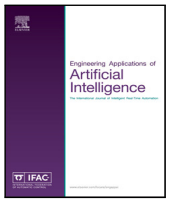
Engineering Applications of Artificial Intelligence 95 (2020) 103869

Contents lists available at ScienceDirect

Engineering Applications of Artificial Intelligence

journal homepage: www.elsevier.com/locate/engappai



Reinforcement learning for quadrupedal locomotion with design of continual–hierarchical curriculum

Taisuke Kobayashi *, Toshiki Sugino

Division of Information Science, Graduate School of Science and Technology, Nara Institute of Science and Technology, Nara 630-0192, Japan

ARTICLE INFO

Keywords:
Continual learning
Curriculum learning
Hierarchical learning
Reservoir computing
Fractal network

ABSTRACT

End-to-end reinforcement learning is a promising approach to enable robots to acquire complicated skills. However, this requires numerous samples to be implemented successfully. The issue is that it is often difficult to collect the sufficient number of samples. To accelerate learning in the field of robotics, knowledge gathered from robotics engineering and previously learned tasks must be fully exploited. Specifically, we propose using a sample-efficient curriculum to establish quadrupedal robot control in which the walking and turning tasks are divided into two hierarchical layers, and a robot learns them incrementally from lower to upper layers. To develop such a curriculum, two core components are designed. First the fractal design of neural networks in reservoir computing is aimed at allocating the tasks to be learned to respective modules in fractal networks. This allows mitigating the problem of catastrophic forgetting in neural networks and achieves the capability of continuous learning. The second task includes hierarchical task decomposition according to robotics knowledge for controlling legged robots. Owing to the combination of these two components, the proposed curriculum enables a robot to tune the lower layer even when the upper layer is optimized. As a result of implementing the proposed design, we confirm that a quadrupedal robot in a dynamical simulator succeeds in learning skills hierarchically according to the given curriculum, starting from moving legs and finally, walking/turning, unlike the considered conventional curriculums that are unable to achieve such results.

1. Introduction

Recently, the concept of reinforcement learning (RL) (Sutton and Barto, 2018) has been introduced to enable an agent to learn tasks that are difficult to be solved analytically based on raw high-dimensional input. RL is intended to be implemented jointly with neural networks (NN) (Levine et al., 2018; Modares et al., 2015; Luo et al., 2018; Tsurumine et al., 2019). One of the open problems in such end-to-end learning is a need for providing big data. Collecting the sufficient amount of data and learning corresponding tasks on the basis of these data is challenging for robots. To address this issue, several researchers have focused on investigating the concept of sample-efficient RL (Tsurumine et al., 2019; Schulman et al., 2015; Van Seijen et al., 2016; Kobayashi, 2019); however, the suggested approaches are still in the proof-of-concept phase. Another problem is a lack of interpretability of NN (Smilkov et al., 2017; Ross and Doshi-Velez, 2018; Huang et al., 2020). The features implicitly extracted by a NN are difficult to be reused in other tasks even if they have similar features. As a result, considerable amount of data can be wasted since the tasks are forced to be solved from scratch.

Therefore, in the present study, we focus on the question of how to reuse the knowledge obtained through the tasks already learned in the pursuit of sample efficiency. An appropriate curriculum to learn multiple tasks can be introduced to solve this problem. To design such a curriculum, two types of components are integrated: (i) continuous learning to mitigate the problem of catastrophic forgetting (Kobayashi and Sugino, 2019); (ii) hierarchical learning, which allows fully exploiting domain knowledge to establish a hierarchy. As a proof-of-concept, we propose a new curriculum for learning the control procedure of walking for a quadrupedal robot (see Fig. 1).

Specifically, as the first component, we employ a fractal reservoir computing module (Kobayashi and Sugino, 2019), which is a reservoir computing module (Jaeger and Haas, 2004; Lukoševičius and Jaeger, 2009) with a fractal structure (Rozenfeld et al., 2007) to enable continuous learning. It can be used to allocate tasks to the corresponding modules. In addition, its fractality facilitates transferring (copy) knowledge from a module (i.e., control skill of a leg) to the others. Second, to establish hierarchy, we divide the whole task into three layers according to our knowledge of robotics engineering (Heess et al., 2016; Peng et al., 2017; Nachum et al., 2018): a perception layer to define a walking direction and moving speed based on high-dimensional input; a template layer to control the walking direction and moving speed; and an actuation layer to control the joints of legs.

* Corresponding author.
E-mail address: kobayashi@is.naist.jp (T. Kobayashi).

https://doi.org/10.1016/j.engappai.2020.103869
Received 5 February 2020; Received in revised form 6 June 2020; Accepted 28 July 2020
Available online 10 August 2020

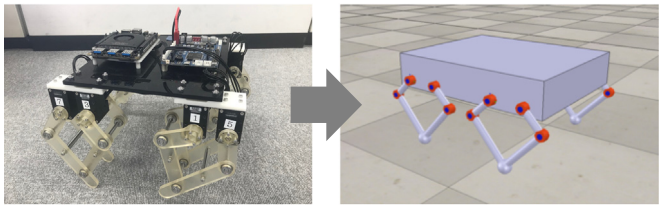


Fig. 1. Developed quadrupedal robot and its simulation model— this robot has four legs with closed linkages; each leg has two actuators to rotate in a sagittal plane; In this study, we only consider a simulation model for simplicity.

Considering these layers, we aim to design a curriculum enabling a robot (i) to incrementally learn how to control a leg with different speeds; (ii) to copy gathered knowledge from other modules for the corresponding legs; and (iii) to learn how to control walking/turning speed (see Fig. 2). In particular, the integration of the abovementioned two components allows actuation layer to be tuned even when template layer is optimized, although the concept of conventional NN implies freezing the layer already learned (Heess et al., 2016; Peng et al., 2017). Overall, using a dynamic simulator, we aim to design a curriculum to enable a robot to learn how to control the process of walking/turning, in contrast to the other three conventional curriculums including end-to-end learning that fail to resolve this problem.

2. Related work and contributions of the current study

2.1. Continual learning

The phenomenon of catastrophic forgetting is considered as one of the most critical problems associated with NN (McCloskey and Cohen, 1989; French, 1999). In this problem, the network parameters optimized during the previously learned tasks are overwritten when new tasks are learned incrementally, as all of them are updated via backpropagation (see Fig. 3).

To mitigate this problem, three ways can be considered:

1. rehearsal of previous tasks by storing their data, or sampling from the generative models for the previous tasks (Lin, 1992; Parisotto et al., 2015; Shin et al., 2017)
2. regularization implying that parameters are constrained toward ones optimal for the previous tasks (Kirkpatrick et al., 2017; Zenke et al., 2017; Kobayashi, 2018)
3. the modularization of a network architecture and the allocation of corresponding tasks to particular modules (Ellefsen et al., 2015; Velez and Clune, 2017; Kobayashi and Sugino, 2019)

Here, the rehearsal is associated with the considerable memory costs required to store the data corresponding to the previous tasks or to learn the generative models for them. Therefore, this option is excluded from consideration in pursue of implementing a memory-efficient method.

Concerning regularization, elastic weight consolidation (EWC) (Kirkpatrick et al., 2017) is the most widely used method that has been introduced in recent years. EWC implies that network parameters can be sampled from a multivariate diagonal normal distribution with the optimal location for previous tasks and its precision. In this case, the parameters with the large precision have to be fixed on the optimal location. The others can be updated toward the location that is optimal for new tasks. In this way, EWC enables NN to learn new tasks incrementally. However, the performance of EWC depends on the accuracy of inferring the location and precision.

