Binary Neural Networks Playing Atari Space Invaders (1) Trained by Evolution Strategy

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

Abstract: This study investigates the application of Evolution Strategy (ES) to train binary neural network controllers for the Atari game *Space Invaders*, extending previous work for control tasks such as Pendulum and Acrobot. Unlike conventional networks using real-valued weights, this approach represents connection weights using binary values from the set {-1, 1}. Experimental results evaluate the performance of multilayer perceptrons (MLPs) with varying numbers of hidden units and weight bit precision (1-bit vs. 64-bit). Key findings indicate that 1-bit MLPs achieve comparable or superior performance to 64-bit MLPs, particularly when using 16 hidden units. Moreover, performance does not degrade significantly even with minimal hidden units, suggesting that binary quantization may not necessitate increased model complexity. Additionally, results demonstrate that increasing the number of offspring per generation enhances ES effectiveness more than increasing the number of generations. These findings highlight the potential of binary-weight neural networks for efficient and effective reinforcement learning in resource-constrained settings.

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

1. Introduction

The author has previously reported experimental results on evolutionary reinforcement learning for neural network controllers applied to the Pendulum task [1,2] and the Acrobot task [3,4]. In those studies, connection weights between neurons were represented not as real numbers, but as binary values from the set {-1, 1}. In this paper, a new experimental result is presented, focusing on the evolutionary training of binary neural networks to play the Atari game *Space Invaders*. The training is performed using Evolution Strategy (ES) [5,6]. The network topology and the activation function are kept consistent with the prior studies. This study compares two different ES configurations, as well as varying the number of hidden units and the number of bits used per connection weight.

2. Atari Space Invaders

Atari Space Invaders is a reinforcement learning task available in the OpenAI Gym (and Gymnasium[1]) framework, based on the classic arcade game Space Invaders. This task serves as a widely used benchmark for evaluating agents that learn decision-making from visual input. Figure 1 shows an example of the game screen.

In the game, the player (agent) controls a spaceship that can move horizontally along the bottom of the screen and aims to shoot down waves of enemies (invaders) descending from above. The agent can select from the following six discrete actions: (1) no operation (NOOP), (2) move left, (3) move right, (4) fire, (5) move left while firing, or (6) move right while firing.

[1] https://ale.farama.org/environments/space_invaders/

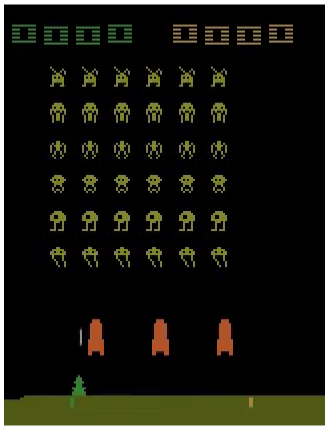


Figure 1. Screenshot of Atari Space Invaders.

In Atari environments, two types of observation modalities are available: image-based observations and RAM-based observations. In the image-based observation mode, each observation consists of a frame rendered by the game, with a resolution of 210×160 pixels and three RGB color channels. While the raw images can be used directly, it is common practice to apply preprocessing techniques such as grayscale conversion and downsampling to reduce dimensionality and improve learning efficiency. In contrast, the RAM-based observation mode provides a snapshot of the internal RAM state of the Atari 2600. Each observation is a one-dimensional array of length 128, with each element represented as an 8-bit unsigned integer (uint8) ranging from 0 to 255. This array contains compact, low-level information about the game state, such as score, enemy positions, and projectile coordinates, directly encoded in memory. In this study, the author employs the RAM-based environment, `ALE/SpaceInvaders-ram-v5`.

The reward in the Space Invaders environment corresponds directly to the in-game score. The agent receives a positive reward when it successfully destroys an enemy, with the magnitude depending on the enemy type. No reward is given for missed shots or movement alone. The total cumulative reward at the end of an episode reflects the agent's performance and serves as the main evaluation metric. In addition to standard enemy units, the game occasionally spawns a UFO (also known as the "mystery ship") that traverses the top of the screen. Successfully shooting down the UFO yields a relatively high bonus reward, making it a valuable target for maximizing the cumulative score. The game terminates if the agent loses all available lives.

