International Journal of Scientific Research in Computer Science and Engineering **Research Paper**
Vol.10, Issue.3, pp.13-18, June (2022) **E-ISSN:** 2320-7639

Evolutionary Reinforcement Learning of Neural Network Controller for Pendulum Task by Evolution Strategy

Hidehiko Okada

Faculty of Information Science and Engineering, Kyoto Sangyo University, Kyoto, Japan

Author's Mail Id: hidehiko@cc.kyoto-su.ac.jp

Available online at: www.isroset.org

Received: 27/Apr/2022, Accepted: 29/May/2022, Online: 30/Jun/2022

***Abstract*—** Reinforcement learning of neural networks requires gradient-free algorithms because labeled training data are not available. Evolutionary algorithms are applicable to the reinforcement learning because the algorithms do not rely on gradients. To successfully train neural networks by evolutionary algorithms, we need to carefully choose appropriate algorithms because many algorithm variations are available. The author experimentally evaluates Evolution Strategy, an instance of evolutionary algorithms, for the reinforcement learning of neural networks. A pendulum control task is adopted in this work. Experimental results revealed that ES could successfully train an MLP so that the trained MLP could make and keep the pendulum upright quickly, if the MLP was equipped with sufficient hidden units. For the task adopted in this work, 16 units are the best among 8, 16 and 32 units in terms of the task performance and the computational efficiency. Besides, the results revealed that exploration contributes more for ES to search for better solutions than exploitation.

***Keywords*—**Evolutionary algorithm; Evolution strategy; Neural network; Neuroevolution; Reinforcement learning.

I. INTRODUCTION

Neural networks can be trained by gradient based methods for supervised learning tasks where labeled training data are available: errors between neural network outputs and their target values can be observed and the errors can be backpropagated through the neural networks to modify node connection weights and node biases. On the contrary, neural networks require gradient-free training algorithms for reinforcement learning tasks, because labeled training data are not available. Evolutionary algorithms [1-5] are applicable to the reinforcement learning of neural networks because the algorithms do not rely on gradients. Q learning [6-8] is also a representative reinforcement learning method. Q learning needs to obtain reward $r(t)$ for action $a(t)$ at state $s(t)$ to determine the next action $a(t+1)$ where t is the time step. Evolutionary algorithms do not need $r(t)$ at every step but an evaluation *after* an episode is finished. Thus, evolutionary algorithms release us from designing appropriate reward for every pair of state and action. Evolution Strategy [9,10], Genetic Algorithm [11-14], Deferential Evolution [15-17] are representative evolutionary algorithms. To successfully train neural networks by evolutionary algorithms, we need to carefully i) choose appropriate algorithms because many algorithm variations are available and ii) design its hyperparameters because the design substantially affects performance. The author experimentally evaluates Evolution Strategy for the reinforcement learning of neural networks. A pendulum control task is adopted in this work.

II. PENDULUM CONTROL TASK

As a task that requires reinforcement learning to solve, this work employs "Pendulum" control task[1,2] provided at OpenAI Gym[3]. The goal is "to swing the pendulum up so it stays upright".[1] Fig. 1 shows a screenshot of the system. The round arrow shows the direction and the strength of the torque by the controller.

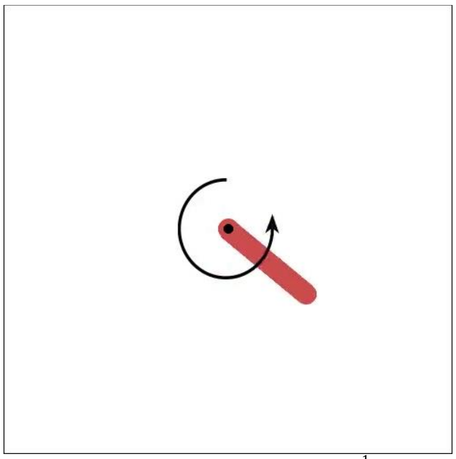


Figure 1. Pendulum System.[1]

The author changed the system so that the control task starts with the pendulum at the opposite position to the goal, i.e., the task starts with the state shown in Fig. 2(a) and the goal is to make and keep the pendulum as shown in Fig. 2(b). In addition, the author changed the system so that the control task starts with the pendulum with 0 angular velocity.

