# Neural Networks with {-1,0,1} Weights Playing Atari Space Invaders (2) Trained by Genetic Algorithm

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

**Abstract:** Discrete-valued neural networks have attracted increasing attention because they require less memory and lower computational cost than conventional floating-point neural networks. While discrete weights are often obtained by quantizing pretrained continuous-valued networks, such approaches are not applicable to reinforcement learning tasks that cannot be trained by gradient-based methods. This paper investigates the training of multilayer perceptrons (MLPs) with ternary connection weights {−1,0,1} using Genetic Algorithm (GA) for the Atari Space Invaders reinforcement learning task. The performance of ternary-weight networks is compared with that of binary-weight networks {−1,1} under identical network topologies and GA configurations. Experimental results show that GA configuration employing a smaller population size and a larger number of generations significantly outperforms a configuration with a larger population size and fewer generations when the total number of fitness evaluations is fixed. Furthermore, no statistically significant difference is observed between ternary and binary weight representations in terms of game performance. Although ternary weights exhibit slightly better median and worst-case performance, binary weights occasionally achieve higher best scores while requiring less memory. These findings suggest that binary-weight neural networks provide an attractive trade-off between performance and memory efficiency, making them an attractive representation for evolutionary reinforcement learning.

**Keywords: evolutionary algorithm, genetic algorithm, ternary neural network, neuroevolution, reinforcement learning.**

## 1. Introduction

The connection weights of neural networks have been traditionally represented as floating-point values. In recent years, however, discrete-valued weights have been adopted to reduce memory requirements and to lower the computational cost of feedforward calculation. Typical examples of such discrete representations include binary values $\{-1,1\}$ and ternary values $\{-1,0,1\}$. An approach to discretizing neural network weights is to train the network using standard continuous-valued weights with gradient descent and then quantize the weights to low-bit representations after training. However, this approach is not applicable to learning tasks for which supervised learning is unsuitable.

In prior work, we have investigated reinforcement learning tasks and reported results on training neural networks with discrete connection weights, using evolutionary algorithms rather than gradient-based methods. For example, experimental results on the Atari *Space Invaders* task have been reported [1]. In the previous study, the connection weights were binary values $\{-1,1\}$ and ternary values $\{-1,0,1\}$, and the training algorithm was Evolution Strategy. In the present study, we evaluate the performance of Genetic Algorithm on the same task.

## 2. Atari Space Invaders

In the experiments presented in this article, as well as in our previous study [1], a neural network with discrete-valued connection weights is trained via reinforcement learning using an evolutionary algorithm to act as a controller for the Atari *Space Invaders* environment, with the objective of achieving as high a score

as possible. 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.

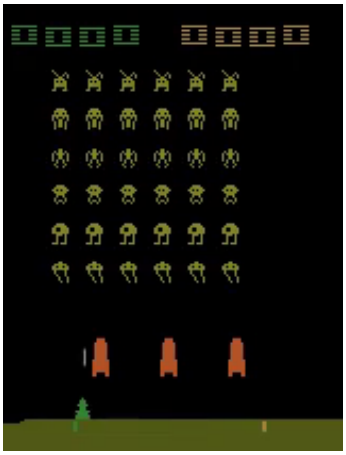

Figure 1. Screenshot of Atari *Space Invaders*.

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.

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 $210 \times 160$ 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, we employ the RAM-based environment, `ALE/SpaceInvaders-ram-v5` where `frameskip` is set to 1 (the default value is 4).

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.

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

## 3. Neural Networks with Ternary Connection Weights

The topology of the neural network used in this experiment is identical to that employed in our previously reported experiments [1]. However, whereas the connection weights in the previously reported experiments were restricted to binary values of $\{-1,1\}$, the present experiment employs ternary values $\{-1,0,1\}$. Both of the two studies employ a feedforward neural network, known as a multilayer perceptron (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 ternary, i.e., either of $-1$, $0$, or $1$. The unit activation function is the hyperbolic tangent (tanh) so that the network outputs real numbers within the interval $(-1.0, 1.0)$. The feedforward calculations are described in [2].

The MLP serves as the policy function: action(t) = F(observation(t)). Since we 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 (N=128 in Figure 2), where each value is normalized from $[0, 255]$ into $[-1.0, 1.0]$.

The game allows for six possible actions; therefore, the output layer is designed with six units (L=6 in Figure 2), and the action corresponding to the unit with the highest output value is selected as the network's decision.