3. Neural Networks with Binary Connection Weights

This study employs a feedforward neural network, known as a multilayer perceptrons (MLP), as the controller to the game. Figure 2 illustrates the topology of the network. A single hidden layer is included, and the connection weights are binary, i.e., either -1 or 1. The unit activation function is the hyperbolic tangent (tanh) so that the network outputs real numbers within the range [-1.0, 1.0]. The same network was employed in the prior studies; the feedforward calculations are the same as those described in [1-4]. The MLP serves as the policy function: action(t) = F(observation(t)). The input layer consists of 128 units (N=128 in Figure 2) so that each unit can receive corresponding value in the 128-dimensional vector of game observation, where each value is normalized from [0, 255] into [-1.0, 1.0]. The output layer comprises six units (L=6 in Figure 2), each of which corresponds to one of the six actions. The game agent selects the action whose unit outputs the largest value.

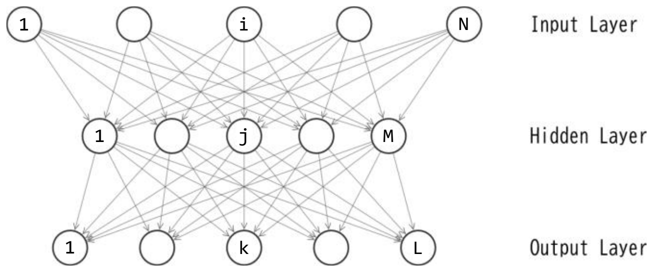


Figure 2. Topology of the MLP.

4. Training of Binary Neural Networks by Evolution Strategy

The three-layered perceptron, as depicted in Figure 2, 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=128 and L=6, leading to D=135M+6. 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 $\mathbf{x}$ to its corresponding connection weight or unit bias, the feedforward calculations can be processed.

In this study, the binary vector $\mathbf{x}$ is optimized using Evolution Strategy [5,6]. ES treats $\mathbf{x}$ as a chromosome (a genotype vector) and applies evolutionary operators to manipulate it. The fitness of $\mathbf{x}$ is the game score played by the MLP as the phonotype of $\mathbf{x}$. Figure 3 illustrates the ES process. The process is the same as that in the previous studies [1,3]. 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 exploration more. In Step 2, values in each vector $\boldsymbol{y}^c$ $(c = 1,2,\ldots,C)$ are applied to the MLP and the MLP play a game. The fitness of $\boldsymbol{y}^c$ is then evaluated with the game score. 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 exploitation 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. 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. $b_S = -1$ and $b_G = 1$. 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 exploration more.

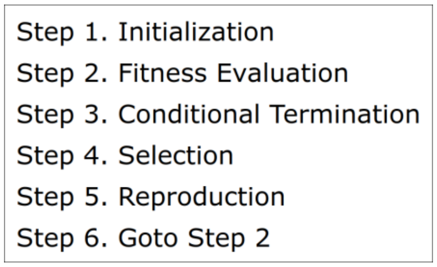

Step 1. Initialization
Step 2. Fitness Evaluation
Step 3. Conditional Termination
Step 4. Selection
Step 5. Reproduction
Step 6. Goto Step 2

Figure 3. Process of Evolution Strategy.