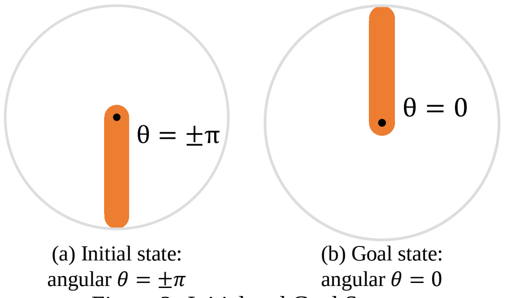

θ = ±π
θ = 0
(a) Initial state: angular $\theta = \pm\pi$
(b) Goal state: angular $\theta = 0$

Figure 2. Initial and Goal States.

One episode consists of 200 time steps. In each step, the controller observes the current state and then determines the action. An observation obtains $\cos(\theta)$, $\sin(\theta)$ and the angular velocity where $-1.0 \le \cos(\theta) \le 1.0$, $-1.0 \le \sin(\theta) \le 1.0$ and $-8.0 \le \text{angular velocity} \le 8.0$. The action is the torque to the pendulum where $-2.0 \le \text{torque} \le 2.0$. Note that the pendulum never reaches from the initial position to the goal position with the fixed torque 2.0 (or -2.0): the pendulum needs to be swinged so that the controller gets assisted by the gravity to let the angular velocity enough to climb over.
In this work, the author defines the fitness of a neural network controller as:

$$\text{Fitness} = \frac{1}{200}\sum_{t=1}^{200}(1 - \text{Error}(t)), \qquad (1)$$

$$\text{Error}(t) = \frac{|\theta(t)|}{\pi}. \qquad (2)$$

$\theta(t)$ denotes the angular at time step t. In the initial state, $\text{Error}(t) = |\pm\pi|/\pi = 1$ so that $1 - \text{Error}(t) = 0$. In the goal state, $\text{Error}(t) = 0/\pi = 0$ so that $1 - \text{Error}(t) = 1$. The fitness score is larger as $\text{Error}(t)$ is smaller for more time steps. Thus, an MLP controller fits better as it can make the pendulum upright more quickly and keeps the state for longer time steps.

III. NEURAL NETWORKS

In this work, the author adopts a three-layered feedforward neural network (a multilayer perceptron, MLP [18,19]) as the controller. Fig. 3 shows the topology of the MLP. Equations (3)-(7) show the feedforward calculations.

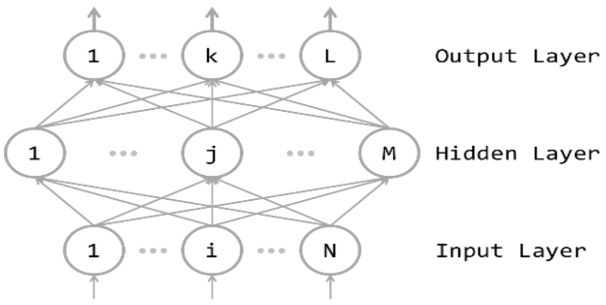


Figure 3. Topology of Neural Network in this Work.

Input layer:

$$out_i^{(1)} = x_i, i = 1,2,\ldots,N \qquad (3)$$

Hidden layer:

$$in_j^{(2)} = \theta_j^{(2)} + \sum_i w_{i,j}^{(2)}\, out_i^{(1)}, j = 1,2,\ldots,M \qquad (4)$$

$$out_j^{(2)} = h(in_j^{(2)}), j = 1,2,\ldots,M \qquad (5)$$

Output layer:

$$in_k^{(3)} = \theta_k^{(3)} + \sum_j w_{j,k}^{(3)}\, out_j^{(2)}, k = 1,2,\ldots,L \qquad (6)$$

$$out_k^{(3)} = h(in_k^{(3)}), k = 1,2,\ldots,L \qquad (7)$$

The symbols in (3)-(7) denote as follows:

x_i	Input value to i-th input unit.
$out_i^{(1)}$	Output value from i-th input unit.
$in_j^{(2)}$	Input value to j-th hidden unit.
$w_{i,j}^{(2)}$	Weight value of connection from i-th input unit to j-th hidden unit.
$\theta_j^{(2)}$	Bias value of j-th hidden unit.
$out_j^{(2)}$	Output value from j-th hidden unit.
$in_k^{(3)}$	Input value to k-th output unit.
$w_{j,k}^{(3)}$	Weight value of connection from j-th hidden unit to k-th output unit.
$\theta_k^{(3)}$	Bias value of k-th output unit.
$out_k^{(3)}$	Output value from k-th output unit.