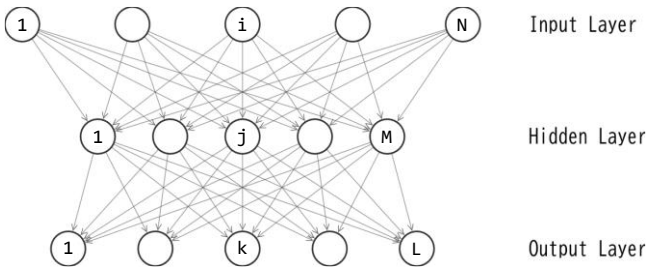


Figure 2. Topology of the MLP.

## 4. Training of Neural Networks with Ternary Connection Weights by Genetic Algorithm

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, we set N=128 and L=6, leading to D=135M+6. The training of this perceptron is essentially an optimization of the D-dimensional ternary 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 ternary variable, $x_i \in \{-1,0,1\}$. 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 ternary vector $\mathbf{x}$ is optimized using Genetic Algorithm [3,4]. GA handles $\mathbf{x}$ 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 phenotype of $\mathbf{x}$. Figure 3 illustrates the GA process. The process is the same as that in the previous study [5]. In Step 1, D-dimensional ternary vectors $\mathbf{y}^1, \mathbf{y}^2, \ldots, \mathbf{y}^P$ are randomly initialized where P denotes the population size. A larger value of P promotes exploration more. In Step 2,

values in each vector $\mathbf{y}^{\mathrm{p}}$ ($\mathrm{p} = 1,2,\ldots,\mathrm{P}$) are applied to the MLP and the MLP plays the game. The fitness of $\mathbf{y}^{\mathrm{p}}$ is then evaluated with the game score. In Step 3, the loop of evolutionary training is finished if a preset condition is satisfied. A simple example of the condition is the maximum number of fitness evaluations. Elites are the best E vectors among all offspring evaluated so far, where E denotes the size of elite population. To locally search around good solutions found so far, elite vectors are kept as parent candidates for the crossover. The elite population is empty at first. In Step 4, members in the elite population, $\mathbf{z}^1, \mathbf{z}^2, \ldots, \mathbf{z}^{\mathrm{E}}$, are updated. Step 5 consists of the crossover and the mutation. In Step 5.1, new P offspring vectors are produced by applying the crossover operator to the union of the elite vectors $\mathbf{z}^1, \mathbf{z}^2, \ldots, \mathbf{z}^{\mathrm{E}}$ and the vectors $\mathbf{y}^1, \mathbf{y}^2, \ldots, \mathbf{y}^{\mathrm{P}}$ in the current population. A single crossover with two parents produces two new offspring, where the two parents are randomly selected from the union of $\mathbf{z}^1, \mathbf{z}^2, \ldots, \mathbf{z}^{\mathrm{E}}$ and $\mathbf{y}^1, \mathbf{y}^2, \ldots, \mathbf{y}^{\mathrm{P}}$. To produce new P offspring vectors, the crossover is performed $\mathrm{P}/2$ times. Each vector in the union of $\mathbf{z}^1, \mathbf{z}^2, \ldots, \mathbf{z}^{\mathrm{E}}$ and $\mathbf{y}^1, \mathbf{y}^2, \ldots, \mathbf{y}^{\mathrm{P}}$ can be selected as a parent two or more times (several vectors in the union may not be selected any time). The new P offspring vectors replace the population $\mathbf{y}^1, \mathbf{y}^2, \ldots, \mathbf{y}^{\mathrm{P}}$. Crossover operators applicable to ternary chromosomes are a single-point crossover, a multi-point crossover and the uniform crossover. In Step 5.2, each element in the new offspring vectors is changed randomly under the mutation probability $pm$. A greater $pm$ promotes exploration more.

Step 1. Initialization
Step 2. Fitness Evaluation
Step 3. Conditional Termination
Step 4. Elitism
Step 5. Reproduction
  Step 5.1. Crossover
  Step 5.2. Mutation
Step 6. Goto Step 2

**Figure 3.** Process of Genetic Algorithm.

## 5. Experiment

### 5.1 Neural Network

Six configurations are employed for the number of hidden units; $\mathrm{M} \in \{1, 2, 4, 8, 16, 32\}$.

### 5.2 Genetic Algorithm

The total number of individuals evaluated in a single run of GA is given by $\mathrm{P} \times \mathrm{G}$, where P and G denote the population size and the maximum number of generations respectively. A larger P 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 are consistently configured such that $\mathrm{P} \times \mathrm{G} = 10{,}000$ (see Table 1). Configuration (a) emphasizes exploitation more than configuration (b), whereas configuration (b) emphasizes exploration more than configuration (a). The number of elites E is set to 20% of the population size P. The mutation probability pm is set to 1/D where D denotes the length of a ternary genotype vector.

