}{500+1}$
Threshold of EMA $\delta_{\min}$	0.025
The number of MLP layers (actor function) Λ	2
MLP architecture (value function)	(48, 64)
Activation function	ReLU
Threshold of layer1 update $C^1_{\max}$	450
Threshold of layer2 update $C^2_{\max}$	450

6.3. Comparison between the proposed and the ordinary RL method

We experimented multiple times applying both the ordinary RL and the proposed methods to compare the stability of learning, and collected the same amount of data. In Fig. 7, the horizontal axis denotes generation and the vertical axis shows the averaged max evaluation function in every 500 generations. The number of data for each environment is 40. From this figure, we can see the validity of the proposed method, as the performance of the proposed method is superior to that of the ordinary RL. In summary, it is confirmed the proposed method is effective for the optimization problem (Eq. (6)).

7. CONCLUSION

In this paper, we proposed a novel method to optimize both the discrete and continuous parameters. First, we referred to two major issues related to the ordinary RL. Then, we explained the necessity for the discrete parameter to be updated. We adopted a strategy to decrease the number of nodes monotonously during training, which is inspired by the correspondence with neural development in the human brain. Besides, assuming that the importance of each node is recognizable based on the outputs of hidden layers, we proposed the method to select target ones from the ranking that includes elimination candidates. In simulation experiments, we confirmed the effectiveness of our method which yielded a better controller

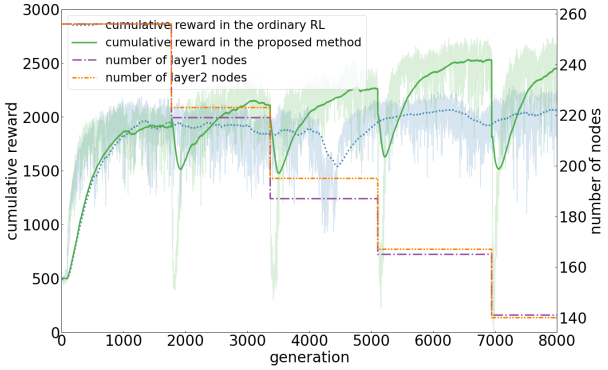


Fig. 6: An example of the results. The thick and thin lines of cumulative reward represent g^k and b^k respectively.

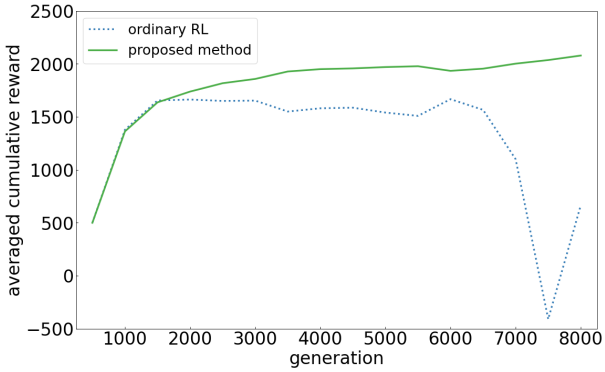


Fig. 7: Comparison between the proposed and the ordinary RL method.

with less computational burden during the same period of learning steps as the ordinary RL. It is also confirmed that there is a benefit from transfer learning.

Our future goal is to ensure the effectiveness of our method when applied to different tasks since we imposed the robot only on the walking task. Furthermore, our current discussion is based on empirical facts. Consolidating the basis of our argument from a mathematical point of view remains to be a challenge.

ACKNOWLEDGMENT

This work was supported in part by JST, Moonshot RD, JP-MJMS223B and in part by the "Innovation Inspired by Nature" Research Support Program, Sekisui Chemical Company, Ltd..

REFERENCES

[1] E. Hart and L. K. Le Goff, "Artificial evolution of robot bodies and control: On the interaction between evolution, learning, and culture," *Philosophical Transactions of the Royal Society B: Biological Sciences*, vol. 377, no. 1843, 2022.

[2] B. Zoph and Q. V.Le, "Neural Architecture Search With Reinforcement Learning," Google Brain, 2017

[3] C. White, M. Safari, R. Sukthanker, B. Ru, T. Elsken, A. Zela, D. Dey, and F. Hutter, "Neural Architecture Search: Insights from 1000 Papers," *arXiv:2301.08727v2*

[4] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint, arXiv:1707.06347*, pp. 1–12, 2017.

[5] E. I. Vlahogianni, M. G. Karlaftis, and J. C. Golias, "Optimized and meta-optimized neural networks for short-term traffic flow prediction: A genetic approach," *Transportation Research Part C* 13, 2005.

[6] K. O. Stanley, J. Clune, J. Lehman, et al., "Designing neural networks through neuroevolution," *Nat Mach Intell* 1, 24–35 (2019).

[7] Neural Network Pruning Through Constrained Reinforcement Learning

[8] Y. He, J.Lin, Z.Liu, H.Wang, L.Li, and S.Han, "AMC: AutoML for model compression and acceleration on mobile devices," in *European COnf. on Computer Vision. 2018*, Springer International Publishing.

[9] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," *arXiv:1506.02438*, 2015.

[10] G. Brockman, L. Cheung, Vicki and Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI gym," *arXiv:1606.01540*, 2016.

[11] Gymnasium Documentation, https://gymnasium.farama.org/environments/mujoco/ant/, (accessed: 2.1.2024)

[12] R. Koike, R. Ariizumi, and F. Matsuno, "Simultaneous Optimization of Discrete and Continuous Parameters Defining a Robot Morphology and Controller," *IEEE Transactions on Neural Networks and Learning Systems*, 2023.

[13] J. Stiles and T. L. Jernigan, "The Basics of Brain Development," Neuropsychol Rev 2010.

[14] 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 15*, 2014

[15] E. Todorov, T. Erez, and Y. Tassa, "MuJoCo: A physics engine for model-based control," *IEEE International Conference on Intelligent Robots and Systems*, pp. 5026–5033, 2012.

[16] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable- Baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021.