$h()$ is a unit activation function, where the hyperbolic tangent (tanh) is adopted in this work. Fig. 4 shows the shape of tanh function.

$$h(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \qquad (8)$$

$$-1.0 < h(x) < 1.0 \qquad (9)$$

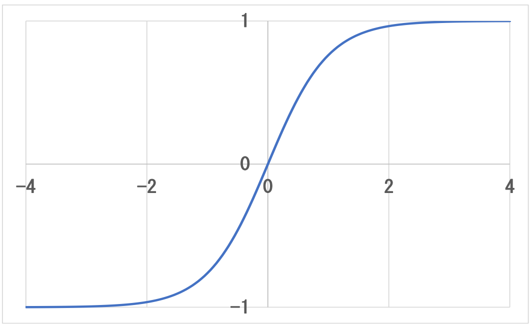


Figure 4. Tanh Function.

The MLP works as a policy function in this work: $action(t) = F(observation(t))$. The number of units in the input layer is three, where the value of $\cos(\theta)$, $\sin(\theta)$ and the angular velocity described in Section II are input to the three units respectively. The value of angular velocity is divided by 8.0 so that the input value becomes within the range $[-1.0, 1.0]$. The number of units in the output layer is one, and the output value from the unit is applied as the torque to the pendulum. The output value is multiplied by 2.0 so that the torque becomes within the range $[-2.0, 2.0]$.

IV. Training Of Neural Networks By Evolution Strategy

A three-layered perceptron with the topology shown in Fig. 3 includes $M+L$ units and $NM+ML$ connections. Thus, the total number of parameters in the perceptron is $M+L+NM+ML$. Let D denote the number $M+L+NM+ML$. Training of the perceptron in Fig. 3 is equivalent to optimization of the D-dimensional real vector. Let $x=(x_1,x_2,\dots,x_D)$ denote the D-dimensional vector, where each x_i corresponds to one of the D parameters in the perceptron. The feedforward calculation in (3)-(7) can be processed by applying each value in x to its corresponding connection weight or unit bias.

Training of neural networks by evolutionary algorithms is called neuroevolution [20,21]. Neuroevolution has been applied to games [22-25], e.g., Togelius et al. [25] applied neuroevolution to simulated car racing. In this work, the D-dimensional vector x is optimized by ES. ES processes x as chromosome and applies evolutionary operators to x. The fitness of x is measured by (1).

Fig. 5 shows the process of ES. In Step 1, vectors $y^1,y^2,\dots,y^C$ are randomly initialized within a preset range, $[min,max]^D$, where C denotes the number of offsprings. A larger value of C promotes explorative search more. In this work, min and max are set as -10.0 and 10.0 respectively. In Step 2, values in each vector y^c ($c=1,2,\dots,C$) are applied to the MLP and the MLP controls the pendulum for a single episode with 200 time steps. The fitness of y^c is then evaluated with the result of the episode. Let $f(y^c)$ denote the fitness. In Step 3, the loop of evolutionary training is finished if a preset condition is satisfied. A simple example of the condition is the limit number of fitness evaluations. In Step 4, among the P vectors in the current parent population ($z^1,z^2,\dots,z^P$) and the C vectors in the current offspring population ($y^1,y^2,\dots,y^C$), vectors with the top P fitness scores survive as the parents in the next reproduction and the remaining vectors are deleted. P denotes the number of parents. A smaller value of P promotes exploitive search more. Note that, for the first time of Step 4, the parent population is empty so that vectors with the top P fitness scores survive among the C vectors in the current offspring population ($y^1,y^2,\dots,y^C$). In Step 5, new C offspring vectors are produced by applying the reproduction operator to the parent vectors $z^1,z^2,\dots,z^P$ which are selected in the last Step 4. The new offspring vectors form the new offspring population $y^1,y^2,\dots,y^C$. Fig. 6 denotes the process of reproduction.

```
Step 1. Initialization
Step 2. Fitness Evaluation
Step 3. Conditional Termination
Step 4. Selection
Step 5. Reproduction
Step 6. Goto Step 2
```

Figure 5. Process of Evolution Strategy.