**Table 1.** GA hyperparameter configurations.

| Hyperparameters | (a) | (b) |
|---|---|---|
| Population size (P) | 10 | 50 |
| Generations ($G$) | 1,000 | 200 |
| Fitness evaluations | 10×1,000=10,000 | 50×200=10,000 |
| Number of elites (E) | 10×0.2=2 | 50×0.2=10 |
| Mutation probability (pm) | 1/D | 1/D |

5.3 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 first applied to the 24 data points in Table 2(i)(a) and the corresponding 24 data points in Table 2(i)(b). This test revealed that configuration (a) outperformed configuration (b) with statistical significance (p=1.8e-3). The same test was next applied to the 24 data points in Table 2(ii)(a) and the corresponding 24 data points in Table 2(ii)(b). This test also revealed that configuration (a) outperformed configuration (b) with statistical significance (p=2.2e-3). Thus, for both ternary and binary weight representations, configuration (a) was found to outperform configuration (b). This can be attributed to the characteristics of GA, which is effective at global exploration but less effective at local exploitation. Because configuration (a) uses a larger number of generations than configuration (b), it is more likely to promote local exploitation in the later stages of the search. Consequently, configuration (a) can better compensate for the GA's weakness in local exploitation, thereby contributing more effectively to the improvement of the search performance than configuration (b). In contrast, configuration (b) employs a larger population size than configuration (a), which provides an advantage during the early stages of global exploration. However, because GA is inherently well suited to global exploration, this advantage of configuration (b) does not appear to have translated into a substantial improvement in search performance.

**Table 2.** Fitness scores among 11 runs.

(i) {-1,0,1} weights

| | M | Best | Worst | Average | Median |
|---|---|---|---|---|---|
| **(a)** | **1** | 1440 | 800 | 1024.5 | 945 |
| | **2** | 1350 | 800 | 1094.1 | 1145 |
| | **4** | 1560 | 800 | 1171.8 | 1140 |
| | **8** | 1505 | 800 | 1065.0 | 1020 |
| | **16** | 1635 | 850 | 1258.6 | 1210 |
| | **32** | 1600 | 800 | 1124.1 | 1140 |
| **(b)** | **1** | 1130 | 815 | 1010.5 | 1020 |
| | **2** | 1125 | 830 | 975.9 | 970 |
| | **4** | 1185 | 855 | 1040.5 | 1050 |
| | **8** | 1250 | 845 | 1049.1 | 1030 |
| | **16** | 1265 | 840 | 1049.5 | 1045 |
| | **32** | 1115 | 940 | 1037.3 | 1035 |

(ii) {-1,1} weights

| | M | Best | Worst | Average | Median |
|---|---|---|---|---|---|
| **(a)** | **1** | 1300 | 900 | 1023.2 | 975 |
| | **2** | 1440 | 800 | 1043.6 | 1060 |
| | **4** | 1785 | 850 | 1165.9 | 1105 |
| | **8** | 1390 | 800 | 1055.5 | 1085 |
| | **16** | 1695 | 800 | 1233.6 | 1165 |
| | **32** | 1705 | 800 | 1140.9 | 1110 |
| **(b)** | **1** | 1240 | 805 | 1029.1 | 1020 |
| | **2** | 1200 | 855 | 1019.1 | 1070 |
| | **4** | 1215 | 800 | 1008.6 | 995 |
| | **8** | 1245 | 885 | 1027.3 | 1000 |
| | **16** | 1140 | 855 | 1004.1 | 975 |
| | **32** | 1125 | 975 | 1042.7 | 1040 |

Next, we compare the performance of the $\{-1,0,1\}$ weights with that of the $\{-1,1\}$ weights. The Wilcoxon signed-rank test was first applied to the 24 data points in Table 2(i)(a) and the corresponding 24 data points in Table 2(ii)(a). This test revealed that no statistically significant difference was observed between the $\{-1,0,1\}$ weights and the $\{-1,1\}$ weights when configuration (b) was adopted (p=0.56). The

same test was next applied to the 24 data points in Table 2(i)(b) and the corresponding 24 data points in Table 2(ii)(b). This test also revealed that no statistically significant difference was observed between the $\{-1,0,1\}$ weights and the $\{-1,1\}$ weights when configuration (b) was adopted (p=0.60). Thus, given an identical neural network topology, the $\{-1,1\}$ weights did not significantly underperform the $\{-1,0,1\}$ weights for the task examined in this experiment. Since binary weights require less memory than ternary weights, the $\{-1,1\}$ weights are more advantageous when both performance and memory requirements are considered. This result is consistent with the previous study [1] in which Evolution Strategy was adopted.