The cause of the catastrophic forgetting problem lies in the use of backpropagation that would update all parameters that are related to the minimization of losses for new tasks. However, if backpropagation is executed selectively for corresponding modules, the catastrophic forgetting problem can be avoided. In this regard, several studies (Ellefsen et al., 2015; Velez and Clune, 2017) have proposed modifying a network architecture to implement the modular one using the evolutionary algorithm. However, such an approach requires long time for the optimization of a network architecture, and therefore, adding new tasks is associated with increasingly expensive computational costs.

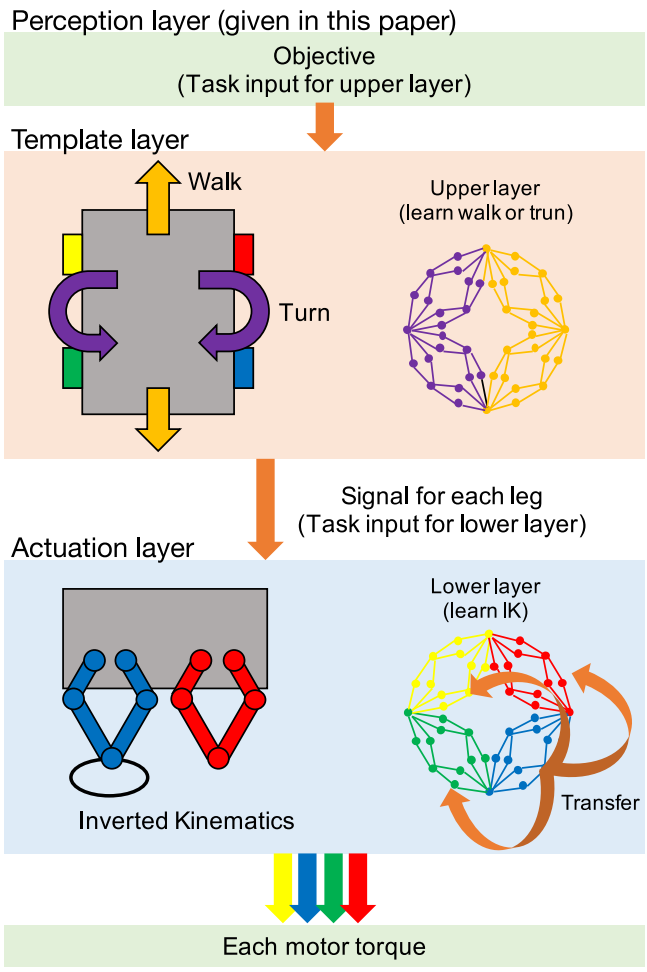


Fig. 2. Proposed network architecture for learning quadrupedal robot control— it has three layers corresponding to perception, template generation, and actuation; in the actuation layer, the skill to control a leg is first learned, and then, it is transferred to other three legs; using them as an action space in the template layer, walking/turning speed control is learned; the reference walking/turning speed can be obtained from the perception layer, although this part is out of the scope of the present study.

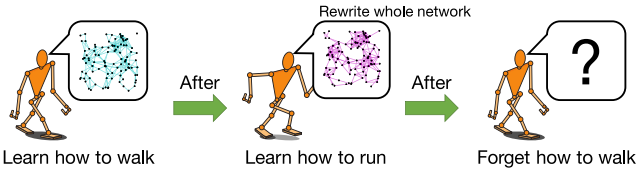


Fig. 3. Scheme of the catastrophic forgetting problem: if an agent learns tasks incrementally, the parameters optimized for the previous tasks would be overwritten by the gradients of new tasks; as a result, the agent would forget knowledge learned in the previous tasks completely.

2.2. Hierarchical learning for locomotion

The invisibility of NN leads to omitting hierarchical subtasks. In particular, locomotion including walking, running, and so on, comprises mainly three following subtasks corresponding to the perception,

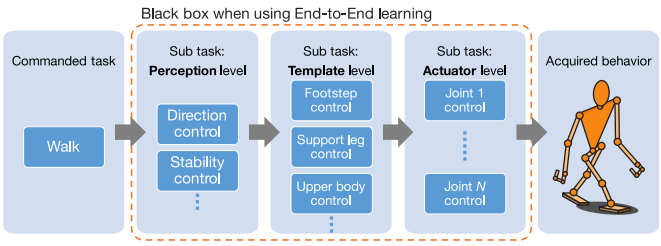


Fig. 4. Scheme of the invisibility of NN: although general tasks have hierarchical subtasks to be resolved, NN does not deal with them explicitly; therefore, NN cannot be reusable for the other tasks with similar subtasks, thereby increasing the learning cost associated with new tasks.

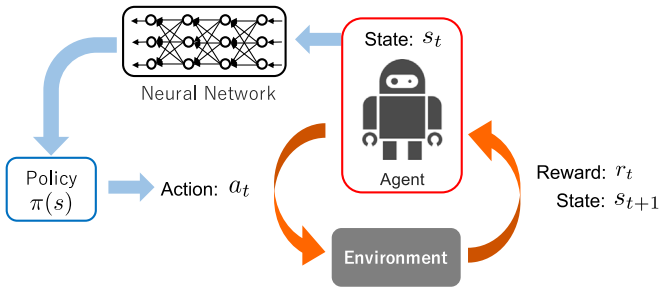


Fig. 5. Framework of reinforcement learning: an agent interacts with an environment through an action sampled from the policy over a current state; the environment returns the next state according to dynamics and the reward to be maximized.

template, and actuation layers (Johnson et al., 2015; Kobayashi et al., 2018), as shown in Fig. 4; however, end-to-end learning considers them together.

To explicitly learn such a hierarchy corresponding to given tasks, several studies have proposed implementing the hierarchical RL frameworks (Heess et al., 2016; Peng et al., 2017; Nachum et al., 2018). Such frameworks basically imply dividing a given task into two layers: the template layer that is merged to perception and the actuation layer that is aimed to reduce learning costs. It should be noted that in the present study, we consider a simplification, namely, only using the template and actuation layers. However, it is known that the simultaneous learning of multiple layers is difficult to implement due to non-stationary situations and the catastrophic forgetting problem. Accordingly, the implementation of learning curriculums can be realized in two main modes, as described in the following studies: the lower-level layer can be frozen while the higher-level layer is learned (Heess et al., 2016; Peng et al., 2017); otherwise, both layers can be concurrently learned by heuristics, such as reducing the frequency of decision making on the upper layer (Nachum et al., 2018).

2.3. Contribution of the present study

To reduce the optimization cost for network modularization, the proposed continuous learning approach employs the pre-designed fractal network (Rozenfeld et al., 2007; Kobayashi and Sugino, 2019). By including EWC (Kirkpatrick et al., 2017) as a regularization method, we allow preserving knowledge related to the tasks already learned in the corresponding modules and enable an agent to transfer knowledge to the other modules easily by exploiting its scalability.

To efficiently learn hierarchical tasks, we employ a general curriculum capable of learning the tasks from the lower to upper layer. In addition, by exploiting the ability of continual learning, the upper layer can be optimized without freezing the lower one, which allows fine-tuning the upper tasks without forgetting the lower ones.

Therefore, the main contribution of this study lies in the combination of the continual and hierarchical learning methods for the proposed curriculum. Such a combination enables the accumulation and adjustment of knowledge from simple to complicated tasks, which is more biologically plausible. In addition, although the concept itself has been already reported in the conference paper (Kobayashi and Sugino, 2019), a more detailed explanation and its application to locomotion serves an additional contribution. In the next sections, we introduce the proposed design for legged robots.