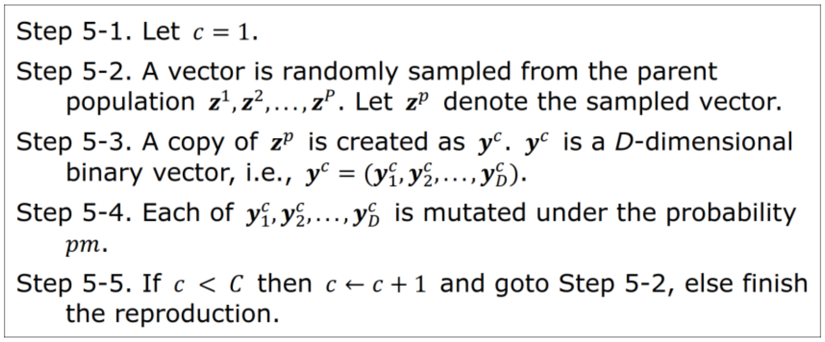

Step 5-1. Let $c = 1$.
Step 5-2. A vector is randomly sampled from the parent population $\boldsymbol{z}^1, \boldsymbol{z}^2, \ldots, \boldsymbol{z}^P$. Let $\boldsymbol{z}^p$ denote the sampled vector.
Step 5-3. A copy of $\boldsymbol{z}^p$ is created as $\boldsymbol{y}^c$. $\boldsymbol{y}^c$ is a D-dimensional 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 4. Reproduction process in Evolution Strategy.

4. Experiment

4.1 Neural Network

The neural network employed in this experiment consists of three layers, including a single hidden layer. Since the author utilized the `ALE/SpaceInvaders-ram-v5` environment, each observation yields 128 numerical values. To use these values as input to the neural network, the input layer is configured with 128 units. The game allows for six possible actions; therefore, the output layer is designed with six units, and the action corresponding to the unit with the highest output value is selected as the network's decision. The author compares five configurations for the number of hidden units: 1, 2, 4, 8, and 16. The activation function used for both the hidden and output layers is the hyperbolic tangent (tanh) function. As a result, the outputs of the hidden and output layer units are real values in the range (-1.0, 1.0).

To compare the performance of a neural network with binarized connection weights against that of a conventional (non-quantized) neural network of the same topology, experiments are conducted using both binary and real-valued neural networks. In the binary neural network, connection weights take on values of either -1 or 1 and are initialized using a uniform random distribution. In the real-valued neural network, connection weights are represented as 64-bit floating-point numbers. Values outside the range [-10.0, 10.0] are clipped to remain within this interval. The weights are initialized using a uniform random distribution in the range [-1.0, 1.0].

4.2 Evolution Strategy

Let C denote the number of offspring generated per generation, and G the maximum number of generations. Then, the total number of individuals evaluated in a single run of ES is given by C × G. A larger C facilitates broader exploration in the early generations, while a larger G promotes more thorough local search in the later generations. In this study, two different combinations of C and G were configured such that C × G = 1000 (see Table 1). Configuration (a) emphasizes local exploration more than configuration (b), whereas configuration (b) emphasizes global exploration more than configuration (a). The number of parent individuals P was set to 10% of C and was kept constant across both configurations (a) and (b).

Table 1. ES hyperparameter configurations.

Hyperparameters	**(a)**	**(b)**
Number of offsprings (C)	10	50
Generations (G)	100	20
Fitness evaluations	10×1,00=1,000	50×20=1,000
Number of Parents (P)	10×0.1=1	50×0.1=5

The perturbation method for generating offspring differs depending on whether connection weights are represented using 1-bit or 64-bit precision. In the case of 1-bit weights, the genotype is a binary vector, and each element of the vector undergoes bit-flip mutation with a fixed probability (see Section 3). In this experiment, the bit-flip probability is set to 1%. The total number of weights and biases in the network with M hidden units is 135M + 6, which is also the length of the genotype vector. For example, when M = 10, the genotype vector length is 1356. Therefore, with a 1% bit-flip probability, the expected number of mutated bits per individual is approximately 13.56. On the other hand, when weights are represented with 64-bit precision, the genotype is a real-valued vector, and bit-flipping is not applicable. Instead, a different perturbation method is employed: adding a small random noise vector ***r*** to the genotype. Each element r_i of vector ***r*** is drawn from a uniform distribution over the interval [-R, R]. In this study, the value of R is set to 0.25.

