# Training Neural Networks with {-1,1} Weights by Evolution Strategy

Hidehiko Okada
Faculty of Information Science and Engineering, Kyoto Sangyo University, Japan.
hidehiko@cc.kyoto-su.ac.jp

**Abstract: The author previously reported an experimental result of evolutionary reinforcement learning of neural network controllers. In the previous study, a conventional multilayer perceptron was employed in which connection weights were real numbers. In this study, the author experimentally applies an evolutionary algorithm to the reinforcement training of binary neural networks. In both studies, the same task and the same evolutionary algorithm are utilized, i.e. the Acrobot control problem and Evolution Strategy respectively. The differences lie in the memory size per connection weight and the model size of the neural network. The findings from this study are (1) the optimal number of hidden units for the binary MLP was 128 among the choices of 16, 32, 64, 128 and 256; (2) a larger population size contributed better for ES than a greater number of generations; and (3) binary connection weights can achieve comparable control performance while reducing memory size by half.**

**Keywords: evolutionary algorithm, evolution strategy, binary neural network, neuroevolution, reinforcement learning.**

## 1. Introduction

The author has been investigating a reinforcement learning approach for training neural networks using evolutionary algorithms. For instance, the author previously reported an experimental result of evolutionary reinforcement learning of neural network controllers for the Acrobot task [1]. In the previous study, a conventional multilayer perceptron was employed in which connection weights were real numbers. On the contrary, researchers are exploring neural networks in which the weights are discrete values rather than real numbers, accompanied by the corresponding learning methodologies [2-9]. An advantage of discrete neural networks lies in their ability to reduce the memory footprint required for storing trained models. To maximize the effect of memory size reduction achieved through neural network discretization, it is essential to limit the increase in model size while simultaneously minimizing the number of bits per connection weight.

In this study, the author experimentally applies an evolutionary algorithm to the reinforcement training of binary neural networks and compares the result with the previous experimental result [1] in which real-valued neural networks were employed. In both studies, the same task and the same evolutionary algorithm are utilized, i.e. the Acrobot control problem and Evolution Strategy respectively. The differences lie in the memory size per connection weight and the overall model size of the neural network.

## 2. Acrobot Control Task

As a task that requires reinforcement learning to solve, this study employs the Acrobot control task provided at OpenAI Gym. Figure 1 shows a screenshot of the system. The webpage for this system describes as follows[1]; *The system consists of two links connected linearly to form a chain, with one end of the chain*

[1] https://www.gymlibrary.dev/environments/classic_control/acrobot/

*fixed. The joint between the two links is actuated. The goal is to apply torques on the actuated joint to swing the free end of the linear chain above a given height while starting from the initial state of hanging downwards.*

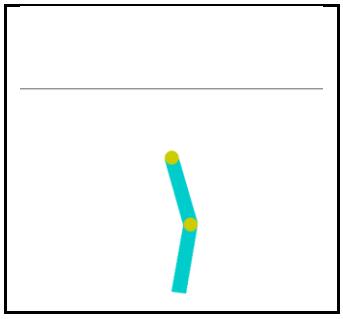
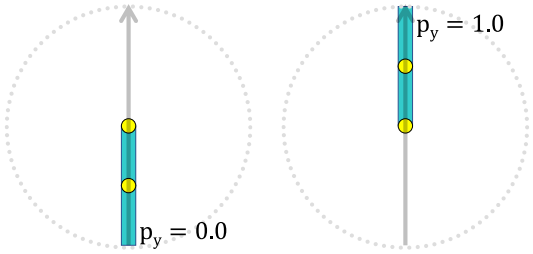


Figure 1: Acrobot system[1].