3. Preliminaries

3.1. Reinforcement learning

RL enables an agent to learn an optimal policy that can allow achieving the maximum return (namely, the sum of rewards) from an environment (Sutton and Barto, 2018). Here, the Markov decision process (MDP) is assumed as the following tuple $(\mathcal{S}, \mathcal{A}, \mathcal{R}, p_0, p_T, \gamma)$. The process of MDP can be described as follows (also see Fig. 5).

The agent first obtains the initial state $s_0 \in \mathcal{S}$ randomly: $s_0 \sim p_0(s_0)$. At the time step $t \in \mathbb{N}$, the agent selects action $a_t \in \mathcal{A}$ from the policy π over the current state $s_t \in \mathcal{S}$: $a_t \sim \pi(a_t \mid s_t)$. Here, a_t is applied to the environment, and the agent receives the next state s_{t+1} according to the transition probability model p_T: $s_{t+1} \sim p_T(s_{t+1} \mid s_t, a_t)$. At the same time, the agent obtains a reward $r_t \in \mathcal{R}$ generated by the reward function: $r_t = r(s_t, a_t, s_{t+1})$.

The agent maximizes the return $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$ with the discount factor $\gamma \in [0, 1)$ by optimizing the policy to π^*. At this point, we employ the actor–critic algorithm (Konda and Tsitsiklis, 2000) with the student-t policy (Kobayashi, 2019) and the true online temporal difference learning (Van Seijen et al., 2016). Here, the value function is additionally defined to represent the expectation value of the return from the current state: $V(s) = \mathbb{E}[R_t \mid s_t]$. In this algorithm, for the d_s-dimensional state space and d_a-dimensional action one, the value function $V(s) \in \mathbb{R}$ and three parameters for the policy: location $\mu(s) \in \mathbb{R}^{d_a}$, scale $\sigma(s) \in \mathbb{R}_+^{d_a}$, and degrees of freedom $\nu \in \mathbb{R}_+$, have to be approximated using function approximators.

3.2. Reservoir computing

As a function approximator, we employ reservoir computing (RC) (Jaeger and Haas, 2004; Lukoševičius and Jaeger, 2009) that corresponds to one of NN. RC is a model of cerebellum in biology, and as a linear regression model, is easy to learn mathematically with a small number of samples, unlike in deep learning. The simplest dynamics of RC with N neurons' internal states $x \in \mathbb{R}^N$ can be described as follows:

$$x_t = f(W^{\mathrm{rc}} x_{t-1} + W^{\mathrm{in}} s_t) \tag{1}$$

$$y_t = g(\theta^\top [x_t^\top, s_t^\top]^\top) \tag{2}$$

where $W^{\mathrm{rc}} \in \mathbb{R}^{N \times N}$ and $W^{\mathrm{in}} \in \mathbb{R}^{N \times d_s}$ are the fixed weights that are randomly given in general, whereas they are appropriately designed in this study. Here, W^{rc} is resized to guarantee that its spectral radius ρ, namely, $\max |\lambda|$ where λ are the eigenvalues of W^{rc}, is smaller than 1 for stability.

y is the output from RC obtained through a $g(\cdot)$ mapping function for example, an exponential function for σ to ensure the positive real value, and therefore, $y = [V, \mu^\top, \sigma^\top, \nu]^\top$ in this study. $\theta \in \mathbb{R}^{(N+d_s)\times(2d_a+2)}$ denotes the readout weight, which can be optimized. In the present study, θ is updated using Adam (Kingma and Ba, 2014), one of the stochastic gradient descent methods, with the learning rate $\alpha \in \mathbb{R}_+$.

In RC, the gradients with respect to θ only depend on x (and s). If the activation function $f(\cdot)$ is rarely non-zero, θ is rarely updated. Such sparse update is useful to mitigate the catastrophic forgetting problem, that is, it can be never caused when only neurons in modules for the corresponding tasks have non-zero values.

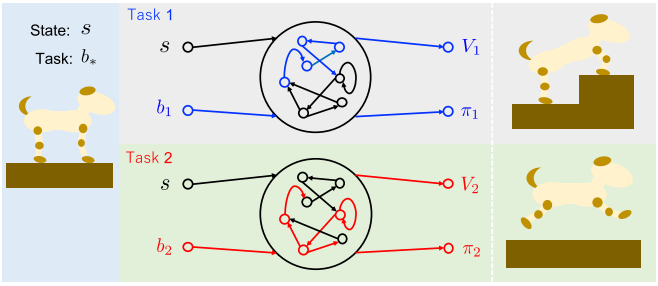


Fig. 6. Scheme of the task input into RC: task input b is additionally provided to RC as one of the inputs to specify what task the agent should execute; it can facilitate modularization of neurons according to RC dynamics.

Thereafter, we employ the following sparse activation function (Kobayashi and Sugino, 2019), $f(\cdot) := \mathrm{ls}(\cdot; p)$:

$$\mathrm{ls}(x; p) = \begin{cases} k(p)\mathrm{sgn}(x)\left\{\frac{1}{2}|x| - \frac{1}{2\pi}\sin(\pi|x|)\right\}^p & |x| \le 2 \\ k(p)\mathrm{sgn}(x) & |x| > 2 \end{cases} \tag{3}$$

where $p \in \mathbb{R}^N$ is the hyperparameter to determine the sparsity (namely, the width of the dead zone where the output is almost equal to 0). Here, $k(p)$ is defined in accordance with p to standardize the integral value within $x \in [-2, 2]$. Using this activation function, we expect achieving acceptable performance when the task-specific neurons close to the centers of modules are with large p, and the common neurons for all tasks are with small p.

3.3. Elastic weight consolidation

To mitigate the catastrophic forgetting problem, EWC (Kirkpatrick et al., 2017) can be employed in addition to the designed RC described in the previous section. EWC assumes that the parameters in NN (hereinafter, denoted as θ) can be sampled from the multivariate diagonal normal distribution $p_{\mathrm{EWC}}(\theta; \theta^*, F) = \mathcal{N}(\theta^*, F^{-1})$, where θ^* denotes the optimal values for the already learned tasks, and F is the importance of the respective parameters.

To satisfy this assumption, θ can be regularized as follows:

$$\mathcal{L}_{\mathrm{EWC}}(\theta; \theta^*, F) = -\lambda_{\mathrm{EWC}} \log p_{\mathrm{EWC}}(\theta; \theta^*, F) \tag{4}$$

$$\nabla_\theta \mathcal{L}_{\mathrm{EWC}}(\theta; \theta^*, F) = \lambda_{\mathrm{EWC}} F(\theta - \theta^*) \tag{5}$$

where λ_{EWC} is the magnitude of regularization. That is, if θ^* and F are correctly defined, the part of θ with the large F (the important parameters for the already learned tasks) can be fixed to θ^*; the others would be freely updated to learn new tasks.

It should be noted that several types of relatives have been proposed to approximate θ^* and/or F (Zenke et al., 2017) Due to the non-verification target, in this study, we employ a moving average to estimate θ^* and F for simplicity.

4. Fractal network design for continual learning

4.1. Task input into the attracting network corresponding to different patterns

As mentioned above, W^{rc} and W^{in} in RC are defined randomly. Such randomness can be sometimes useful as a universal function approximator as it generates statistically diverse neurons. In the context of continual learning, however, this network architecture can be applied to mitigate the problem of catastrophic forgetting.

Before this step, we define the task input for the network. The task that the agent should execute can be commanded as a one-hot vector or the probabilities of B tasks, $b \in \mathbb{R}^B$. In that case, b is added to RC as a bias multiplied by the bias weight $W^{\mathrm{bs}} \in \mathbb{R}^{B\times B}$. According to the fixed dynamics of RC, the activated network regions (or neurons) corresponding to the given tasks can be manipulated by providing the appropriate task input (see Fig. 6). There are two ways to perform this step: optimizing b or designing W^{bs} (and W^{rc}), and the latter is selected in this study. It should be noted that we further design W^{in} to facilitate modularization.