4.3 Results

Table 2 presents the best, worst, average, and median game scores by the trained MLPs across the 11 runs. Each of the two hyperparameter configurations (a) and (b) in Table 1 was applied. A larger score is better in Table 2. 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(i). This test revealed that configuration (b) is better than configuration (a) with a statistical significance ($p < .01$). This result suggests that increasing the number of offspring generated per generation is more effective than simply increasing the number of generations. As ES is effective at local search but less capable in global exploration, increasing the number of offspring helps compensate for this limitation by enhancing global search capability. Consequently, this adjustment achieves a better balance between global and local search, leading to improved overall performance. This finding is consistent with the previous study [3]; when training binary neural networks using ES, prioritizing a larger number of offspring over a higher number of generations is preferable.

Next, the author examines whether there is a statistically significant difference in the performances among the five binary MLPs with the different numbers of hidden units M. For each M of 1, 2, 4, 8 and 16, the author conducted 11 runs using the configuration (b), resulting in 11 game scores. The Wilcoxon rank sum test was applied to the 11×5 data points. The results showed that no statistically significant differences were observed between any pair of values for M (with all pairwise comparisons yielding p>.05). For instance, the performance with M=1 was not significantly worse than with M=2, 4, 8, or 16. Generally, binarizing connection weights in neural networks tends to degrade performance, which is typically compensated for by increasing the number of hidden units. However, the present experiment reveals that, even with quantized neural networks, increasing the number of hidden units is not necessary to maintain performance. One possible explanation is that the game environment used in this study provided RAM-based observations rather than image-based ones. This significantly reduced the dimensionality of the input data and, moreover, the RAM observations likely included task-relevant information that directly contributed to achieving high scores in the game. In contrast, when using image-based observations, a larger number of hidden units may be necessary to extract useful features—an issue that remains an open question for future work.

Table 2. Fitness scores among 11 runs.

(i) 1bit

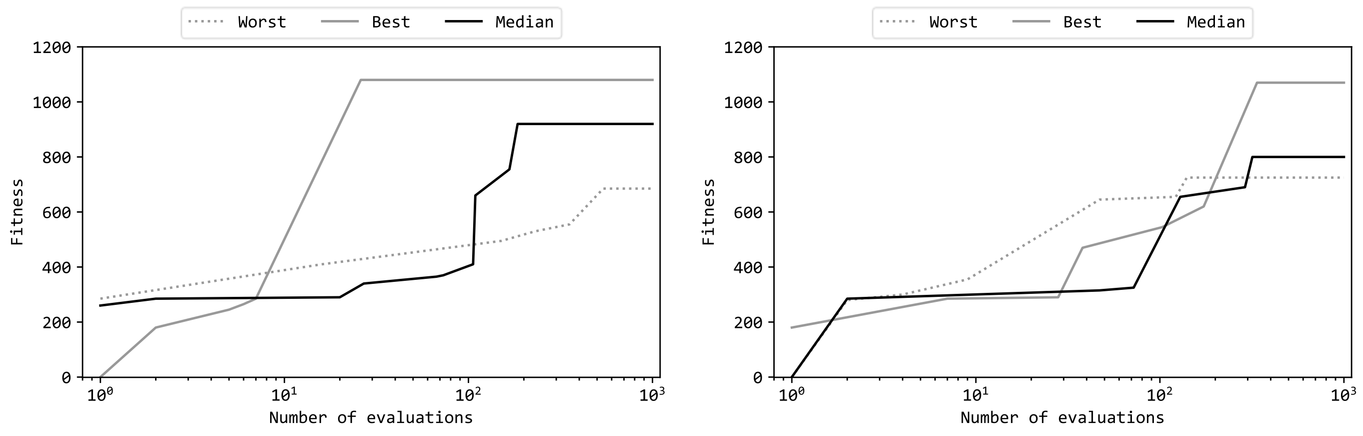
	M	**Best**	**Worst**	**Average**	**Median**
	1	1035	285	662.7	760
	2	1185	285	770.9	805
(a)	**4**	1125	285	839.5	850
	8	1070	500	863.2	890
	16	1175	665	850.9	830
	1	1080	685	912.7	920
	2	1090	745	888.6	860
(b)	**4**	1680	800	971.4	905
	8	1285	800	918.2	850
	16	1155	825	965.0	975