Step 5-1. Let $c=1$.

Step 5-2. A vector is randomly sampled from the parent population $z^1,z^2,\dots,z^P$. Let z^p denote the sampled vector.

Step 5-3. A copy of z^p is created as y^c. y^c is a D-dimensional vector, i.e., $y^c=(y_1^c,y_2^c,\dots,y_D^c)$.

Step 5-4. Each of $y_1^c,y_2^c,\dots,y_D^c$ is perturbed by (10)-(12) where s is a hyperparameter called step size and $rand$ is a uniform random number sampled from the interval $[-1.0,1.0]$. A greater value of s promotes explorative search more.

Step 5-5. If $c<C$ then $c\leftarrow c+1$ and goto Step 5-2, else finish the reproduction.

Figure 6. Process of Reproduction in Evolution Strategy.

$$y_d^c \leftarrow y_d^c + s * rand \tag{10}$$

$$if\ y_d^c < min\ then\ y_d^c \leftarrow min \tag{11}$$

$$if\ \max < y_d^c\ then\ y_d^c \leftarrow max \tag{12}$$

V. Experiment

The neural network adopted as a pendulum controller in this word is a multilayer perceptron with a single hidden layer. Every unit is fully connected to units in the next layer. The ability of MLPs in modeling nonlinear functions depends on the number of hidden units.

Evolutionary optimization of an MLP with a smaller number of units is easier because the genotype length (the number of variables to be optimized) is smaller. However, an MLP with a smaller number of units may not be able to successfully control the pendulum because the MLP has insufficient modeling ability. On the contrary, an MLP with a larger number of units is likely to successfully control the pendulum, but evolutionary optimization of the larger MLP becomes more difficult because the genotype length is larger. Besides, an MLP with a larger number of units requires more memory to implement on a computer. This tradeoff must be managed by designing appropriate number of hidden units for the task. In this work, the author investigates three variations: 8, 16, and 32 hidden units.

The hyperparameters of ES are empirically set as shown in Table 1 based on results of preparatory experiments. The number of generations is 500 (or 100) if population size is 100 (or 500) so that the number of fitness evaluations is consistently 50,000 (= number of generations × population size).

An MLP with 8, 16 or 32 hidden units is trained 11 times independently. Table 2 shows the best/median/worst fitness scores of the trained MLPs among the 11 runs. For example, the best MLP with 8 hidden units achieved (a)0.829 and (b)0.833 while the worst MLP with the same number of hidden units achieved (a)0.520 and (b)0.583.

Table 2(a) reveals that the median fitness score with 8 units is substantially smaller than those with 16 and 32 units, which indicates 8 hidden units are not sufficient for this task. Besides, Table 2(a)(b) reveal that the fitness scores with 32 units are close to those with 16 units, which indicates 16 units are sufficient to the task and units more than 16 are unnecessary because they do not contribute well to improve the fitness scores.

Table 1. ES Hyperparameters

	(a)	(b)
Population size	10	50
Generations	500	100
Fitness evaluations	50000	50000
Number of parents	5	5
Step size	1	1

Table 2. Best/Median/Worst Fitness Scores among 11 Runs

(a) Population size = 10, 500 Generations.

Units	Best	Median	Worst
8	0.829	0.612	0.520
16	0.833	0.823	0.579
32	0.832	0.831	0.613

(b) Population size = 50, 100 Generations.

Units	Best	Median	Worst
8	0.833	0.825	0.583
16	0.833	0.832	0.612
32	0.833	0.831	0.586

Wilcoxson rank sum tests are applied to test whether the fitness scores with 8 (16 or 32) hidden units are significantly better than those with other number of units. Table 3 shows the test result. Table 3 reveals that (1) the fitness scores with 16 units and 32 units are significantly better than those with 8 units, and (2) the fitness scores with 32 units are not significantly better than those with 16 units. The test result confirms that 16 units are sufficient.