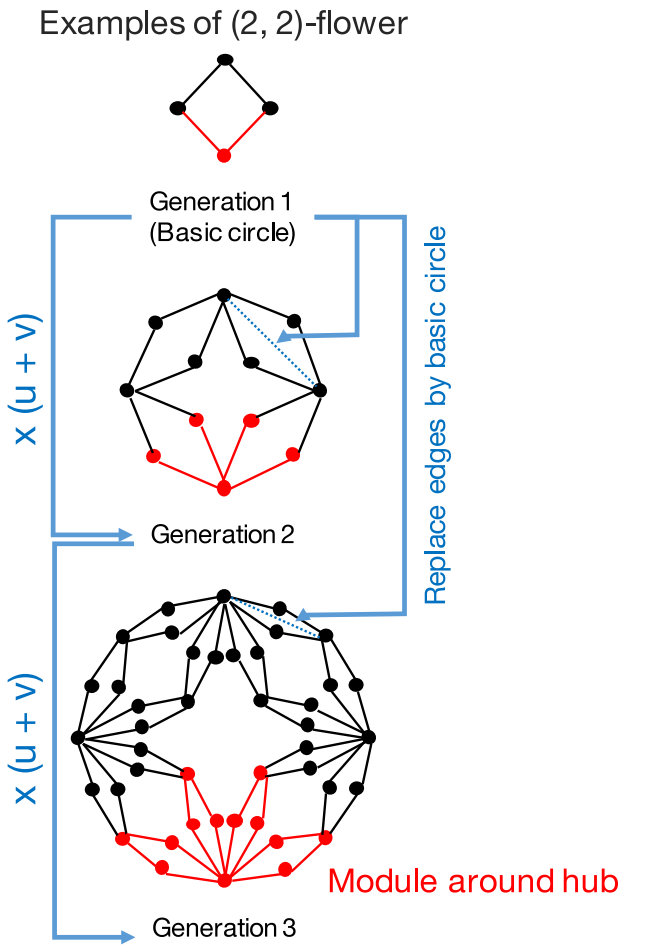


Fig. 7. Example of the generation process of (u, v)-flower (Rozenfeld et al., 2007): the $(2, 2)$-flower up to the third generation is illustrated with the module indicated in red; the number of neurons N is given to be the number of generation multiplied by $u+v$; this architecture is represented as adjacency matrix A.

4.2. Fractal network for the reservoir layer

Concerning the high modularity (and scalability) of W^{rc} that represents the dynamics of RC, we find that using a fractal network is acceptable. As one of these models, a (u, v)-flower network (Rozenfeld et al., 2007) is employed in this study. It should be noted that u and v (and nth generation) are the hyperparameters to decide its structure. As $u = 1$ has no modularity, a modular (u, v)-flower network is defined under the following condition: $v \ge u \ge 2$. It is hierarchically structured, meaning that the degrees of neurons are basically the multiplication of $u + v$, and it has $u + v$ hubs with the maximum degree $(n - 1)(u + v)$. In other words, submodules can be easily increased, as the previous main modules are stored while its generation progresses.

The generation algorithm for (u, v)-flower is described below, and its example is illustrated in Fig. 7.

1. Define a cycle C_1 composed of two completely different paths with u and v edges, respectively, as the first generation graph;

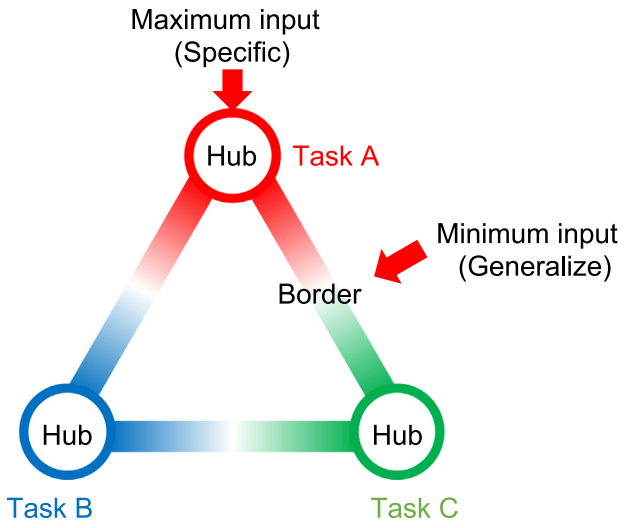


Fig. 8. Task input allocation by designing W^{bs}: the neurons close to hubs are regarded as the task-specific ones; the neurons around the boundaries between modules are considered as the common neurons for all tasks.

2. Replace all edges in the current generation with C_1 while retaining the existing nodes in the current generation;
3. Repeat nth generation.

After this procedure, the adjacency matrix A that is a binary representation of the presence or absence of connections between neurons can be obtained. That is, W^{rc} is given combined with A and random matrix R.

$$W^{\mathrm{rc}} = A \circ R \tag{6}$$
$$\text{s.t. } \rho < 1$$

where $\circ$ denotes the Hadamard product.

4.3. Design of bias weight for allocation of task input

Next, W^{bs} is designed to activate the corresponding modules by adding b. As a naive idea, when W_i^{bs} of ith neuron is given according to the distance to the hubs for respective tasks, the neurons close to the hubs are easily activated only for the respective tasks. In addition, the neurons around the boundaries between the modules represent common knowledge for all tasks (see Fig. 8).

Therefore, we design W^{bs} according to the following procedure:

1. From the entire neuron set $\mathcal{N}$, select a hub set for all tasks to be learned, $\mathcal{H} \subset \mathcal{N}$;
2. Set a hyperparameter of the kernel function to measure the distance between two inputs, $k(\cdot,\cdot)$ ($K(\cdot,\cdot)$ Gram matrix for it);
3. Set W^{bs} of $\mathcal{H}$ according to the distance between them and $\mathcal{N}$ with uniformly random plus–minus sign.

$$W^{\mathrm{bs}} = \pm K(\mathcal{H}, \mathcal{N}) \tag{7}$$

It should be noted that as the kernel function, the KCS kernel (Remaki and Cheriet, 2000) that is a modified Gaussian kernel defined to consider the upper and lower bounds of the distance between two inputs, is employed so that the bounds are on the intermediate points with adjacent hubs.

Even when the generation of the (u, v)-flower is used as an increment to add new tasks, it is expected that the distance between the adjacent hubs would not be smaller than that before the generation increment. Namely, this design does not lose scalability.

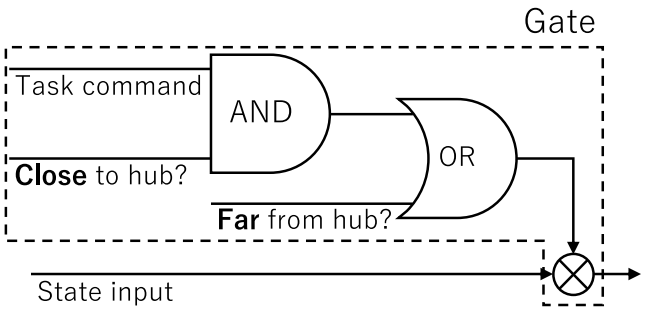


Fig. 9. Graphical image of the gate for the state input: if the neuron is close to any hubs, the gate is open only when the task command is given; if the neuron is far from any hubs, the gate is always open.