Let $p_y$ denote the height of the free end of the linear chain, where the minimum (maximum) value of $p_y$ is 0.0 (1.0) as shown in Figure 2. The goal of the task is originally to achieve $p_y \geq 0.5$, and an episode is finished when the goal is achieved or the time step reaches to a preset limit. In this study, the goal is changed so that the free end of the linear chain is kept as high as possible (i.e., let the value of $p_y$ as greater as possible) throughout an episode, where an episode consists of 200 time steps. Besides, the author changed the system so that (i) the control task starts with the state shown in Figure 2(a) where $p_y$=0.0, and (ii) the applicable torque to the actuated joint is continuous within [-1.0, 1.0] while the torque is originally discrete (either of -1, 0 or 1).




(a) initial state (b) best state

Figure 2: Initial and best states.

In each step, the controller observes the current state and then determines the action. An observation obtains $\cos(\theta_1)$, $\sin(\theta_1)$, $\cos(\theta_2)$, $\sin(\theta_2)$, and the angular velocity of $\theta_1$ and $\theta_2$, where $\theta_1$ is the angle of the first joint and $\theta_2$ is relative to the angle of the first link[1]. The ranges are $-1.0 \leq \cos(\theta_1), \sin(\theta_1), \cos(\theta_2), \sin(\theta_2) \leq 1.0$, $-4\pi \leq$ angular velocity of $\theta_1 \leq 4\pi$, and $-9\pi \leq$ angular velocity of $\theta_2 \leq 9\pi$ respectively.

In this study, the author defines the fitness of a neural network controller as shown in eq. (1). In eq. (1), $p_y(t)$ denotes the height $p_y$ at each time step t. The fitness score is larger as $p_y(t)$ is larger for more time steps. Thus, a controller fits better as it can keep the free end of the linear chain as higher as possible.

$$\text{Fitness} = \frac{1}{200}\sum_{t=1}^{200} p_y(t) \tag{1}$$

## 3. Neural Networks with Binary Connection Weights

In the previous study [1] the author employed a three-layered feedforward neural network known as a multilayer perceptron (MLP) as the controller. An MLP with the same topology is utilized again in this study, where connection weights are binary in this study while real numbers in the previous study. Figure 3 illustrates the topology of the MLP. The feedforward calculations are the same as those described in [1]. Note that the unit activation function is the hyperbolic tangent (tanh), which is the same as in the previous study. Thus, the MLP with binary weights outputs real numbers within the range [-1.0, 1.0].

In both of this study and the previous one, the MLP serves as the policy function: action(t) = F(observation(t)). The input layer consists of three units (N=6 in Figure 3), each corresponding to the values obtained by an observation. The output layer comprises one unit (L=1 in Figure 3), and its output value is applied as the torque to the pendulum system.

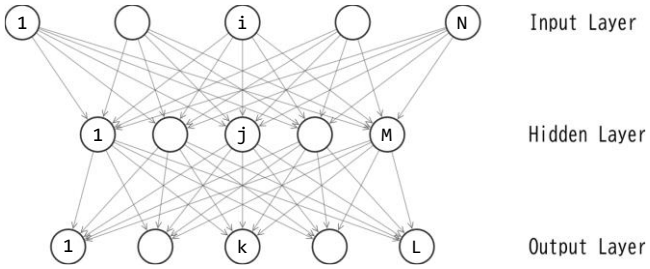



**Figure 3.** Topology of the MLP.

## 4. Training of Binary Neural Networks by Evolution Strategy

A three-layered perceptron, as depicted in Figure 3, includes M+L unit biases and NM+ML connection weights, resulting in a total of M+L+NM+ML parameters. Let D represent the quantity M+L+NM+ML. For this study, the author sets N=6 and L=1, leading to D=8M+1. The training of this perceptron is essentially an optimization of the D-dimensional binary vector. Let $\mathbf{x} = (x_1, x_2, \ldots, x_D)$ denote the D-dimensional vector, where each $x_i$ corresponds to one of the D parameters in the perceptron. In this study, each $x_i$ is a binary variable, $x_i \in \{-1,1\}$. By applying the value of each element in **x** to its corresponding connection weight or unit bias, the feedforward calculations can be processed.