Table 3. Wilcoxson Rank Sum Tests for Number of Hidden Units

(a) Population size = 10, 500 Generations.

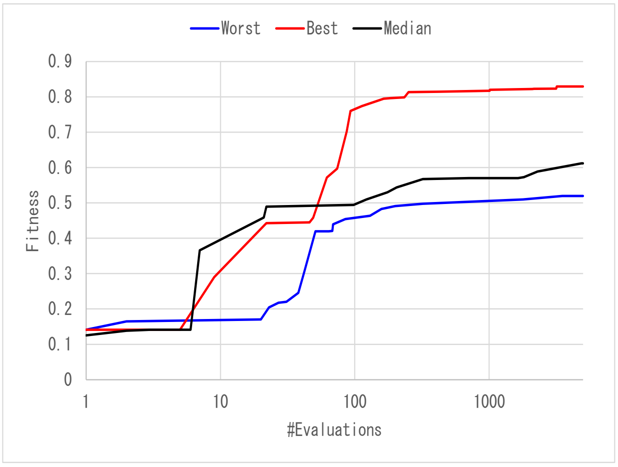
Units	Units	p-value	
16	8	0.01680	*
32	8	0.00093	**
32	16	0.42350	

(b) Population size = 50, 100 Generations.

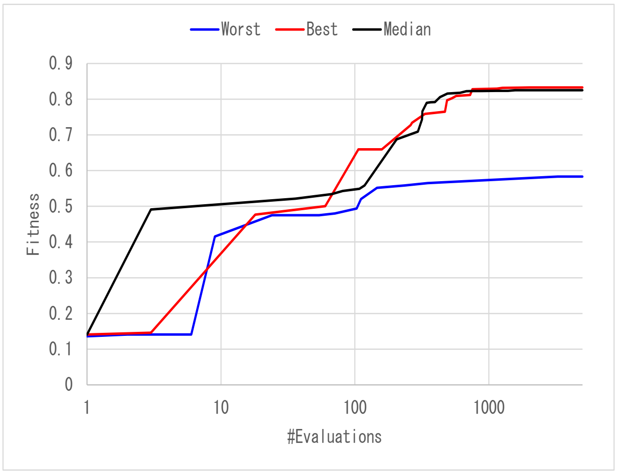
Units	Units	p-value	
16	8	0.01165	*
32	8	0.01999	*
32	16	0.71910	

(*)p<.05 (**)p<.01

Fig. 7 shows the best/median/worst learning curves among the 11 runs with 8 hidden units. Fig. 8 and Fig. 9 show those with 16 and 32 units respectively. These learning curves reveal that the fitness scores tend to increase slower while the scores are in [0.4, 0.6] and [0.7, 0.8]. Thus, it is easy for ES to train MLPs so that the MLPs achieve fitness scores of 0.4, but after that it becomes much difficult to train them so that they control the pendulum better. In the worst run among the 11 runs, ES failed to train MLPs to break thorough the score of 0.6. This result reveals a weakness of ES on robustly searching for better solutions.

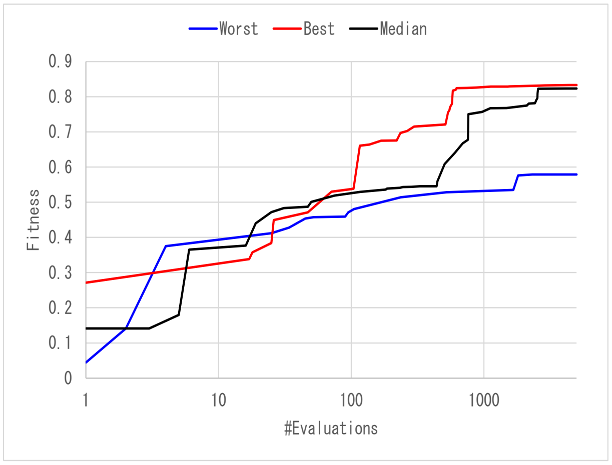


(a) Population size = 10, 500 Generations.