4.4. Gate for the state input

In the tasks with the large state space like locomotion control of multi-legged robots, the state input can enable any neurons to be easily activated without the sufficient task input. To facilitate modularization, we additionally suppress the dominant influence of the state input.

Thereafter, a gate matrix, $G(b; W^{\mathrm{bs}}) \in \mathbb{R}^{N\times N}$, is implemented to smoothly cut the state input to the neurons close to the deactivated hubs (the center of the modules). $G(b; W^{\mathrm{bs}})$ is defined as follows:

$$G_{ij} = \begin{cases} 0 & i \neq j \\ \mathrm{ls}(2x_i; 1) & i = j \end{cases} \tag{8}$$
$$x_i = \left| \frac{W_i^{\mathrm{bs}} b}{\max(1, \|W_i^{\mathrm{bs}}\|)\max(1, \|b\|)} \right| + \left(1 - \sqrt{\min(1, \|W_i^{\mathrm{bs}}\|)}\right)$$

where W_i^{bs} denotes the vector for ith neuron bias weight. This gate is open regardless of b if W_i^{bs} is small; otherwise, it is equal to 1 or 0 if the corresponding b is (not) given (see Fig. 9).

4.5. Redefinition of reservoir dynamics

The formulas provided in Eqs. (1) and (2) should be redefined according to the aforementioned design. Let W^{in} be given from a uniform random distribution $[-1, 1]$. In this case, a new dynamics can therefore be defined as follows:

$$x_t = \mathrm{ls}(W^{\mathrm{rc}} x_{t-1} + G(b_t; W^{\mathrm{bs}}) W^{\mathrm{in}} s_t + W^{\mathrm{bs}} b_t; p) \tag{9}$$
$$y_t = g(\theta^{\top}[x_t^{\top}, s_t^{\top}, b_t^{\top}]^{\top}) \tag{10}$$

where W^{rc}, W^{bs}, and G are designed in Eqs. (6), (7), and (8), respectively.

Here, p is defined for each neuron according to W_i^{bs} by inverting Eq. (3).

$$p_i = \frac{\ln(r_f)}{\ln\left\{\frac{\|W_i^{\mathrm{bs}}\|}{2} - \frac{1}{2\pi}\sin(\pi\|W_i^{\mathrm{bs}}\|)\right\}} \tag{11}$$

where $r_f = 10^{-12}$ is a hyperparameter to determine the sparseness of the activation function. Owing to this design, the neurons close to hubs are likely to escape the dead zone easily when the sufficient task input is provided.

We demonstrate the performance of modularization (namely, the frequency of neurons) in Fig. 10. As expected, the activated neurons clearly correspond to three tasks defined in Kobayashi and Sugino (2019).

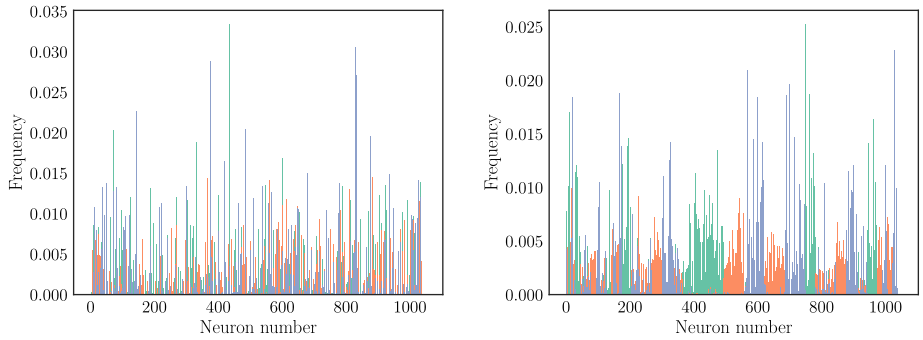


Fig. 10. Frequency of neurons when the three tasks are given: green, red, and blue lines denote the 1st, 2nd, and 3rd tasks, respectively; although (a) the conventional design fails to establish modularization, (b) the proposed design achieves performing the distinguished activation of neurons. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

5. Curriculum learning with hierarchy

5.1. Transferring knowledge from one leg to others

Concerning multi-legged robots, legs are often designed as separate modules for compatibility and decentralized control (Inagaki et al., 2006; Kenneally et al., 2016). Using such hardware modularity and scalability in the fractal RC, the knowledge about control of one leg can be easily transferred to others. This would allow improving the sample efficiency of RL.

As shown in Fig. 11, the procedure to copy the original network for one leg to others that alternates the general generation of (u, v)-flower network, can be summarized as follows:

1. Copy the original network except ith neuron and paste it by connecting on ith neuron;
2. Copy the original network except jth neuron and paste it by connecting on jth neuron;
3. Repeat the processes (1–2) for $(u + v)/2 - 2$ times;
4. Copy the original network except ith and jth neurons and paste it to close the chained network;
5. Copy the values of W^{rc}, W^{in}, and W^{bs} from the origin network to the copied one.

It should be noted that as $v \geq u \geq 2$, this transferring is valid only for robots with four or more legs. In addition, a quadrupedal robot in the considered case has to satisfy the following: $u = v = 2$.

Unlike perfectly divided networks for respective legs, the proposed network has the connections between modules, and they can communicate with each other. That can be advantageous to improve the control performance corresponding to the whole body control, as all legs interact with each other through the robot's body and ground.

5.2. Hierarchy suitable for locomotion control

To reuse the previously gained knowledge, the hierarchical learning can be designed as follows. It should be emphasized that, in multi-legged locomotion tasks, we can specify three layers (see Fig. 2): (i) the perception layer to analyze external environment (is out of scope of this study); (ii) the template layer to generate the task input; (iii) the actuation layer, which takes the input from the template layer and controls all actuators established in a robot.

As the task input into the template layer b^t, the walking direction and the speed or footstep location would be considered as parameters to select (Kobayashi et al., 2018; Perrin et al., 2011). To prioritize locomotion itself rather than locomotion in complex environments, we select the walking and turning speeds as the task input into the template layer. That is, b^t is defined as the commands corresponding to the normalized walking and turning speeds, v^w and v^t, respectively: $b^t = [v^w, v^t]^\top$.

As the task input into the actuation layer b^a (namely, the outputs of the template layer), the Cartesian-space leg poses or phase deviation of leg oscillators, such as central pattern generator (CPG) (Inagaki et al., 2006), are considered as parameters to select. It is well known that CPG has succeeded in various types of locomotion concerning many real multi-legged robots, even though it only deals with the phases of respective legs. We therefore select using the phase deviation as the task input into the actuation layer. Owing to this design, the action space of the template layer can be reduced from eight (namely, two-dimensional leg positions for respectively legs) to four dimensions. Here, to exploit modularity, the positive and negative deviations are divided into different tasks. That is, b^a is defined as $b^a = [\max(0, \omega_1), \min(0, \omega_1), \ldots, \max(0, \omega_4), \min(0, \omega_4)]^\top$ with ω_i the phase deviation of ith leg.

For the purpose of acquiring the reference phases from the template layer, the actuation layer aims to control the actuators corresponding to all legs. It should be noted that according to the previous section, this control can be performed for each leg. In this way, although the network in the actuation layer receives all states about all legs and all reference phases, they are allocated to respective modules by implementing W^{in} and W^{bs}. Such allocation can simplify the learning process of leg control.

5.3. Curriculum on a hierarchical system

The proposed system has the three capabilities of reusing the knowledge already learned: modularity for continuous learning; transferring knowledge about leg control; the hierarchy suitable for locomotion. To fully exploit these capabilities, we need to establish a curriculum for learning locomotion.