In this study, the binary vector **x** is optimized using Evolution Strategy [10-12]. ES treats **x** as a chromosome (a genotype vector) and applies evolutionary operators to manipulate it. The fitness of **x** is evaluated based on eq. (1), which is the same as in the previous study [1]. Figure 4 illustrates the ES process. The process is the same as that in the previous study [1]. In Step 1, D-dimensional binary vectors $\boldsymbol{y}^1, \boldsymbol{y}^2, \ldots, \boldsymbol{y}^C$ are randomly initialized where $C$ denotes the number of offsprings. A larger value of $C$ promotes explorative search more. In Step 2, values in each vector $\boldsymbol{y}^c$ $(c = 1,2,\ldots,C)$ are applied to the MLP and the MLP controls the Acrobot for a single episode with 200 time steps. The fitness of $\boldsymbol{y}^c$ is then evaluated with the result of the episode. Let $f(\boldsymbol{y}^c)$ 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 $P + C$ vectors in the parent population $(\boldsymbol{z}^1, \boldsymbol{z}^2, \ldots, \boldsymbol{z}^P)$ and the offspring population $(\boldsymbol{y}^1, \boldsymbol{y}^2, \ldots, \boldsymbol{y}^C)$, 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 offspring population $(\boldsymbol{y}^1, \boldsymbol{y}^2, \ldots, \boldsymbol{y}^C)$. In Step 5, new $C$ offspring vectors are produced by applying the reproduction operator to the parent vectors $(\boldsymbol{z}^1, \boldsymbol{z}^2, \ldots, \boldsymbol{z}^P)$ which are selected in the last Step 4. The new offspring vectors form the new offspring population $(\boldsymbol{y}^1, \boldsymbol{y}^2, \ldots, \boldsymbol{y}^C)$. Figure 4 denotes the process of reproduction. This reproduction process is slightly different from that in the previous study [1] because the genotype vectors are not real-valued vectors but binary ones. In Step5-4, each of $y_1^c, y_2^c, \ldots, y_D^c$ is mutated under the probability $pm$. Let denote $b_S$ ($b_G$) is the smaller (greater) value for the binary parameter, e.g. { $b_S$, $b_G$ } = {-1, 1}. The mutation flips the value of $y_d^c$ from $b_S$ to $b_G$ (or from $b_G$ to $b_S$). A greater value of $pm$ promotes explorative search more.

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

**Figure 4.** Process of Evolution Strategy.

Step 5-1. Let $c = 1$.
Step 5-2. A vector is randomly sampled from the parent population $\boldsymbol{z}^1, \boldsymbol{z}^2, \ldots, \boldsymbol{z}^P$. Let $\boldsymbol{z}^p$ denote the sampled vector.
Step 5-3. A copy of $\boldsymbol{z}^p$ is created as $\boldsymbol{y}^c$. $\boldsymbol{y}^c$ is a $D$-dimensional binary vector, i.e., $\boldsymbol{y}^c = (y_1^c, y_2^c, \ldots, y_D^c)$.
Step 5-4. Each of $y_1^c, y_2^c, \ldots, y_D^c$ is mutated under the probability $pm$.
Step 5-5. If $c < C$ then $c \leftarrow c + 1$ and goto Step 5-2, else finish the reproduction.

**Figure 5.** Reproduction process in Evolution Strategy.

## 4. Experiment

In the previous study using MLPs with real-valued connection weights, the number of fitness evaluations included in a single run was set to 5,000 [1]. The number of new offsprings generated per generation was either of (a) $C$=10 and (b) $C$=50. The number of generations for each case of (a) or (b) was 500 and 100 respectively. The total number of fitness evaluations were 10 × 500 = 5,000 for (a) and 50 × 100 = 5,000 for (b). The experiments in this study, using MLPs with binary connection weights, employ the same configurations except that the number of generations are 1,000 for (a) and 200 for (b). Thus, the number of fitness evaluations included in a single run was 10,000 for both of (a) and (b). The hyperparameter configurations for ES are shown in Table 1. The number of parents, denoted as $P$, is set to 10% of the $C$ offsprings for both (a) and (b). The mutation probability $pm$ is set to 0.01. The values of $P$ and $pm$ were adjusted based on preliminary experiments.