Figure 5 shows the learning curves for the best, median, and worst trials out of 11 trials conducted under the same conditions where the weights are $\{-1,0,1\}$, and Figure 6 shows the learning curves in the same manner where the weights are $\{-1,1\}$. The GA configuration is (a) in Table 1. The number of hidden units is M=16 for both Figure 5 and 6. Note that the horizontal axis represents the number of evaluations on a logarithmic scale in these figures.

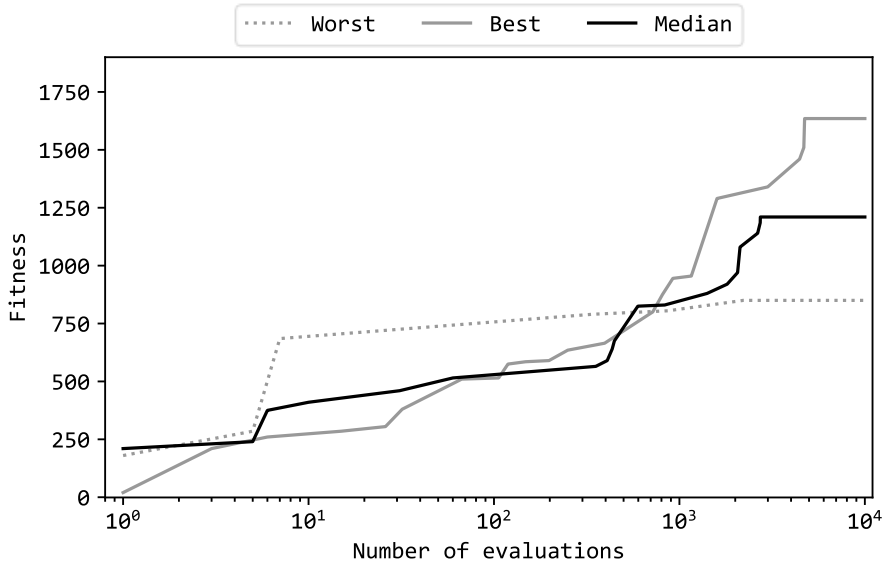


**Figure 4**. Learning curves of MLP with {-1,0,1} weights and 16 hidden units.

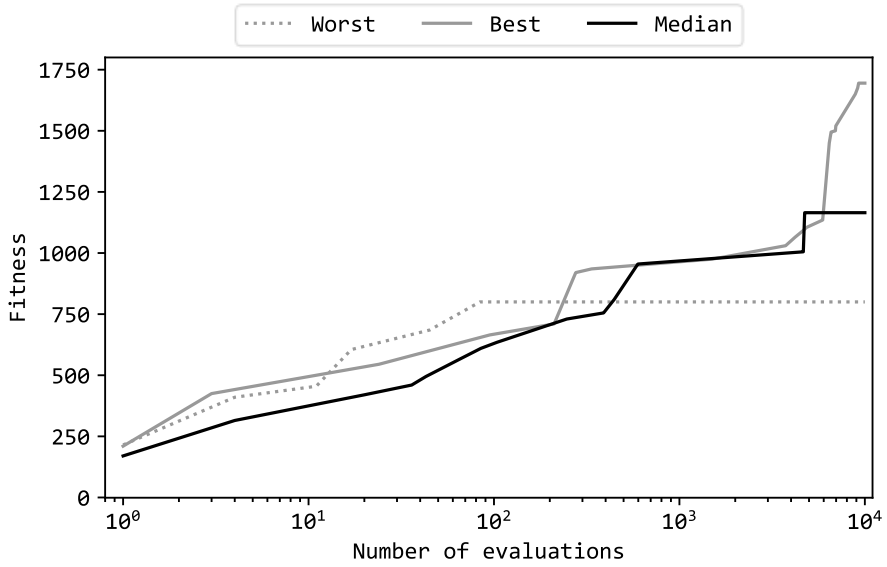


**Figure 5**. Learning curves of MLP with {-1,1} weights and 16 hidden units.

Figure 5 demonstrates that the fitness generally improves as the number of evaluations increases, indicating that the evolutionary algorithm successfully evolves better neural networks over time. In the best run, the fitness increases dramatically after approximately 1,000 evaluations and eventually reaches more than 1,600. This indicates that the ternary weight representation is capable of discovering highly effective solutions. In the median run, the fitness improves gradually throughout the search process and finally reaches approximately 1,200. Several significant jumps in fitness are observed, suggesting that the algorithm escapes local optima and discovers better solutions during evolution. In the worst run, the fitness rises rapidly to around 700 in the early stage, but only slight improvements are observed afterward, and the final fitness remains around 850. Overall, these results indicate that the ternary weight representation can achieve high-quality solutions, although the performance varies considerably among different runs.