First, a quadrupedal robot needs to learn how to control a leg on the ground as a part of the actuation layer. The normalized phase deviation ω is defined $\{0, 0.5, 1.0, -0.5, -1.0\}$ in order. At this point, other legs are fixed at the initial states to reduce the randomness of the environment. Owing to continuous learning, we can avoid random commands for ω, which would require considerable time and would be inefficient. A robot can focus on learning the skills for constant ordered commands, and interpolation is naturally expected by preserving all skills for the given commands appropriately. After learning this step, the knowledge about control of a leg is transferred to other legs to complete the actuation layer.

Thereafter, the template layer is ready to be trained through the constant walking/turning speed commands. Although the actuation layer is not perfectly optimized (in particular, considering the interpolation of ω and transferred policies), the proposed system allows the robot to learn it at the same time as the template layer. Such fine-tuning of the

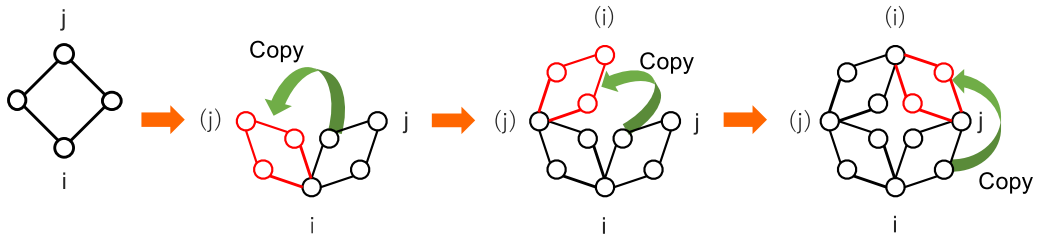


Fig. 11. Procedure of transferring the network for a leg to the others: the original network is copied, and replicas are connected one by one; as the network (u, v)-flower has fractality, its generation can be replaced through such copy processes.

actuation layer combined with the proposed network design in which the modules for all legs are connected with each other is intended to facilitate the coordination of legs.

6. Simulations

6.1. Simulation model

The developed quadrupedal robot is simulated in V-REP (Rohmer et al., 2013), as shown in Fig. 1. This robot has four modular leg structures with the five-section parallel link mechanism. They can be launched by two actuators attached on their roots, and their tips move on a sagittal plane.

The robot has an IMU sensor on its torso, which can be used to measure the walking speed on the horizontal plane and yaw-axis rotation angle and its speed. In total, the sum of the four-dimensional state space for the torso and the four-dimensional state space for each leg (the angles and angular velocities of two actuators), namely, 20-dimensional state space, is obtained. It should be noted that, in the actuation layer for each leg control, the reference sagittal position of a tip of each leg is additionally defined corresponding to the state space in the actuation layer; that is, the controller of each leg has the six-dimensional state space.

The robot itself has the eight-dimensional action space (namely, the reference angular velocities of all actuators). In the template layer, however, it has the four-dimensional action space due to assuming leg cyclic motions.

The reward function for the template layer is defined as follows:

$$
\begin{aligned}
r_1 &= \exp\left(-c_1^t \left| \frac{v}{v_{\max}} - b_1^t \right| \right) - 0.5 \\
r_2 &= \exp\left(-c_2^t \left| \frac{\omega}{\omega_{\max}} - b_2^t \right| \right) - 0.5 \\
r^t &= r_1 + r_2
\end{aligned} \tag{12}
$$

where v and ω denote the velocity and angular velocity of the torso, respectively, and are normalized by their maximums, $v_{\max}$ and $\omega_{\max}$. Here, $c_{1,2}^t$ are the hyperparameters used to tune the decaying rates of the respective reward terms ($c_1^t = 2$ and $c_2^t = 1.5$ in the present study). By maximizing this reward, the robot can realize locomotion with the desired walking and turning speeds, $b_{1,2}^t$.

In addition, the reward function in the actuation layer is defined as follows:

$$
\begin{aligned}
e &= \sqrt{\left(\frac{x^{\mathrm{obs}} - x^{\mathrm{ref}}}{d_x}\right)^2 + \left(\frac{z^{\mathrm{obs}} - z^{\mathrm{ref}}}{d_z}\right)^2} \\
r^a &= 2\exp\left(-c^a \frac{e}{e_{\max}}\right) - 1
\end{aligned} \tag{13}
$$

where x and z denote the positions of the tip of a leg. Superscripts obs and ref are consistent with the observation and reference values, respectively. The trajectory error e is calculated through normalization by respective radiuses, d_x and d_z. Similarly as in Eq. (12), we define the reward function as exponential decay according to the error (normalized by its maximum $e_{\max}$) with the decaying rate $c^a = 2.5$. Although this reward is not the main purpose of locomotion task, it can be helpful to gain the skills to control legs and reuse them.

6.2. Conditions

Here, Table 1 summarizes the parameters of the implementation. It should be noted that $N^t = 3512$ and $N^a = 684 \times 4$ are given by the fourth generation of the (4, 4)-flower and the fifth generation of the (2, 2)-flower for four legs (namely, one transferring), respectively. We have confirmed that these parameters can be used to learn the tasks stably in other primitive tasks (Kobayashi and Sugino, 2019), and therefore, we decide to reuse them.

6.3. Continual learning of the actuation layer

First, we demonstrate the performance of continuous learning in the actuation layer. The proposed method with the (u, v)-flower is compared with the conventional approach based on the standard random network (Jaeger and Haas, 2004; Lukoševičius and Jaeger, 2009). As mentioned above, the curriculum to learn the actuation layer (to control a leg) is defined so that the reference signal to the leg is set as $b^a = \{0, 0.5, 1.0, -0.5, -1.0\}$ in order.

In Figs. 12(a) and (b), the learning curves gained from the conventional and proposed methods are represented. We find that the conventional method fails to learn the ± 0.5 phase deviations. However, they can be learned successfully by the proposed method in all cases. This can be explained by the adverse effect of EWC, which is likely to preserve important parameters excessively; that is, the conventional method can use all parameters corresponding to the previous tasks to some extent and have no room for learning new tasks.

After formulating the curriculum, we test all conditions again without learning. The test results indicate that the average reward of the proposed method ($\sim$0.35) outperform that of the conventional approach ($\sim$ 0.15). Even in the case of $b^a = 0.75$ that is not learned in the curriculum, the proposed method gains the $\sim$0.40 average reward.

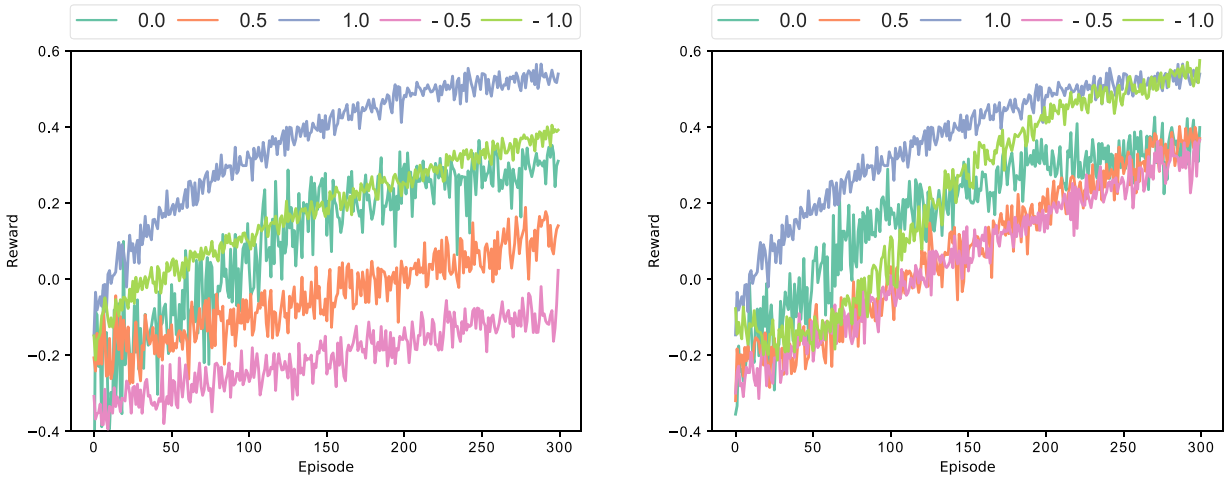
6.4. Learning of the template layer using the proposed curriculum

We consider the four kinds of simulations as follows:

1. end-to-end learning from scratch;
2. simultaneous learning of the template and actuation layers from scratch similarly as in Nachum et al. (2018);
3. learning of the template layer with the actuation layer frozen after learning similarly as in Heess et al. (2016) and Peng et al. (2017);
4. the proposed curriculum: after learning the actuation layer, both template and actuation layers are optimized for the given tasks.