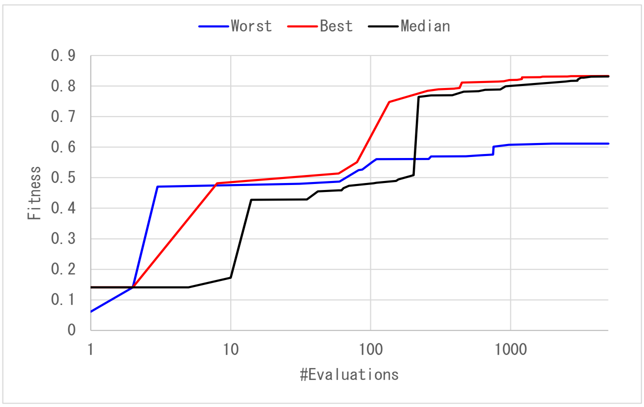


(b) Population size = 50, 100 Generations.

Figure 7. Learning curves with 8 hidden units.

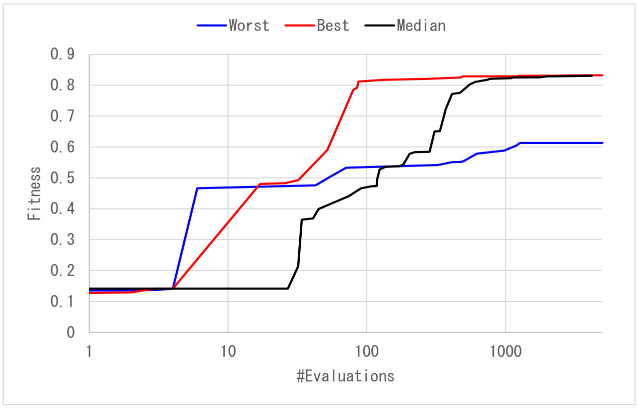


(a) Population size = 10, 500 Generations.

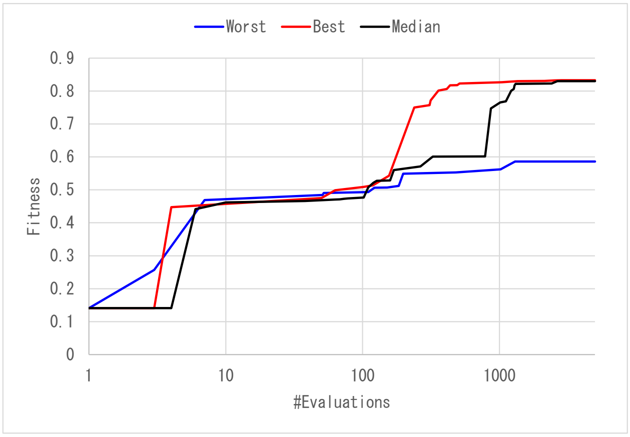


(b) Population size = 50, 100 Generations.

Figure 8. Learning curves with 16 hidden units.



(a) Population size = 10, 500 Generations.



(b) Population size = 50, 100 Generations.

Figure 9. Learning curves with 32 hidden units.

This experiment employs two sets of ES hyperparameters as shown in Table 1. Although the total number of evaluations are consistently 50,000 for both (a) and (b), (a) employs less population size and more generations while (b) employs more population size and less generations. Thus, ES with (a) parameters will be better at exploiting locally better solutions while ES with (b) parameters will better at exploring globally better solutions. Table 2 reveals that fitness scores are greater for (b) than (a) especially with 8 units. Wilcoxon rank sum tests are applied to test whether (b) is significantly better than (a) on this task. Table 4 shows the test result.

Table 4. Wilcoxon Rank Sum Tests for ES parameters.