As the two values for the binary connection weights, {-1,1} is used in this study. In the previous study using real-valued MLPs, it was found that, among 8, 16, or 32 hidden units, 8 units yielded the most desirable results [1]. The binary MLPs used in this study are expected to require a greater number of hidden units to achieve performance comparable to the real-valued MLP with 8 hidden units. Therefore, in this experiment,

the author tested five options: 16, 32, 64, 128 and 256. A binary MLP with either of 16, 32, 64, 128 or 256 hidden units underwent independent training 11 times.

**Table 1.** ES Hyperparameter Configurations.

| **Hyperparameters** | **(a)** | **(b)** |
|---|---|---|
| Number of offsprings ($C$) | 10 | 50 |
| Generations | 1,000 | 200 |
| Fitness evaluations | 10×1,000=10,000 | 50×200=10,000 |
| Number of parents ($P$) | 10×0.1=1 | 50×0.1=5 |
| Mutation probability ($pm$) | 0.01 | 0.01 |

Table 2 presents the best, worst, average, and median fitness scores of the trained MLPs across the 11 runs. Each of the two hyperparameter configurations (a) and (b) in Table 1 was applied. A larger value is better in Table 2 with a maximum of 1.0.

**Table 2.** Fitness Scores among 11 Runs.

| | **M** | **Best** | **Worst** | **Average** | **Median** |
|---|---|---|---|---|---|
| **(a)** | **16** | 0.391 | 0.330 | 0.363 | 0.356 |
| | **32** | 0.421 | 0.376 | 0.401 | 0.401 |
| | **64** | 0.432 | 0.245 | 0.395 | 0.412 |
| | **128** | 0.440 | 0.385 | 0.424 | 0.428 |
| | **256** | 0.448 | 0.305 | 0.418 | 0.426 |
| **(b)** | **16** | 0.386 | 0.347 | 0.362 | 0.360 |
| | **32** | 0.419 | 0.377 | 0.396 | 0.394 |
| | **64** | 0.440 | 0.394 | 0.416 | 0.413 |
| | **128** | 0.448 | 0.426 | 0.435 | 0.433 |
| | **256** | 0.448 | 0.426 | 0.438 | 0.438 |

In order to investigate which of the two configurations (a) or (b) is superior, the Wilcoxon signed-rank test was applied to the 20×2 data points presented in Table 2. This test revealed that configuration (b) is better than configuration (a) with a statistical significance ($p < .01$). Thus, increasing the population size and reducing the number of generations is more favorable than the opposite approach. ES excels in local exploitation but struggle with global exploration. Increasing the population size can enhance the performance of evolutionary algorithms on global exploration, thereby compensating for this weakness. Consequently, it is posited that increasing the population size improves the balance between global and local search, resulting in better performance compared to increasing the number of generations.

Next, the author examines whether there is a statistically significant difference in the performances among the three MLPs with the different numbers of hidden units M. For each M of 16, 32, 64, 128 and 256, the author conducted 11 runs using the configuration (b), resulting in 11 fitness scores. The Wilcoxon rank sum test was applied to the 11×5 data points. The results showed that,

(1) M=16 was significantly worse than any of M=32, 64, 128 and 256 ($p < .01$),
(2) M=32 was significantly worse than any of M=64, 128 and 256 ($p < .01$), and
(3) M=64 was significantly worse than any of M=128 and 256 ($p < .01$).

M=128 was worse than M=256 but the difference was not statistically significant ($p > .05$). Therefore, from the perspective of the trade-off between performance and memory size, the most desirable number of hidden units was found to be 128.