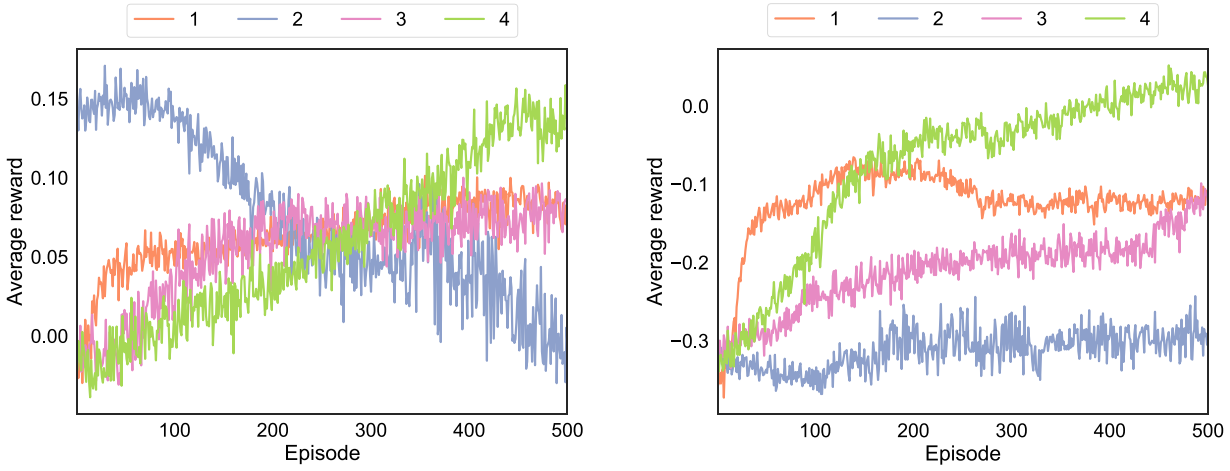
It should be noted that, in the end-to-end learning, an agent does not have multiple layers and outputs the actions corresponding to the angular velocities of all actuators over all states that are observed by the agent. The first task is with $b^t = [1, 0]^\top$, walking forward. The other task is with $b^t = [0, 1]^\top$, turning left.

The learning curves from all curriculums are illustrated in Figs. 13(a) and (b). The first and third curriculums finally achieve the similar performance, even though their policies cannot enable walking/turning appropriately. The second curriculum fails to learn



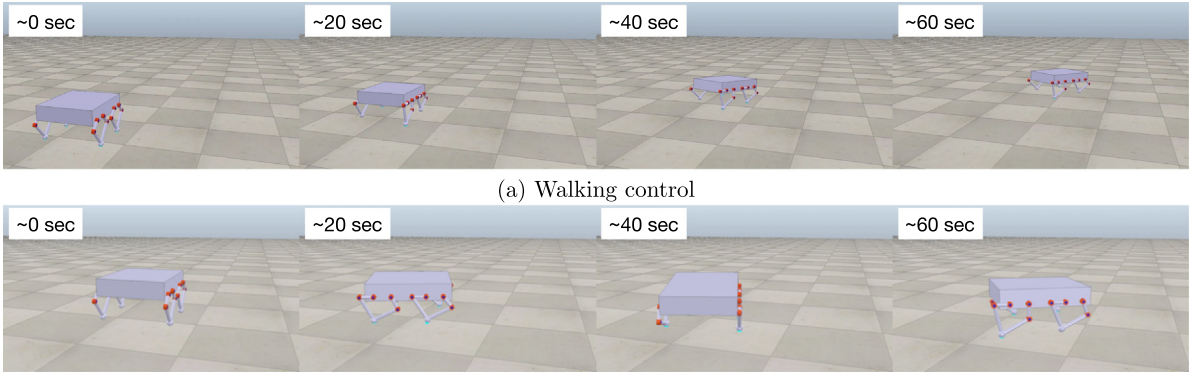
(a) Conventional method in the random network (b) Proposed method with the (u, v)-flower

Fig. 12. Learning curves of the actuation layer with two conditions: in both (a) and (b), the first task, $b^a = 0$, can achieve the same performance; (a) when the modular network is not used, the performance in other tasks that relearned incrementally after the first task, in particular $b^a = \pm 0.5$, deteriorates; (b) in contrast, when the proposed design is used, all tasks achieve the acceptable performance.



(a) Walking control (b) Turning control

Fig. 13. Learning curves of the template layer with four curriculums: the four legends correspond to the list in the main body; in both walking and turning tasks, the proposed curriculum outperforms the alternative ones, as they fail to learn both tasks due to the lack of abilities to simplify the tasks and to reuse and modify the previously learned skills.



(a) Walking control

(b) Turning control

Fig. 14. Snapshots of simulations after learning: in (a) walking control, the robot can walk forward, although it turns right gradually due to the insufficient rotation feedback; in (b) turning control, the robot can turn left on the spot with the almost constant speed.

Table 1
Parameters of the implemented algorithm.

Symbol	Meaning	Value
$N^{t,a}$	Number of neurons in the template or actuation layers	3512 or 684 × 4
$\rho^{t,a}$	Spectral radius in the template or actuation layers	0.2 or 0.08
$\alpha^{t,a}$	Learning rate in the template or actuation layers	3e-3/N^t or 5e-3/N^a
ϵ	Numerical stabilization term in Adam	1e−2
γ	Discount factor	0.99
λ_{EWC}	Scale for EWC regularization	1e-10

both tasks due to interference between two layers. It should be noted that the reason for the fact that its learning curve at the early stage of walking control task is high is that it mostly stops by chance, and the reward for turning control is high.

In contrast, the proposed curriculum succeeds in learning both walking and turning tasks. The learning curves are stably increased and exceed the results of the other considered curriculums. Indeed, the motions after learning are as expected (see Fig. 14).

7. Conclusion

The present paper was focused on incorporating the knowledge gathered from robotics engineering and tasks already learned. From this perspective, we contributed to designing a curriculum in which a robot learned subtasks from the lower to upper layer incrementally without freezing the networks already optimized. The proposed curriculum comprised two components. The first one was the fractal RC providing modularity to allocate tasks to modules for continuous learning and establishing fractality to copy the module with primitive knowledge to similar modules. The second component was designed heuristically to provide hierarchy for locomotion based on robotics engineering knowledge. In dynamical simulations, during the learning from the lower layer, the proposed fractal network design succeeded in mitigating the problem of catastrophic forgetting and establishing control of the rotation speed of a leg incrementally. As a result, the proposed curriculum allowed improving the performance in terms of the speed control of the walking and turning tasks compared with the three considered conventional curriculums.