(ii) 64bit

	M	**Best**	**Worst**	**Average**	**Median**
	1	1035	285	667.3	785
	2	1085	310	806.4	850
(a)	**4**	1015	810	923.2	955
	8	1140	610	831.8	800
	16	1250	640	855.0	845
	1	1070	725	842.3	800
	2	1005	800	872.3	860
(b)	**4**	1000	720	851.8	840
	8	1095	775	890.5	860
	16	1025	765	860.9	840

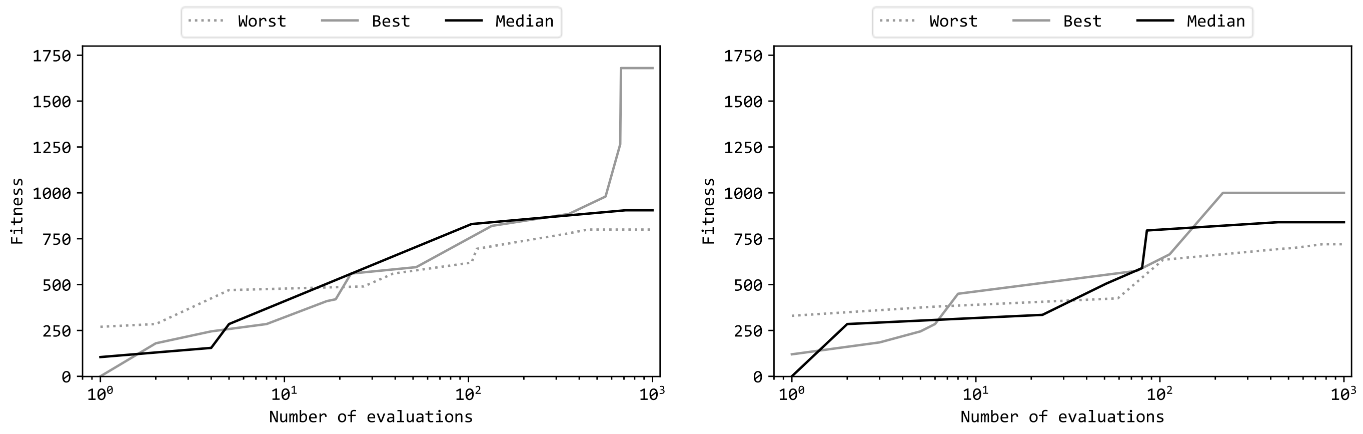
Next, the author compares the performance of the 1-bit MLP with that of the 64-bit MLP having the same topology. For all four variations M=1,2,4, or 8, the 1-bit MLP did not perform significantly worse than the 64-bit MLP (all comparisons yielded $p>.05$). Moreover, in the case of M=16, the 1-bit MLP significantly outperformed the 64-bit MLP ($p<.05$). These results showed that, for the task used in this study, it is possible to reduce memory usage by quantizing the connection weights to 1 bit without sacrificing the control performance of the MLP. One possible explanation for this outcome is that tuning the 1-bit binary weight vector may be less complex than tuning a 64-bit real-valued weight vector. Consequently, the quantized MLP may have been easier for ES to optimize effectively.