Figure 6 presents learning curves of the best, median, and worst runs among the 11 runs where the configuration (b) was applied. Note that the horizontal axis of these graphs is in a logarithmic scale. Figures 6(i)-(v) depict learning curves of MLPs with different numbers of hidden units respectively (M=16, 32, 64, 128 and 256). The five median curves depicted in Figures 6(i)-(v) exhibit similar shapes; however, it is evident that the median curve in Figure 6(i) shows a slower increase in values compared to the others, while the median curve in Figure 6(v) demonstrates a more rapid increase. As the number of hidden units increases, the non-linear function approximation capability of the MLP improves. However, this also leads to a larger length D of the binary vector that ES must optimize, thereby increasing the difficulty of the optimization task. The observation that larger values of M correspond to a greater learning rate indicates that ES was able to efficiently optimize even in high-dimensional vector spaces.

Figure 7(a) illustrates the actions by the MLP and the heights $p_y(t)$ in the 200 steps prior to training, while Figure 7(b) displays the corresponding actions and heights after training. In this scenario, the MLP employed 128 hidden units, and the configuration (b) was utilized. These figures exhibit a similarity to Figure 8 in the previous report [1] that shows the results using real-valued MLPs. Thus, the binary MLP with a sufficient number of hidden units can control the Acrobot as well as the real-valued MLP could. For the real-valued MLP, the optimal number of hidden units was 8 [1], and the total number of parameters in this model is 65. Each real parameter occupies 32 bits, and the memory requirement for this real-valued MLP amounts to $32 \times 65 = 2080$ bits. In contrast, the optimal number of hidden units for the binary MLP, as previously mentioned, is 128, resulting in a total of 1025 parameters. Given that each binary parameter requires only 1 bit, the total memory size for the binary MLP is $1 \times 1025 = 1025$ bits. The memory requirement for the binary MLP is approximately 49% of that needed for the real-valued MLP. This indicates that the binary MLP achieves comparable control performance to the real-valued MLP while utilizing only half the memory size. Supplementary videos[2,3] are provided which demonstrate the motions of the chain controlled by the binary MLP with 128 hidden units.

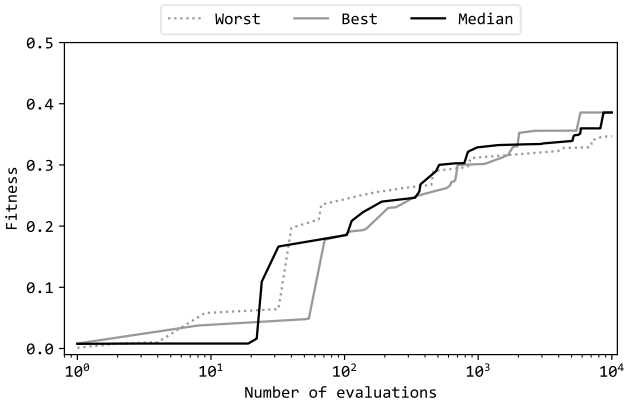
## 5. Conclusion

In this study, Evolution Strategy was applied to the reinforcement learning of a neural network controller for the Acrobot task, where the connection weights in the neural network are not real numbers but binary. The findings from this study are summarized as follows:

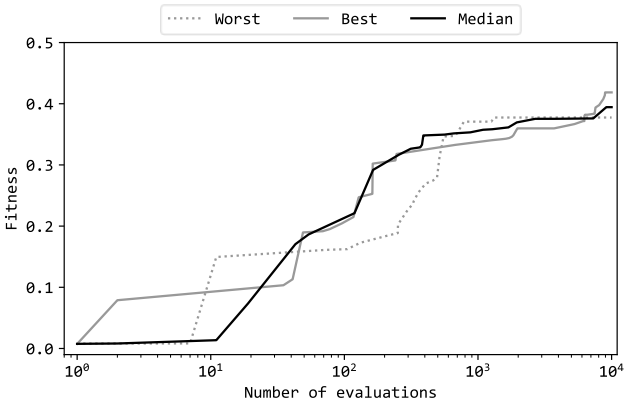
(1) The optimal number of hidden units for the binary MLP was found to be 128 among the choices of 16, 32, 64, 128 and 256.
(2) For the ES hyperparameters, the configuration (b) yielded superior results compared to the configuration (a) in Table 1. The configuration (b) involved a larger population size and a smaller number of generations compared to the configuration (a).
(3) The motion of the Acrobot controlled by a binary MLP with 128 hidden units was similar to that by a real-valued MLP with 8 hidden units. The memory requirements for the binary MLP with 128 hidden units were only 49% of those needed for the real-valued MLP with 8 hidden units. This indicates that binary connection weights can achieve comparable control performance while making the memory size a half.