As future study, we plan to introduce a perception layer in addition to the other layers. It would allow resolving goal-oriented tasks, and therefore, a curriculum gradually increasing the complexity of goals (Narvekar et al., 2017) is deemed to be effective. After completing the development of a hierarchical learning system, we will focus on conducting experiments on real robots.

CRediT authorship contribution statement

Taisuke Kobayashi: Conceptualization, Methodology, Writing - review & editing, Supervision, Project administration, Funding acquisition. **Toshiki Sugino:** Methodology, Software, Validation, Formal analysis, Investigation, Writing - original draft, Visualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was supported by JSPS KAKENHI, Japan, Grant-in-Aid for Young Scientists (B), Grant Number 17K12759.

References

Ellefsen, K.O., Mouret, J.-B., Clune, J., 2015. Neural modularity helps organisms evolve to learn new skills without forgetting old skills. PLoS Comput. Biol. 11 (4).
French, R.M., 1999. Catastrophic forgetting in connectionist networks. Trends Cogn. Sci. 3 (4), 128–135.
Heess, N., Wayne, G., Tassa, Y., Lillicrap, T., Riedmiller, M., Silver, D., 2016. Learning and transfer of modulated locomotor controllers. arXiv preprint arXiv:1610.05182.
Huang, J., Angelov, P.P., Yin, C., 2020. Interpretable policies for reinforcement learning by empirical fuzzy sets. Eng. Appl. Artif. Intell. 91, 103559.
Inagaki, S., Yuasa, H., Suzuki, T., Arai, T., 2006. Wave CPG model for autonomous decentralized multi-legged robot: Gait generation and walking speed control. Robot. Auton. Syst. 54 (2), 118–126.
Jaeger, H., Haas, H., 2004. Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. Science 304 (5667), 78–80.
Johnson, M., Shrewsbury, B., Bertrand, S., Wu, T., Duran, D., Floyd, M., Abeles, P., Stephen, D., Mertins, N., Lesman, A., et al., 2015. Team IHMC's lessons learned from the DARPA robotics challenge trials. J. Field Robotics 32 (2), 192–208.
Kenneally, G., De, A., Koditschek, D.E., 2016. Design principles for a family of direct-drive legged robots. IEEE Robot. Autom. Lett. 1 (2), 900–907.
Kingma, D.P., Ba, J., 2014. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.
Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A.A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., et al., 2017. Overcoming catastrophic forgetting in neural networks. Proc. Natl. Acad. Sci. 114 (13), 3521–3526.
Kobayashi, T., 2018. Check regularization: combining modularity and elasticity for memory consolidation. In: International Conference on Artificial Neural Networks. Springer, pp. 315–325.
Kobayashi, T., 2019. Student-t policy in reinforcement learning to acquire global optimum of robot control. Appl. Intell. 49 (12), 4335–4347.
Kobayashi, T., Sekiyama, K., Hasegawa, Y., Aoyama, T., Fukuda, T., 2018. Unified bipedal gait for autonomous transition between walking and running in pursuit of energy minimization. Robot. Auton. Syst. 103, 27–41.
Kobayashi, T., Sugino, T., 2019. Continual learning exploiting structure of fractal reservoir computing. In: International Conference on Artificial Neural Networks. Springer, pp. 35–47.
Konda, V.R., Tsitsiklis, J.N., 2000. Actor-critic algorithms. In: Advances in Neural Information Processing Systems. pp. 1008–1014.
Levine, S., Pastor, P., Krizhevsky, A., Ibarz, J., Quillen, D., 2018. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. Int. J. Robot. Res. 37 (4–5), 421–436.
Lin, L.-J., 1992. Self-improving reactive agents based on reinforcement learning, planning and teaching. Mach. Learn. 8 (3–4), 293–321.
Lukoševičius, M., Jaeger, H., 2009. Reservoir computing approaches to recurrent neural network training. Comp. Sci. Rev. 3 (3), 127–149.
Luo, J., Edmunds, R., Rice, F., Agogino, A.M., 2018. Tensegrity robot locomotion under limited sensory inputs via deep reinforcement learning. In: IEEE International Conference on Robotics and Automation. IEEE, pp. 6260–6267.
McCloskey, M., Cohen, N.J., 1989. Catastrophic interference in connectionist networks: The sequential learning problem. In: Psychology of Learning and Motivation, Vol. 24. Elsevier, pp. 109–165.
Modares, H., Ranatunga, I., Lewis, F.L., Popa, D.O., 2015. Optimized assistive human–robot interaction using reinforcement learning. IEEE Trans. Cybern. 46 (3), 655–667.
Nachum, O., Gu, S.S., Lee, H., Levine, S., 2018. Data-efficient hierarchical reinforcement learning. In: Advances in Neural Information Processing Systems. pp. 3303–3313.
Narvekar, S., Sinapov, J., Stone, P., 2017. Autonomous task sequencing for customized curriculum design in reinforcement learning. In: International Joint Conference on Artificial Intelligence. pp. 2536–2542.
Parisotto, E., Ba, J.L., Salakhutdinov, R., 2015. Actor-mimic: Deep multitask and transfer reinforcement learning. arXiv preprint arXiv:1511.06342.
Peng, X.B., Berseth, G., Yin, K., Van De Panne, M., 2017. Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning. ACM Trans. Graph. 36 (4), 1–13.
Perrin, N., Stasse, O., Baudouin, L., Lamiraux, F., Yoshida, E., 2011. Fast humanoid robot collision-free footstep planning using swept volume approximations. IEEE Trans. Robot. 28 (2), 427–439.

Remaki, L., Cheriet, M., 2000. KCS-new kernel family with compact support in scale space: formulation and impact. IEEE Trans. Image Process. 9 (6), 970–981.

Rohmer, E., Singh, S.P., Freese, M., 2013. V-REP: A versatile and scalable robot simulation framework. In: 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems. IEEE, pp. 1321–1326.

Ross, A.S., Doshi-Velez, F., 2018. Improving the adversarial robustness and interpretability of deep neural networks by regularizing their input gradients. In: AAAI Conference on Artificial Intelligence.

Rozenfeld, H.D., Havlin, S., Ben-Avraham, D., 2007. Fractal and transfractal recursive scale-free nets. New J. Phys. 9 (6), 175.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., Moritz, P., 2015. Trust region policy optimization. In: International Conference on Machine Learning. pp. 1889–1897.

Shin, H., Lee, J.K., Kim, J., Kim, J., 2017. Continual learning with deep generative replay. In: Advances in Neural Information Processing Systems. pp. 2990–2999.

Smilkov, D., Thorat, N., Kim, B., Viégas, F., Wattenberg, M., 2017. Smoothgrad: removing noise by adding noise. arXiv preprint arXiv:1706.03825.

Sutton, R.S., Barto, A.G., 2018. Reinforcement Learning: An Introduction. MIT Press.

Tsurumine, Y., Cui, Y., Uchibe, E., Matsubara, T., 2019. Deep reinforcement learning with smooth policy update: Application to robotic cloth manipulation. Robot. Auton. Syst. 112, 72–83.

Van Seijen, H., Mahmood, A.R., Pilarski, P.M., Machado, M.C., Sutton, R.S., 2016. True online temporal-difference learning. J. Mach. Learn. Res. 17 (1), 5057–5096.

Velez, R., Clune, J., 2017. Diffusion-based neuromodulation can eliminate catastrophic forgetting in simple neural networks. PLoS One 12 (11).

Zenke, F., Poole, B., Ganguli, S., 2017. Continual learning through synaptic intelligence. In: International Conference on Machine Learning. pp. 3987–3995.