Figures 5 and 6 show the learning curves for the best, median, and worst trials out of 11 trials conducted under the same conditions. The ES configuration used in these experiments corresponds to (b) in Table 1. The number of hidden units is M=1 for Figure 6 and M=4 for Figure 7. In both figures, (i) and (ii) correspond to the results obtained with 1-bit and 64-bit MLPs, respectively. Comparing (i) and (ii) in Figure 6 revealed that both the 1-bit and 64-bit models required approximately 100 evaluations to reach a score of around 400, and both achieved scores of approximately 800 within 500 evaluations. The similarity in the shape of the learning curves between the 1-bit and 64-bit models suggests that reducing the bit width does not significantly alter the learning dynamics. As discussed earlier, this may be attributed to the fact that the binary genotypes are easier for ES to optimize, which compensates for the potential loss in representational capacity due to quantization. A similar trend is observed in Figure 7 for M=4, where the median trial in (i) closely resembles that in (ii). Notably, the best trial in Figure 7(i) significantly outperforms the best trial in Figure 7(ii). This result further supports the hypothesis that the binary genotypes are more amenable to optimization via ES, allowing for better exploitation of the MLP's potential compared to their 64-bit counterparts.



(i) 1bit (ii) 64bits

Figure 5. Learning curves of MLP with a single hidden unit (M=1).



(i) 1bit (ii) 64bits

Figure 6. Learning curves of MLP with four hidden units (M=4).

Supplementary videos[2,3] are provided which demonstrate the game screens played by the best MLP with 4 hidden units.

6. Conclusion

In this study, Evolution Strategy was applied to the reinforcement learning of neural network controllers for the Atari Space Invaders, where the connection weights in the neural network are not real numbers but binary (-1 or 1). The findings from this study are summarized as follows:

(1) For the number of hidden units 1, 2, 4, and 8, the game scores achieved by the trained 1-bit MLPs were not significantly lower than those achieved by the trained 64-bit MLPs. In the case of 16 hidden units, the trained 1-bit MLPs significantly outperformed the trained 64-bit MLPs. While representing connection weights with 1 bit reduces the capacity of the MLP compared to using 64-bit precision, it also makes optimization via ES easier. As a result, it was found that using 1-bit weights does not degrade the game-playing performance of the MLP while offering substantial memory savings.

[2] https://youtu.be/E1KSw1nox4o
[3] https://youtu.be/O4w8rXkeXlI

(2) When using 1-bit weights, the MLP with only a single hidden unit did not perform significantly worse than those with 2, 4, 8, or even 16 hidden units. Although quantizing connection weights to 1 bit is generally expected to reduce network capacity, potentially requiring a larger model to compensate, the result suggests that this compensation may not be necessary depending on the specific task.

(3) In training 1-bit MLPs using ES, it was found that increasing the number of offspring per generation was more effective than increasing the number of generations. This is likely because the inherent weakness of ES in global exploration can be mitigated by generating more offspring, thereby achieving a better balance between explorative and exploitative searches.

Future directions of this study include comparing ES with other evolutionary algorithms, such as Genetic Algorithm [7,8], and evaluating performances across other Atari games. Additionally, while this study used binary connection weights of -1 and 1, ternary neural networks employing -1, 0, and 1 have also been proposed. Applying ternary MLPs to the learning tasks in this study and comparing their performance with the binary MLP results reported here is another important avenue for future research.

References

[1] Okada, H. (2023). Evolutionary reinforcement learning of binary neural network controllers for Pendulum task — part1: evolution strategy. Preprints.org. doi: 10.20944/preprints202312.1537.v1

[2] Okada, H. (2024). Evolutionary reinforcement learning of binary neural network controllers for Pendulum task — part2: genetic algorithm. Preprints.org. doi: 10.20944/preprints202406.0933.v1

[3] Okada, H. (2024). Training neural networks with {-1,1} weights by evolution strategy. viXra.org. https://vixra.org/pdf/2410.0101v1.pdf

[4] Okada, H. (2025). Training Neural Networks with {-1,1} weights by genetic algorithm. viXra.org. https://vixra.org/pdf/2503.0107v1.pdf

[5] 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.

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

[7] Goldberg, D.E., Holland, J.H. (1988). Genetic algorithms and machine learning. Machine Learning 3, 95-99. doi: 10.1023/A:1022602019183

[8] Sastry, K., Goldberg, D., & Kendall, G. (2005). Genetic algorithms. in Burke, E.G. and Kendall, G. (eds.), Search methodologies: Introductory tutorials in optimization and decision support techniques, 97–125.