Units	p-value	
8	0.01680	*
16	0.10850	
32	0.21920	

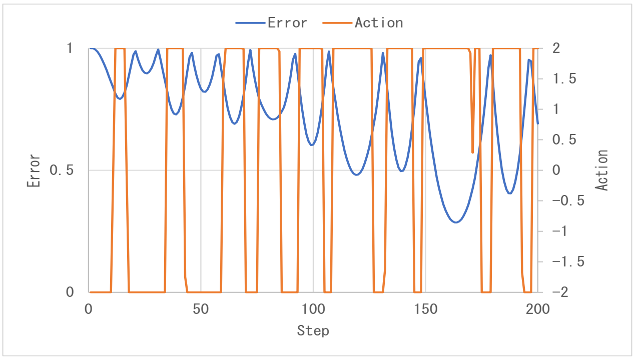
(*)p<.05

Table 4 reveals that (b) is significantly better than (a) with 8 units. Although the p-values are greater than 0.05 with 16 and 32 units, the p-values are much smaller than 0.5, which mean (b) is better than (a) with 16 and 32 units. Thus, on the task employed in this experiment, exploration contributes more for ES to search for better solutions.

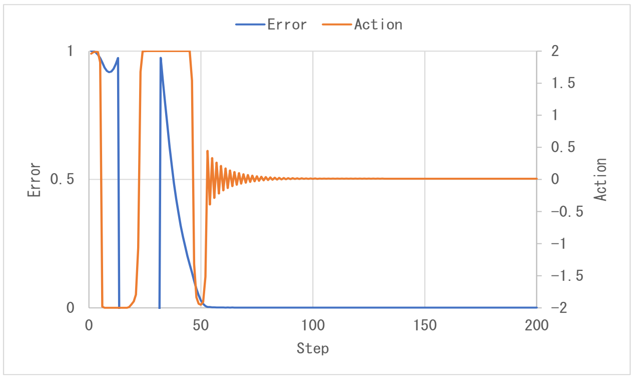
The author next reports how actions and errors are changed after the MLP is trained. Fig. 10(i)(ii) show the actions and errors by the MLP (i)before/(ii)after trained. To show the figures, MLPs with 16 units trained by ES with (a) parameters are employed.

Fig. 10(i) reveals that (1) the MLP before trained repeats the actions (the torque to the pendulum) of -2.0 and 2.0, (2) the error repeats decreasing and increasing, and (3) the error does not become small enough.

In contrast, Fig. 10(ii) reveals that (1) the MLP after trained successfully makes the error to be nearly zero (i.e., the MLP successfully makes the pendulum to be upright), and (2) after the error becomes nearly zero, the action also becomes quickly nearly zero so that the pendulum stays upright.
Movies which show how the pendulum is controlled by the MLPs before/after trained are presented as supplements.[4,5]



(i) before



(ii) after

Figure 10. Actions and errors by the MLP before/after trained.

VI. CONCLUSION

The author experimentally applied Evolution Strategy to reinforcement learning of a neural network controller for the pendulum control task. Experimental results revealed that ES could successfully train an MLP so that the trained MLP could make the pendulum upright quickly, if the MLP was equipped with sufficient hidden units. For the task adopted in this work, 8 hidden units were significantly worse than 16 and 32 hidden units while 32 hidden units were not significantly better than 16 units. Thus, 16 units are the best among the three variations, in terms of the task performance and the computational efficiency. Besides, the results revealed that exploration contributes more for ES to search for better solutions than exploitation. Further evaluations are required to confirm whether this finding holds for evolutionary algorithms other than ES. In addition, the author will further evaluate and improve evolutionary algorithms by applying them to reinforcement learning tasks other than the pendulum control.

REFERENCES

[1] T. Bäck, H.P. Schwefel, "An Overview of Evolutionary Algorithms for Parameter Optimization," *Evolutionary Computation*, Vol.**1**, No.**1**, pp.**1-23**, **1993**.

[2] D.B. Fogel, "An Introduction to Simulated Evolutionary Optimization," *IEEE Transactions on Neural Networks*, Vol.**5**, No.**1**, pp.**3-14**, **1994**.

[3] T. Bäck, "Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms," *Oxford University Press*, **1996**.

[4] A.E. Eiben, R. Hinterding, Z. Michalewicz, "Parameter Control in Evolutionary Algorithms," *IEEE Transactions on Evolutionary Computation*, Vol.**3**, No.**2**, pp.**124-141**, **1999**.

[5] A.E. Eiben, J.E. Smith, "Introduction to Evolutionary Computing (2nd ed.)," *Springer*, **2015**.