Figure 6 also demonstrates that the fitness generally increases as the number of evaluations increases. In the best run, the fitness improves sharply during the later stage of evolution and finally reaches approximately 1,700, which is the highest fitness obtained among all runs. In the median run, the fitness increases relatively steadily from the beginning and reaches approximately 1,150 by the end of the search. A noticeable improvement is also observed in the middle stage, indicating successful evolutionary progress. In the worst run, the fitness improves to approximately 800 before stagnating, with little improvement during the remainder of the search. These results suggest that the binary weight representation is also capable of producing highly fit solutions, although some runs converge prematurely and fail to improve further.

Comparing Figures 5 and 6 reveals the following differences between the ternary and binary weight representations. First, the binary representation achieves a slightly higher maximum fitness (1,695) than the ternary representation (1,635). Second, the ternary representation achieves a slightly higher median fitness (1,210), suggesting better average performance. Third, the worst-case performance of the ternary representation is also slightly better (850), indicating greater robustness across different runs.

Both representations exhibit substantial improvements after approximately 1,000 evaluations. However, the ternary representation includes the additional weight value 0, which effectively disables unnecessary connections and provides greater flexibility in the network structure. This additional degree of freedom may contribute to the better median and worst-case performance. On the other hand, the binary representation has a smaller search space. As a result, it can occasionally discover high-quality solutions, leading to the highest fitness among all runs. However, its limited flexibility may also cause premature convergence in some runs. In summary, the experimental results suggest that the binary weight representation is superior in terms of maximum achievable performance, whereas the ternary weight representation provides greater stability and robustness across multiple runs. Supplementary videos[2-5] are provided which demonstrate the game screens played by the trained/untrained MLPs.

## 6. Conclusion

This study investigated the training of multilayer perceptrons with ternary connection weights $\{-1,0,1\}$ for the Atari Space Invaders task using Genetic Algorithm. The performance of the ternary-weight networks was compared with that of binary-weight networks $\{-1,1\}$ under the same neural network topology and GA configurations. Experimental results demonstrated that, for both weight representations, GA configuration

[2] https://youtu.be/e6CBHD-osbs ({-1,0,1} weights, 16 hidden units, before the training)
[3] https://youtu.be/hsaSPabeOAQ ({-1,0,1} weights, 16 hidden units, after the training)
[4] https://youtu.be/29xJC2Fi8VQ ({-1,1} weights, 16 hidden units, before the training)
[5] https://youtu.be/3aXSExa25_8 ({-1,1} weights, 16 hidden units, after the training)

with a smaller population size and a larger number of generations consistently outperformed the configuration with a larger population size and fewer generations. This result suggests that increasing the number of generations is more beneficial than increasing the population size when the total number of fitness evaluations is fixed. The comparison between ternary and binary weight representations revealed no statistically significant difference in game performance. Although the ternary representation showed slightly better median and worst-case performance, while the binary representation occasionally achieved a higher best score, the overall performance of the two representations was comparable. Considering that binary weights require less memory than ternary weights, the binary representation provides a more favorable trade-off between performance and memory efficiency for the Atari Space Invaders task.

These findings are consistent with our previous study [1] employing Evolution Strategy, suggesting that the advantage of binary-weight neural networks is not limited to a particular evolutionary algorithm. Future work includes evaluating the proposed approach on additional reinforcement learning tasks, investigating deeper neural network architectures, and applying other evolutionary optimization methods to further clarify the relationship between discrete weight representations and evolutionary learning performance.

## References


[1] Okada, H. (2026). Neural networks with {-1,0,1} weights playing Atari Space Invaders (1) Trained by Evolution Strategy. viXra.org. https://vixra.org/pdf/2602.0030v1.pdf

[2] Okada, H. (2022). Evolutionary reinforcement learning of neural network controller for Pendulum task by Evolution Strategy, International Journal of Scientific Research in Computer Science and Engineering, 10(3), 13–18.

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

[4] Holland, J. H. (1992). Genetic algorithms. Scientific American, 267(1), 66-73.

[5] Okada, H. (2025). Binary neural networks playing Atari Space Invaders (2) Trained by Genetic Algorithm. viXra.org. https://vixra.org/pdf/2511.0019v1.pdf