The author plans to further apply and evaluate other evolutionary algorithms to the same task and compare the performance.

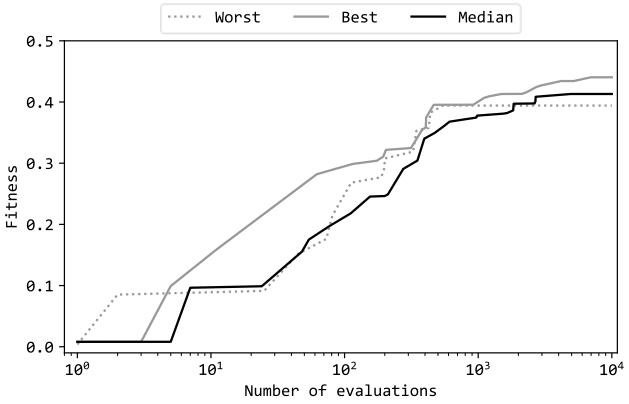
[2] http://youtu.be/zZq9iLo4pkc
[3] http://youtu.be/IB-HgKuppHw



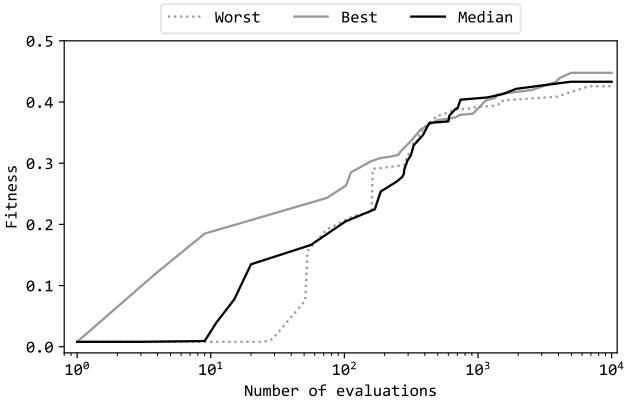

(i) M=16



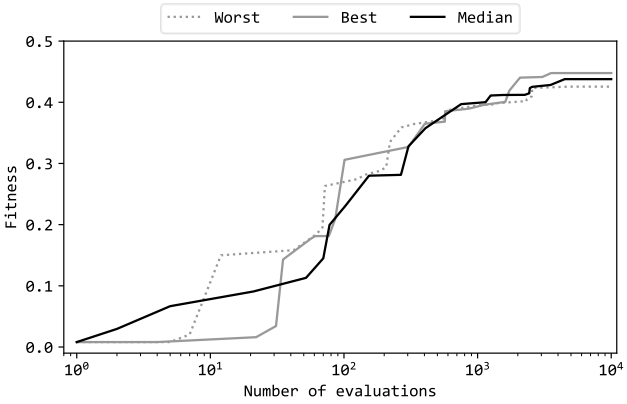

(ii) M=32




(iii) M=64

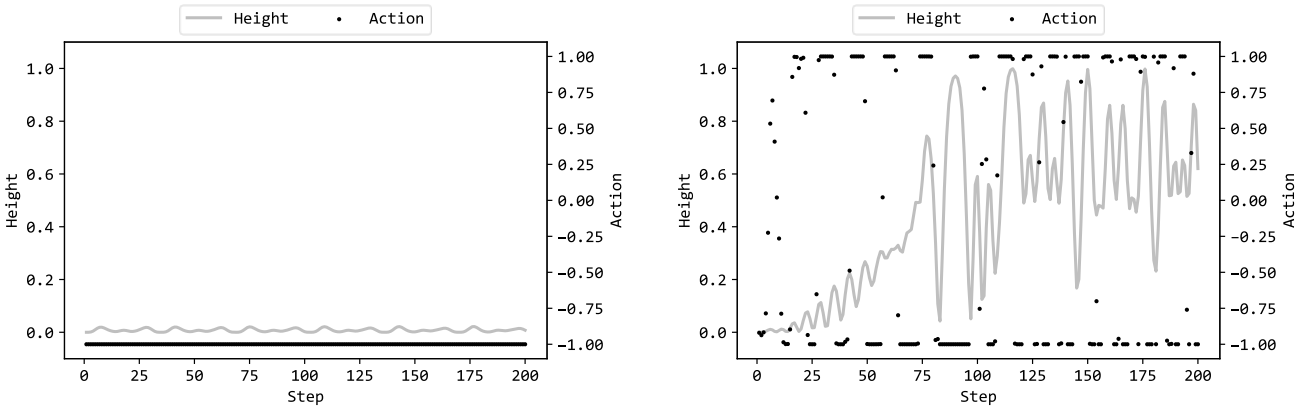



(iv) M=128




(v) M=256

**Figure 6**. Learning curves of MLP with M hidden units.




(a) Before the training. (b) After the training.

**Figure 7**. MLP actions and errors in an episode.

**Acknowledgments:** The author conducted this study under the Official Researcher Program of Kyoto Sangyo University.

## References

[1] Okada, H (2023). Evolutionary reinforcement learning of neural network controller for Acrobot task — Part1: Evolution Strategy. Preprints.org. doi: 10.20944/preprints202308.0081.v1.

[2] Courbariaux, M., Bengio, Y., & David, J.P. (2015). BinaryConnect: training deep neural networks with binary weights during propagations. Proceedings of the 28th International Conference on Neural Information Processing Systems (NIPS'15), 2, MIT Press, 3123–3131.

[3] Courbariaux, M., Hubara, I., Soudry, D., El-Yaniv, R., & Bengio, Y. (2016). Binarized neural networks: training deep neural networks with weights and activations constrained to +1 or -1. arXiv preprint arXiv:1602.02830.