[6] C.J.C.H. Watkins, "Learning from Delayed Rewards," *PhD Thesis*, Cambridge University, **1989**.

[7] C.J.C.H. Watkins, P. Dayan, "Q-Learning," *Machine Learning*, Vol.**8**, No.**3**, pp.**279-292**, **1992**.

[8] R.S. Sutton, A.G. Barto, "Reinforcement Learning: An Introduction (2nd ed.)," *MIT Press*, **2018**.

[9] H.P. Schwefel, "Evolution Strategies: A Family of Non-Linear Optimization Techniques based on Imitating Some Principles of Organic Evolution," *Annals of Operations Research*, Vol.**1**, pp.**165-167**, **1984**.

[10] H.G. Beyer, H.P. Schwefel, "Evolution Strategies: A Compre-hensive Introduction," *Journal Natural Computing*, Vol.**1**, No.**1**, pp.**3-52**, **2002**.

[11] D.E. Goldberg, J.H. Holland, "Genetic Algorithms and Machine Learning," *Machine Learning*, Vol.**3**, No.**2**, pp.**95-99**, **1988**.

[12] J.H. Holland, "Genetic Algorithms," *Scientific American*, Vol.**267**, No.**1**, pp.**66-73**, **1992**.

[13] M. Mitchell, "An Introduction to Genetic Algorithms," *MIT Press*, **1998**.

[14] K. Sastry, D. Goldberg, G. Kendall, "Genetic Algorithms," *Search Methodologies*, Springer, pp.**97-125**, **2005**.

[15] R. Storn, K. Price, "Differential Evolution – A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces," *Journal of Global Optimization*, Vol.**11**, pp.**341-359**, **1997**.

[16] K. Price, R.M. Storn, J.A. Lampinen, "Differential Evolution: A Practical Approach to Global Optimization," *Springer Science & Business Media*, **2006**.

[17] S. Das, P.N. Suganthan, "Differential Evolution: A Survey of the State-of-the-art," *IEEE transactions on evolutionary computation*, Vol.**15**, No.**1**, pp.**4-31**, **2010**.

[18] D.E. Rumelhart, G.E. Hinton, R.J. Williams. "Learning Internal Representations by Error Propagation," in D.E. Rumelhart, J.L. McClelland, and the PDP research group (editors), "Parallel Distributed Processing: Explorations in the Microstructure of Cognition," Vol.**1**: Foundation. *MIT Press*, **1986**.

[19] R. Collobert, S. Bengio, "Links Between Perceptrons, MLPs and SVMs," *Proc. of the Twenty-First International Conference on Machine Learning (ICML'04)*, ACM, **2004**.

[20] X. Yao, Y. Liu, "A New Evolutionary System for Evolving Artificial Neural Networks," *IEEE Transactions on Neural Networks*, Vol.**8**, No.**3**, pp.**694-713**, **1997**.

[21] N.T. Siebel, G. Sommer, "Evolutionary Reinforcement Learning of Artificial Neural Networks," *Internatinal Journal of Hybrid Intelligent Systems*, Vol.**4**, No.**3**, pp.**171-183**. **2007**.

[22] K. Chellapilla, D.B. Fogel, "Evolving Neural Networks to Play Checkers Without Relying on Expert Knowledge," *IEEE Transactions on Neural Networks*, Vol.**10**, No.**6**, pp.**1382-1391**, **1999**.

[23] L. Cardamone, D. Loiacono and P. L. Lanzi, "Evolving Competitive Car Controllers for Racing Games with Neuro-evolution," *Proc. of 11th Annual Conference on Genetic and Evolutinary Computation*, pp.**1179-1186**, **2009**.

[24] S. Risi, J. Togelius, "Neuroevolution in Games: State of the Art and Open Challenges", *IEEE Transactions on Computational Intelligence and AI in Games*, Vol.**9**, No.**1**, pp.**25-41**, **2017**.

[25] J. Togelius, S.M. Lucas, "Evolving Controllers for Simulated Car Racing," *Proc. of 2005 IEEE Congress on Evolutionary Computation*, Vol.**2**, pp.**1906-1913**, **2005**.