[4] Tang, W., Hua, G., & Wang, L. (2017). How to train a compact binary neural network with high accuracy?. Proceedings of the AAAI Conference on Artificial Intelligence, 31(1).

[5] Lin, X., Zhao, C., & Pan, W. (2017). Towards accurate binary convolutional neural network. Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS'17), 344–352.

[6] Bethge, J., Yang, H., Bartz, C., & Meinel, C. (2018). Learning to train a binary neural network. arXiv preprint arXiv:1809.10463.

[7] Qin, H., Gong, R., Liu, X., Bai, X., Song, J., & Sebe, N. (2020). Binary neural networks: a survey. Pattern Recognition, 105, 107281. doi.org/10.1016/j.patcog.2020.107281.

[8] Yuan, C., & Agaian, S.S. (2023). A comprehensive review of binary neural network. Artificial Intelligence Review, 56, 12949–13013. doi.org/10.1007/s10462-023-10464-w.

[9] Sayed, R., Azmi, H., Shawkey, H., Khalil, A. H., & Refky, M. (2023). A systematic literature review on binary neural networks. IEEE Access, 11, 27546–27578. doi: 10.1109/ACCESS.2023.3258360.

[10] Schwefel, H.P. (1984). Evolution strategies: a family of non-linear optimization techniques based on imitating some principles of organic evolution. Annals of Operations Research, 1, 165–167.

[11] Schwefel, H.P. (1995). Evolution and Optimum Seeking. Wiley & Sons.

[12] Beyer, H.G., & Schwefel, H.P. (2002). Evolution strategies: a comprehensive introduction. Journal Natural Computing, 1(1), 3–52.